The Visual Computer (2019) 35:1027–1039
https://doi.org/10.1007/s00371-019-01671-0

ORIGINAL ARTICLE

Depth-aware image vectorization and editing

Shufang Lu[1] · Wei Jiang[2] · Xuefeng Ding[1] · Craig S. Kaplan[4] · Xiaogang Jin[3] · Fei Gao[1] · Jiazhou Chen[1]

Published online: 7 May 2019

Abstract
Image vectorization is one of the primary means of creating vector graphics. The quality of a vectorized image depends crucially on extracting accurate features from input raster images. However, correct object edges can be difficult to detect when color gradients are weak. We present an image vectorization technique that operates on a color image augmented with a depth map and uses both color and depth edges to define vectorized paths. We output a vectorized result as a diffusion curve image. The information extracted from the depth map allows us more flexibility in the manipulation of the diffusion curves, in particular permitting high-level object segmentation. Our experimental results demonstrate that this method achieves high reconstruction quality and provides greater control in the organization and editing of vectorized images than existing work based on diffusion curves.

Keywords Image vectorization · RGB-D images · Depth aware · Diffusion curves · Object segmentation and editing

1 Introduction

Vector graphics are a popular means of representing visual information and offer several advantages over raster images. They are resolution independent and can be scaled to any size without loss of quality. They often have a very compact representation, allowing them to be stored and transmitted efficiently. They also support high-level editing operations on geometric primitives as opposed to pixels. Vector illustrations can easily be authored directly by an artist, generated procedurally, or computed as an approximation of a raster image, a process known as *vectorization*. Vectorization is an important tool in art and graphic design, as it makes a vast universe of expressive photographic content available in a vector graphics context.

Diffusion curves [17] are a recent, high-level vector graphics primitive. A single diffusion curve is a vector path augmented with color samples, each of which is associated with a position on the curve and restricted to the curve's left or right side. In a diffusion curve image (DCI), composed from a set of diffusion curves, the colors diffuse outward from the curves to endow every point in the canvas with an interpolated color. Diffusion curves have been used both as drawing primitives in handmade illustrations and as the building blocks of automated vectorization algorithms. However, vectorization with diffusion curves faces several challenges. First, as with other vectorization algorithms, success depends on robust and accurate edge detection, a perennial problem in image processing. Edges can be difficult to extract from color images, particularly in areas of roughly constant color. Object contours can also be obscured by color variation caused by texture or lighting, especially in detailed photographs (Fig. 3). Second, it can be difficult to edit the resulting vector images at a high level of abstraction. Unlike mesh-based [12] and patch-based [31] vector graphics representations, a DCI consists of a sparse set of disconnected paths, making it difficult to extract and manipulate the subset of paths that contribute to an object in the image. Current techniques are typically limited to curve-level editing [9,11,17].

An RGB-D camera produces both a standard RGB image and a depth map, which reports the depth of every pixel in the image. Several inexpensive RGB-D sensors, such as Microsoft's Kinect, Apple's TrueDepth, and Google's Project Tango, are now available, prompting new research on the use of depth information in standard computer vision algorithms. For example, recent work has used depth to

✉ Jiazhou Chen
cjz@zjut.edu.cn

1 Zhejiang University of Technology, Hangzhou, China

2 University of Victoria, Victoria, BC, Canada

3 Zhejiang University, Hangzhou, China

4 University of Waterloo, Waterloo, ON, Canada

improve the performance of image segmentation [6] and 3D modeling [15].

In this paper, we propose a novel image vectorization technique, based on diffusion curves, which incorporates depth information to address the challenges mentioned above. The depth map provides valuable cues for segmentation and is used to generate the basic contours for objects in the image. Color information is used to extract multi-scale edges. The contour and color edges are then combined and used to generate a set of diffusion curves. We improve the quality of vectorized images by taking advantage of depth information. With depth, we can more easily construct diffusion curves that align with important image features and object contours. Moreover, our proposed method supports object-level editing and processing. We can select an object in an image by using the depth-aware image segmentation. The extracted object can be vectorized to generate the diffusion curves associated with it. This curve–object relationship raises the efficiency of vector image editing, as the user can potentially manipulate the dozens or hundreds of curves making up an object en masse rather than one at a time.

To the best of our knowledge, our work is the first application of RGB-D images in vectorization. We expect that low-cost RGB-D sensors will be deployed widely in home and mobile consumer devices in the near future, and so the time is right to develop depth-aware variants of standard graphics algorithms like vectorization.

2 Related work

The earliest vector graphics systems could represent only the outlines of shapes. Solid color and gradient fills later became standard and permitted the compact representation of a wide range of expressive, scalable images. More recently, research has sought to increase the expressive range of colors and textures in vector graphics via methods that can be classified as patch-based [13,31], mesh-based [12,29], and curve-based [32].

Curves are widely used in computer science to describe objects. Diffusion curves, first proposed by Orzan et al. [17], represent an image by a set of curves augmented with color samples along their lengths. The color samples on either side of a curve diffuse across the entire image canvas smoothly, a process that can be modeled by solving a Poisson equation. Diffusion curves are a flexible primitive that can be used to vectorize realistic images. However, the work of Orzan et al. had difficulty with natural images or digital photographs with rich details. To improve the fit between a raster image and the colors computed from a set of diffusion curves, Jeschke et al. [11] extended the diffusion curve representation with more accurate color and texture attributes. In order to produce photorealistic effects, Hou et al. [7] extended the diffusion curves by adding two new geometric primitives including Poisson curves and Poisson regions. Several other papers use a bi-Laplacian formulation of diffusion curves in order to provide more control over color diffusion from boundary curves [2,4,8]. Xie et al. [32] proposed an automatic vectorization method based on extracting hierarchical diffusion curves in both the Laplacian and bi-Laplacian domains. Although their work achieves a high-quality reconstruction both for vector art and for natural images, their diffusion curves and colors are mostly static and cannot be edited later, apart from tasks such as detail removal and stylization. By minimizing a measure of the color reconstruction residual, Zhao et al. [35] introduced a solution to the inverse diffusion curve problem grounded on the theory of shape optimization. However, this method focuses on the reconstruction error rather than the curve editability; it cannot enable easy editing of generated diffusion curves. Sun et al. [28] proposed a fast multipole representation for diffusion curve images by introducing a new vector graphics primitive called *diffusion points*; their technique can render millions of diffusion curves in real time. Jeschke [9] introduced a generalized and improved diffusion curve representation based on a spatial blending of multiple Laplace functions and a new edge blur formulation. However, these diffusion curves are defined manually by users, not extracted automatically from an image, making the technique unsuitable for vectorization of detailed real-world images.

Current diffusion curve image representations consist of relatively sparse sets of disconnected curves, each of which must be edited individually rather than at the level of the objects in the image. Although Xie et al. [32] employed a hierarchy of multiresolution diffusion curves to make their results more editable, their method was limited to rendering at multiple levels of visual abstraction and could not support shape and color editing. Our method combines color and depth information and is therefore capable of defining diffusion curves that align accurately with both important image features and object contours. As a result, we can edit vector images at a higher semantic level than what was previously possible, by recognizing the relationship between an object and the diffusion curves that represent it.

As reliable and affordable RGB-D sensors become widely available, new work has utilized depth information to simplify common challenges in computer vision and computer graphics, in both 2D and 3D. KinectFusion [15] creates high-quality 3D models in real time from the live depth data captured by a moving Kinect camera. Song and Xiao presented deep learning techniques for 3D object detection in RGB-D images [25,26], demonstrating improved performance and quality over state-of-the-art methods based on RGB images alone. Ge et al. introduced a 3D hand pose estimation method, using multi-view CNNs

with depth cues, to fully recover 3D information about hand joints [5].

In some work, depth information is simply treated as an extra channel appended to the three color channels in an image, providing an additional source of information within the framework of existing image processing algorithms. Gupta et al. proposed a method for edge detection and image segmentation by combining RGB image and depth information effectively [6]. Shen et al. presented a depth-aware single-image seam carving method based on images captured by a Kinect sensor [24]. The results are improved by calculating the seam carving energy from both the RGB color image and its corresponding depth map. Sun et al. proposed a layered RGB-D scene flow estimation method by decomposing a scene into moving layers ordered in the depth map [27]. Wang et al. developed a multi-camera system to generate a high-resolution color image with an accurate depth map. It reproduces many of the effects possible with light-field cameras, including image defocus [30]. Inspired by these methods, we propose to use depth information from RGB-D data to improve the quality of image vectorization.

3 Image vectorization

Given a color image and its depth map at same resolution, our goal is to compute a diffusion curve image that closely approximates the input image. Because color and depth have different behavior and operate at different scales, we extract color edges and depth edges via two distinct methods and combine the edge information afterward. For the color image, we use a multi-scale Canny edge detector to extract edges. For the depth map, we first use a trilateral filtering algorithm to recover any missing depth information and then use a cartoon edge detector on the recovered depth map. Because we extract curves from the color and depth images independently, the more abundant color edges can often overlap depth edges. In order to preserve object contours found via depth, we subtract these contours from the color edges, leaving behind curves representing image details and texture. We then use the two sets of edges to construct diffusion curves. As always, the final vector image can be rendered by solving the Poisson equation with boundary conditions defined by the diffusion curves. The entire process is visualized in Fig. 1.

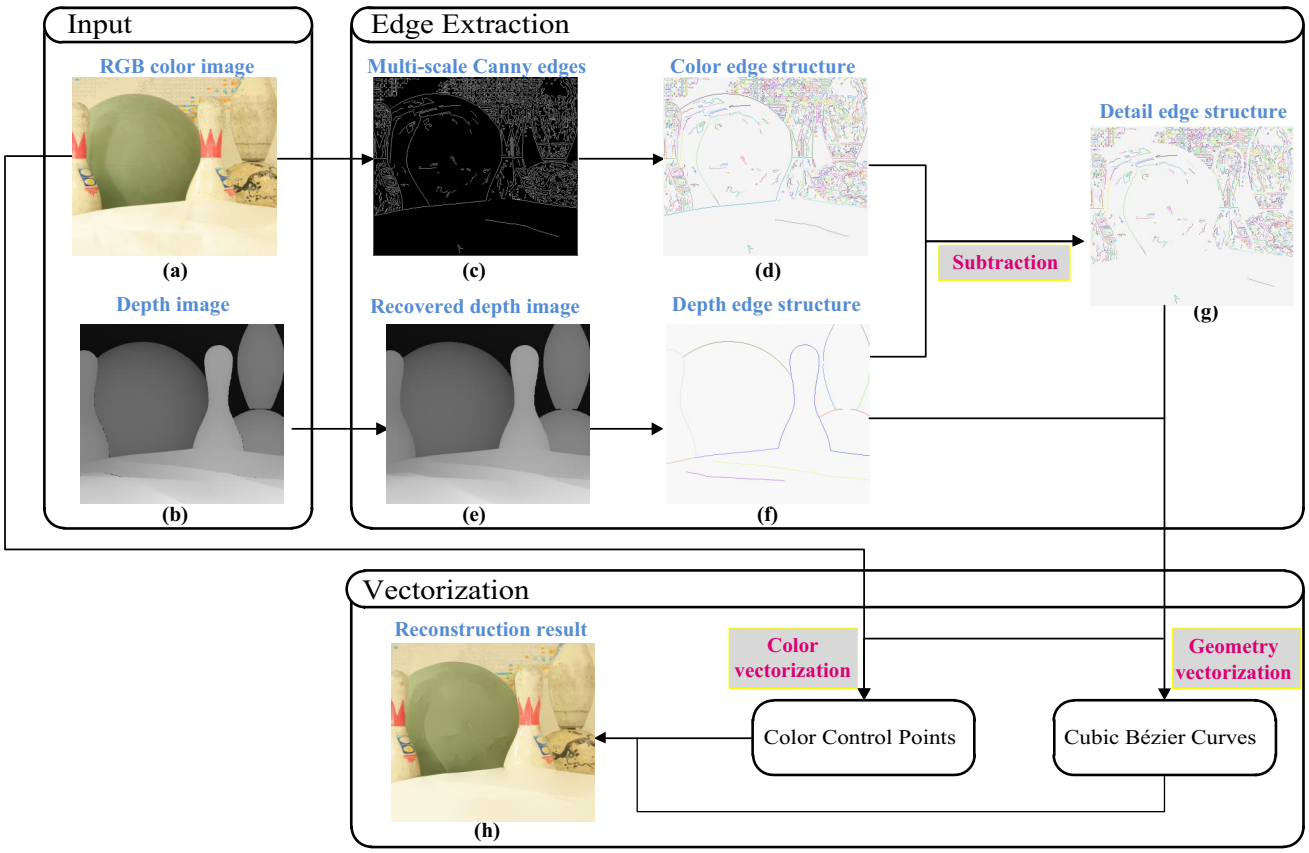

Fig. 1 An overview of our process for vectorization of an RGB-D image

3.1 Edge extraction

3.1.1 RGB edge extraction

Scale invariance is frequently cited as an intrinsic property of real-world images [21]. In particular, multi-scale structures have been observed in the statistics of image edge detection for a long time [14,18]. In order to get the richest possible edge information, we apply Canny edge detection in a Gaussian scale space derived from the input image. Following Orzan et al. [16], we apply a sequence of Gaussian blurs with increasing variances to generate a set of images with features at ever broader spatial scales. We then apply Canny edge detection on each scale image, producing a hierarchy of edges.

Canny edge detection produces a binary bitmap that identifies edge pixels, as shown in Fig. 1c. From this bitmap, we must extract long piecewise linear paths representing coherent image edges, which will drive subsequent vectorization steps. This process is challenging: A given edge pixel may have three or more neighbors in the bitmap, particularly in regions of high detail, making it difficult to decide which paths to extract. In order to find long connected paths while avoiding small noisy edges, we apply a gradient-guided search algorithm. First, select a random edge pixel p and compare its gradient direction $\mathbf{t}_p$ with those of the eight pixels q_i in the 1-ring of p, denoted Ω_p. Of these eight pixels, we consider the subset that belong to the edge bitmap and select a connecting neighbor q^* whose gradient direction aligns most closely with the gradient direction of p, provided that the difference between the two gradient directions is less than 45 degrees. That is, we are searching for

$$q^* = \arg\max_{q_i \in \Omega_p} \mathbf{t}_p \cdot \mathbf{t}_{q_i}, \quad \mathbf{t}_p \cdot \mathbf{t}_{q^*} > \sqrt{2}/2. \tag{1}$$

If a suitable q^* is found, we connect p and q^*. Otherwise, we expand the search area to the 16 pixels in the 2-ring of p. If a pixel q^* can be found in this 2-ring satisfying the conditions above, we connect p and q^* via the pixel that lies between them. Once these potential connections are considered for a pixel p, we repeat the process on other edge pixels, ultimately producing a set of paths E_{color}. Figure 1d shows the resulting connected paths, each one drawn in a different color.

3.1.2 Depth edge extraction

The depth map provided by a sensor such as that in a Kinect may contain many artifacts, including undersampling, missing samples, and additive noise [33]. Therefore, before extracting edges, we first process and filter the depth map to suppress these artifacts. We use the progressive color-guided trilateral filtering method presented by Ye et al. [34]. Their method first dilates regions with missing depth samples to eliminate potentially unstable depth estimates at the boundaries of these regions. Then, they adopt an outside in-depth recovery strategy that iteratively computes depth estimates at boundaries until all holes are filled. Because depth and color information are strongly correlated in capturing geometric structure in a scene, we use RGB edges, as described in Sect. 3.1.1, to guide depth recovery. Our trilateral filtering algorithm considers rough depth variance, spatial distance, and color variance. Given a depth image D, a color image I, and the piecewise polygonal paths E_{color} extracted from the RGB image, we can estimate the depth D_p for a boundary pixel p with missing depth information as follows:

$$D_p = \frac{1}{\omega} \sum_{y \in \Omega(p)} G_\lambda(p-q) G_\xi(\hat{D}_p - D_q) G_\mu(I_p - I_q) D_q. \tag{2}$$

Here, ω is the normalization factor, $\Omega(p)$ is the subset of neighboring pixels of p with valid depths and that are not separated from p by a path in E_{color}, $\hat{D}_p$ is the initially estimated depth of D_p by averaging immediate valid neighbors of p, and I_p is the color at pixel p. The filter is composed from three main factors: the spatial distance $G_\lambda(p-q)$, the depth difference $G_\xi(\hat{D}_p - D_q)$, and the color difference $G_\mu(I_p - I_q)$. In each case, $G_\sigma(\cdot)$ is a Gaussian kernel with standard deviation σ. As depth values are estimated, they are added to the set of known depths, shrinking regions with missing information, until all missing areas are filled. Figure 1b shows an initial depth map, with the corresponding recovered result shown in Fig. 1e. Note that missing depth values, shown as black spots in Fig. 1b, are smoothed over while preserving the 3D geometric structure of the original scene. Please refer to Fig. 7 for more depth recovery results.

A depth map bears some resemblance to a cartoon image: It consists of large regions of constant or slowly varying color, divided by sharp edges. Based on this observation, we extract depth edges using an enhanced version of Cheng's technique for cartoon edge detection [3]. Cheng's algorithm consists of two steps. First, non-maximal suppression is applied to the second derivative of the input image to identify curve pixels. Second, edge pixels are linked to form long curves, while removing unreliable curves made from fewer than three pixels. The paths produced by this algorithm can contain some undesirable artifacts, as shown in Fig. 2. The output of Cheng's algorithm is shown in Fig. 2a, followed by close-ups of the highlighted boxes in (c) and (e). Note that some short edges are left disconnected from nearby longer ones, as in the two adjacent edges in (c). Also, in some places nearly identical curves are drawn side-by-side, as in (e). Starting with the output of Cheng's algorithm, we connect short curves to

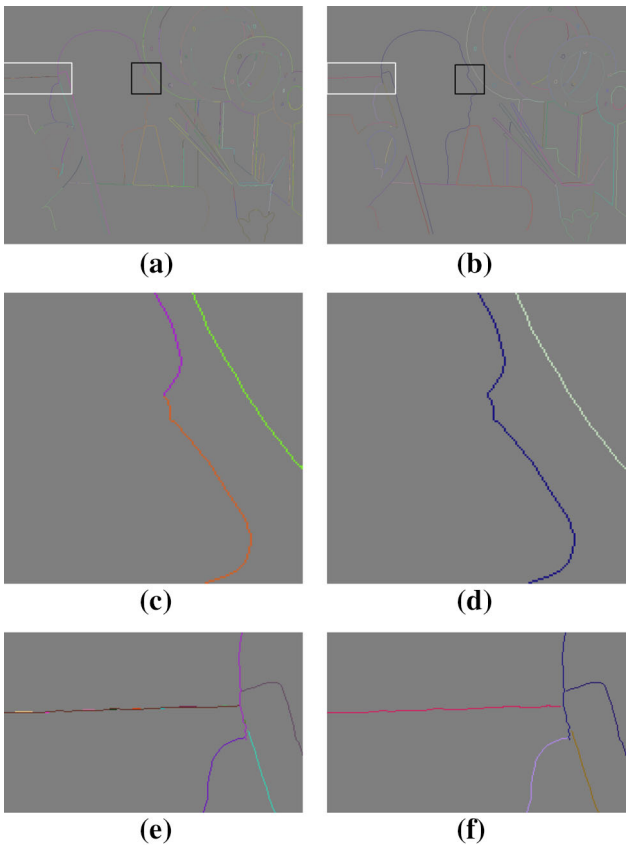

Fig. 2 A comparison of the out from Cheng's cartoon edge detection algorithm [3] and our method. Given the input depth map from the top row of Fig. 7, Cheng's method produces the curves in (**a**) and our method produces the curves in (**b**). The images in **c** and **e** close-ups of the boxed regions in (**a**), (**d**) and (**f**) are close-ups of (**b**)

long (ideally closed) curves and merge doubled curves that run side-by-side. To form long curves, we check the distance between the endpoints of two piecewise linear paths and the difference in depth values at these end points to determine whether they can be connected. If the distance is less than 1.5 pixels and the difference in depths is below a threshold (15 units in the depth map's luminance channel), we connect the two paths. As shown in Fig. 2d, the short curves in Fig. 2c are connected to form a longer curve. Also, if a short curve lies within one pixel of a longer curve along its normal direction, we merge the short curve into the longer curve, allowing us to merge the multiple curves shown in Fig. 2e into the single red curve in Fig. 2f. We refer to the resulting set of extracted depth paths as E_{depth}.

3.1.3 Edge subtraction

When depth discontinuities in a scene occur in places of high color contrast, edges detected in the depth map will also be visible in the color image. However, many additional color edges, marking texture or lighting boundaries on object surfaces, will not be detected in the depth map. In most cases, depth edges delineate object outlines more clearly and robustly than color edges. Thus, we use E_{depth} as a coarse starting point for vectorization and add E_{color} to capture details. However, our vectorization process will perform poorly in the presence of overlapping edge paths. Therefore, we subtract E_{depth} from E_{color}, removing any color edges that are also depth images. Before finding paths as in Sect. 3.1.1, if p is a pixel belonging to E_{depth}, we modify the detected Canny edge pixels by removing pixels in a 3×3 region centered at p. We refer to the resulting paths as E_{detail}, as shown in Fig. 1g.

Figure 3 shows the edges extracted from an RGB color image and its corresponding depth map. As shown in the boxed region in Fig. 3b, it is difficult to extract the contours of the chair's arm from the color image because its pixels are too close in color to those of the floor behind it. By comparison, the contours extracted from the depth image, shown in Fig. 3d, are much more robust.

3.2 Diffusion curve generation

Given the detail paths E_{detail} and depth paths E_{depth} computed from an RGB-D image, our goal is to generate a diffusion curve image that closely approximates the colors of the source RGB image. The procedure of generating diffusion curves involves consideration of both geometry and color information. To vectorize geometry, we fit piecewise smooth cubic Bézier curves in a least-squares sense to the extracted polygonal paths [23]. In order to turn this geometry into a diffusion curve, we must then compute a set of color samples extracted from RGB data.

The paths in E_{detail} and E_{depth} record sharp color transitions or object boundaries. We generate color samples by starting with a dense set of samples along both sides of the path and then simplifying to match colors while avoiding overfitting. For every pixel p on a curved path, let l be the point three pixels away from p in the curve's normal direction, on the left side of the curve. Similarly, let r be three pixels away from p to the right of the curve. We sample the image in a 3×3 neighborhood at l and r to obtain color samples on the left and right sides of the curve at p. However, this neighborhood may be crossed by other paths in regions of high curvature or high detail, and we do not want our sampled color to include information from across these potentially sharp boundaries. We therefore define a neighborhood N_l, consisting of the subset of the pixels in the 3×3 neighborhood around l that are not separated from p by any other paths in the neighborhood. Define the neighborhood N_r similarly.

We then use a modified mode filter at l and r to generate the two color samples for the pixel p. We explain the process for N_l; the same steps hold for N_r. Let the pixels of N_l be

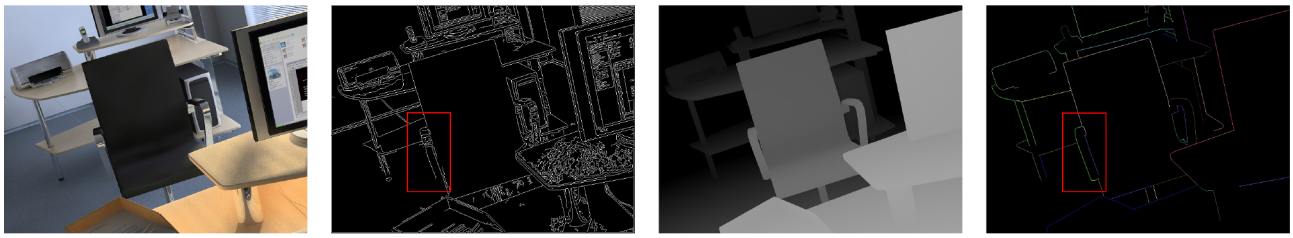

Fig. 3 Edges extracted from color and depth images, **a** is the color image, **b** is the result of RGB edge extraction, **c** is the depth image, and **d** is the result of depth edge extraction

denoted $\{I_0, I_1, \ldots I_{n-1}\}$, where $n \leq 9$. Let S be an initially empty collection of pairs (S_j, ω_j), where S_j is a color and ω_j is an integer weight. For every pixel color I_i, compare it with all the candidate colors stored in S. If the distance between I_i and S_j is less than 50 CIE L*a*b* units, update S_j to a new value $\frac{S_j+I_i}{2}$ and increment its weight ω_j. Otherwise, add the new pair $(I_i, 1)$ to S. This iteration yields a set of candidate colors with associated weights. We create a color sample at p by taking the color with the maximal weight. The pseudocode is summarized in Algorithm 1.

Once we have left-hand and right-hand color samples stored at every pixel along a path, we thin the samples out to avoid overfitting. Specifically, we walk over the samples on one side of the curve and discard any sample that is within 10 CIE L*a*b* units of the previous preserved sample. We then repeat the process with the samples on the other side of the curve.

Algorithm 1: Calculate a modified mode for a set of colors $\{I_0, \ldots, I_n\}$

```
S = ∅
for I_i in {I_0, ..., I_n} do
    for (S_j, ω_j) in S do
        if (||I_i − S_j|| < 50) then
            S_j ← (S_j + I_i)/2
            ω_j ← ω_j + 1
            continue i
        end if
    end for
    S.add( (I_i, 1) )
end for

(S_max, ω_max) = (S_0, ω_0)
for (S_j, ω_j) in S do
    if (ω_j > ω_max) then
        (S_max, ω_max) = (S_j, ω_j)
    end if
end for

return S_max
```

4 Object-level editing

The ability to edit vector graphics at an abstract, geometric level remains one of the reasons for its popularity [19]. However, diffusion curve images typically consist of a sparse set of disconnected curves, which are not necessarily associated with the semantics of the drawing they represent. Consequently, editing a coherent object in a diffusion curve image may require modifications to a large number of individual curves, and the effects on the final image may be hard to control or predict.

The depth information that powers our vectorization algorithm also gives us greater ability to detect and edit whole objects. By design, our depth-based diffusion curves are aligned with object boundaries. In this section, we describe a suite of tools for flexible object segmentation and editing of images.

Before editing, the object must be segmented from the original image. We use the two-stage depth-aware Grabcut algorithm for this purpose. Firstly, the segmentation begins when the user draws a rectangular region on the canvas, containing the object of interest. The user then manually adjusts parameters controlling the range of depths that make up the object, which will flag a subset of the pixels in the rectangle (Fig. 4a). The user can adjust the rectangle extents and depth range to define a selection region that approximates the object of interest as closely as possible. An initial depth mask is generated with its pixels marked with two flags denoting probable foreground and sure background. The probable foreground is marked in red color, while the rest sure background is marked in black color. Then, we use this initial depth mask and the RGB image (Fig. 4e) as the input for the Grabcut [20] algorithm and get the initial segmentation result (Fig. 4b). However, the segmentation result greatly depends on the depth mask. For some images, we can get quite good segmentation results after the first stage, such as the bowling pin in Fig. 9 and the plaster cast in Fig. 10. But in some complicated examples, some foreground pixels have similar or same depth value with its nearby background pixels. As shown in Fig. 4a, the bottom region of the rectangle which covers some parts of the stuff animal and the cardboard box

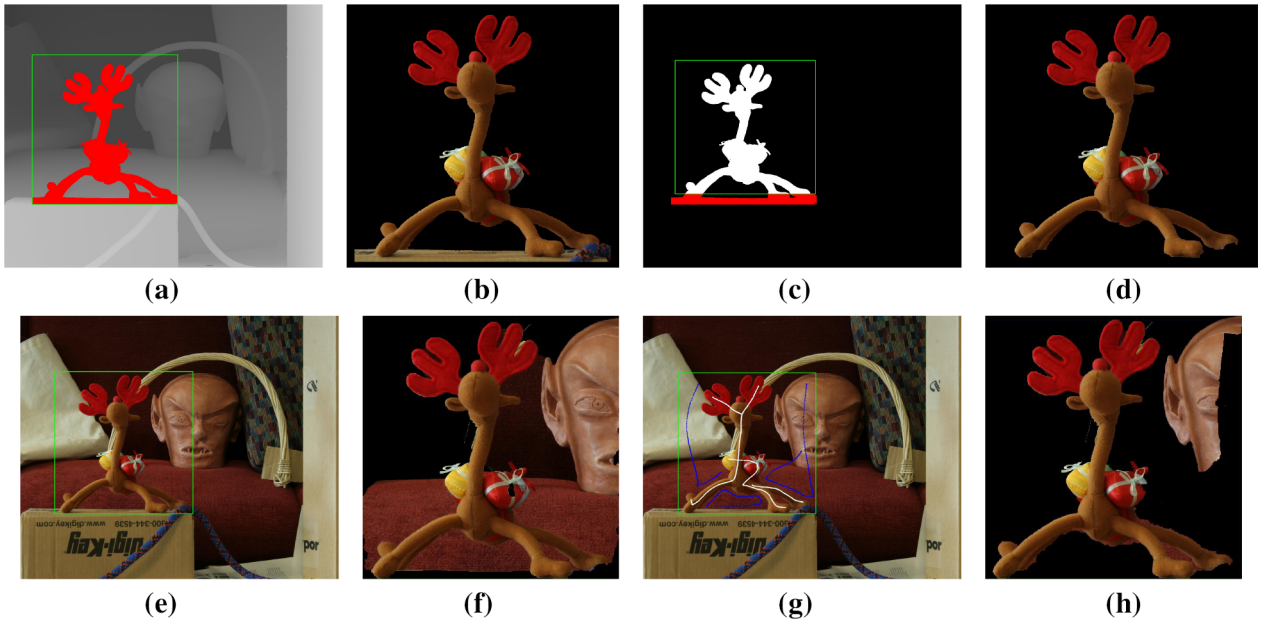

Fig. 4 A demonstration of depth-aware object selection algorithm and its comparison with Grabcut method [20]. The user draws a rectangle and adjusts the depth range to select the red region and generate an initial depth mask in (**a**), the initial segmentation result is generated in (**b**), the depth mask is refined in (**c**), and the final segmentation result is in (**d**). **f** and **h** are the Grabcut [20] results based on a bounding box interface in (**e**) and more precise user seeds interfaces in (**g**), respectively

Table 1 Parameters and flags of two-stage depth-aware Grabcut algorithm

Min_depth	The minimum depth threshold
Max_depth	The maximum depth threshold
Red_flag	Probable foreground pixels
Black_flag	Sure background pixels
White_flag	Sure foreground pixels

almost have same depth, we cannot get a good initial depth mask by adjusting the depth range. As a result, the top part of the cardboard box has come to the initial segmentation result which we do not want (see the bottom area of Fig. 4b). To this end, we perform the second stage which refines the initial depth mask. Specifically, we draw a rectangular region on the initial depth mask and mark those probable foreground pixels located in this rectangle as sure foreground area (Fig. 4c). Therefore, the probable foreground pixels in the initial depth mask (Fig. 4a) are further divided into probable foreground pixels and sure foreground pixels (marked in white color). Then, we use this refined mask for the Grabcut algorithm and obtain the final segmentation result (Fig. 4d). Table 1 shows the parameters and flags used in our two-stage depth-aware Grabcut method.

The comparison between our segmentation algorithm and the previous Grabcut method [20] is shown in Fig. 4. Given the bounding box in Fig. 4e, we can see that the segmentation result in Fig. 4f) is not satisfactory. Even with more precise user seeds interfaces (Fig. 4g), the improved result in Fig. 4h is not as good as the result of our method (Figure 4d). Moreover, our depth-aware Grabcut algorithm is much more efficient than the previous Grabcut algorithm. Because the proposed depth mask defines a selection region that is very close to the target object, all our segmentation results are obtained by performing only one iteration of Grabcut algorithm. By contrast, the previous Grabcut method usually needs many times of iterations to get a better result. Both Figs. 4f and 4h are the results by running the Grabcut algorithm with 20 iterations. This time-consuming process limits the real-time user feedback.

Once an object has been generated, it can be edited collectively at a high level by using its diffusion curves. We support editing operations including linear transformations (translation, rotation, and scaling) as well as nonlinear transformations (localized stretching and bending). As shown in Fig. 5, the extracted stuffed animal in Fig. 4d is warped as a single aggregate object without having to edit individual curves. The curve–object relationship tremendously improves the efficiency of editing diffusion curve images and provides a more natural way to edit at a higher semantic level.

In some ways, a vector graphic is like a collage of objects with each object layered over other objects. Therefore, users can individually move and edit any object independently without ruining the background layers. However, the object

Fig. 5 A demonstration of object-level editing. **a** and **b** are the diffusion curves and its reconstructed result before editing, **c** and **d** are the diffusion curves and its reconstructed result after editing

Fig. 6 A demonstration of image completion and vectorization, **a** is the background image with a hole of black pixels, **b** is the result of image completion, **c** is the diffusion curves, and **d** is the reconstructed result

segmentation leaves a hole of black pixels in the background image as shown in Fig. 6a. Also, some extracted foreground objects are not completed either (Fig. 10a). To this end, we employ a PatchMatch-based image inpainting method [1] and the completion results are shown in Figs. 6b and 10b.

5 Results and discussion

Current vector graphics systems do not offer direct support for diffusion curves as a primitive. To visualize our vectorized diffusion curve images, we adopt the real-time GPU-accelerated parallel rendering algorithm introduced by Jeschke et al. [10]. It relies on a two-stage solution for diffusion curve rendering, which we briefly review here. First, the algorithm rasterizes a set of diffusion curves by dividing each curve into a number of short line segments and rendering a Voronoi diagram of the segments. The Voronoi diagram is used to construct an ID map that records the closest curve point to every pixel and a distance map that records the distance to that curve point. Next, a variable stencil size diffusion solver is used to diffuse the initial guess image, guided by the distance map. The solver applies a scheme similar to Jacobi iterations, but with a large stencil size to transport colors over the image more efficiently. For each iteration, it sets each pixel to the average of its circular neighbors, with a radius computed based on the distance map in order to prevent color mixing from pixels across curve boundaries. To increase the convergence speed and the smoothness of the result, the stencil size is reduced linearly at each step.

Our method is first evaluated on RGB-D images taken from the Middlebury datasets [22]. Figures 1, 7, and 8 show our image vectorization results for the Middlebury images *Bowling*, *Art*, *Motorcycle*, *Adirondack*, and *Jadeplant*. Our method is implemented on a 1.3GHz Intel Core i5 CPU with 4 GB of RAM and an NVIDIA 970M GTX GPU. The rendering of the diffusion curves is implemented on GPU, while the tracing part is implemented on CPU. The statistics and performance of all the images in the paper are shown in Table 2.

Figure 8 compares our results with those produced by Orzan et al. [17]. Given the color image in Fig. 8a and the corresponding depth map in Fig. 8e, Orzan's reconstruction result and our result are shown in Fig. 8b, f, respectively. Because Orzan's method recognizes edges based only on color differences, boundary features separating different objects are not necessarily well preserved, resulting in visible artifacts indicated by green arrows in Fig. 8d. As shown in Fig. 8c, the boundaries of the foreground object are not continuous, allowing colors to mix with the object behind it. In comparison, the inclusion of depth information allows us

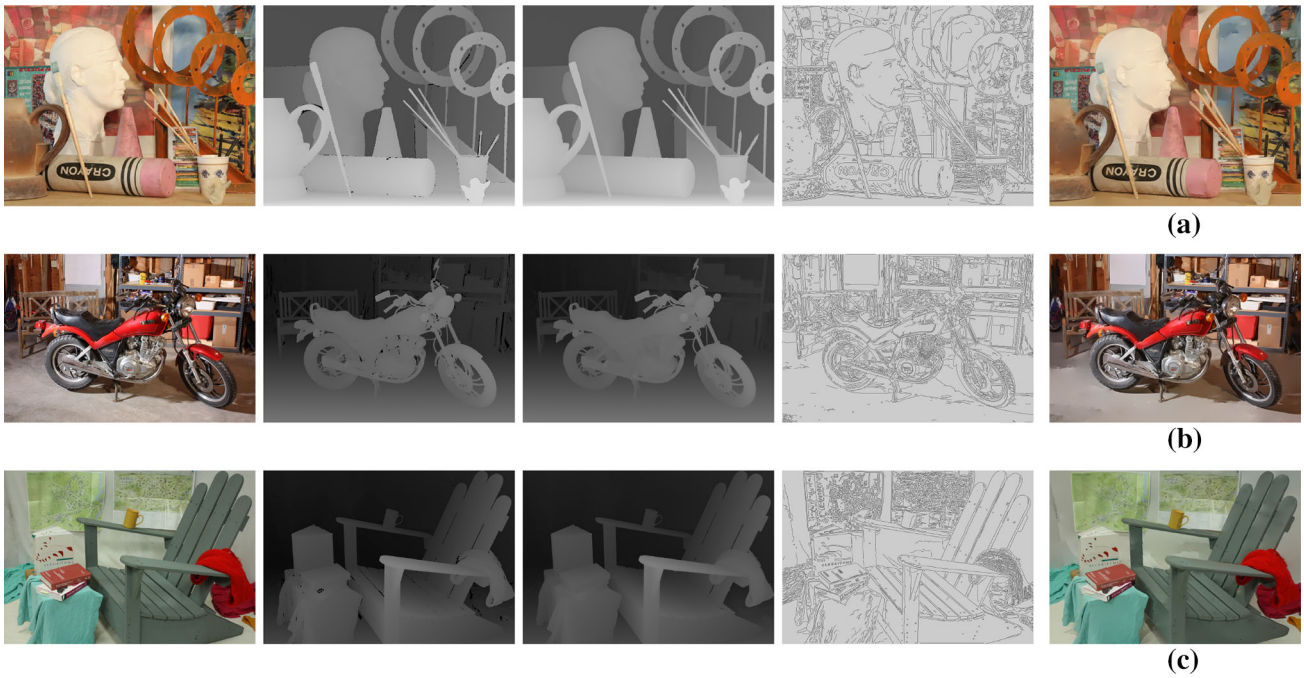

Fig. 7 Our image vectorization results on images from the Middlebury datasets. From left to right, the columns show the input RGB image, input depth map, recovered depth map, diffusion curves, and reconstructed result

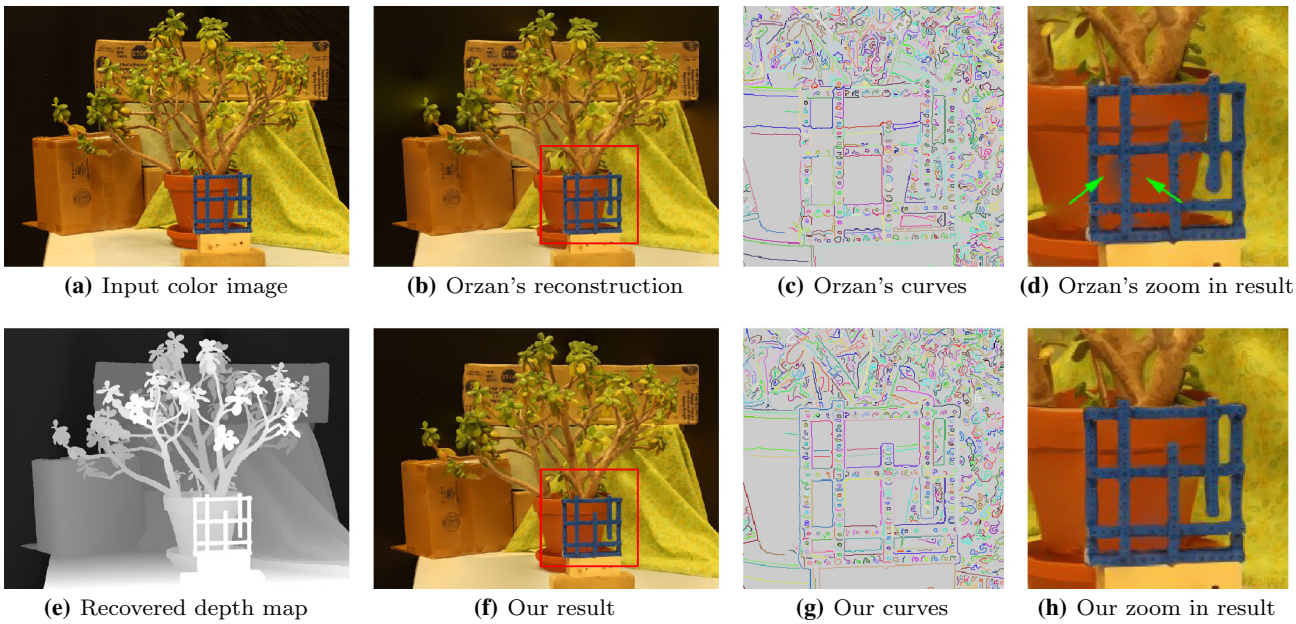

Fig. 8 A comparison of our vectorized images with reconstruction results from Orzan et al. [17]. **a** and **e** are the input color image and the corresponding recovered depth map, **b** is the reconstruction result based on the technique of Orzan et al., and **d** is a close-up of the highlighted region (**b**), with the corresponding curves shown in (**c**), **f** is our reconstruction result, with zoomed curves and close-up in (**g**) and (**h**)

to extract more robust object contours, regardless of color or texture variation in the RGB data. As a result, with the improved curves shown in Fig. 8g, our method obtains better geometric structures and an improved reconstruction result (Fig. 8h).

In addition to reconstruction results of the raster images, our method also makes it easy to perform high object-level editing operations. Most of the diffusion curves methods edit each curve separately rather than an object [9,17,35]. The image vectorization method proposed by [32] which also

Table 2 Statistics and performance of the images shown in this paper. The rendering performance is measured by rendering each diffusion curve image at the same resolution as its input raster images

Dataset	Image resolution	# Depth curves	# Detail curves	Fitting (s)	Rendering (FPS)
Figure 1h	313 × 278	10	272	24.1	1318
Figure 7a	924 × 738	98	1532	108.1	69
Figure 7b	1112 × 746	233	1983	159.4	40
Figure 7c	956 × 660	46	1332	94.5	73
Figure 8f	659 × 497	408	899	62.9	128
Figure 12d	1000 × 750	104	2017	149.9	48

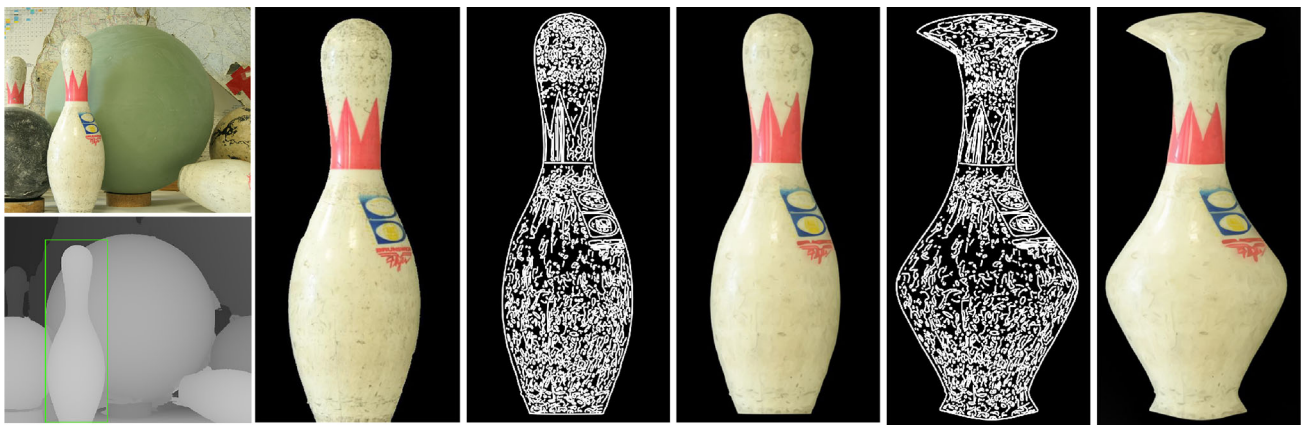

Fig. 9 Depth-aware image segmentation and object-level editing. From left to right, the columns show the RGB and depth images, and an extracted bowling pin with its diffusion curves and reconstructions before and after editing

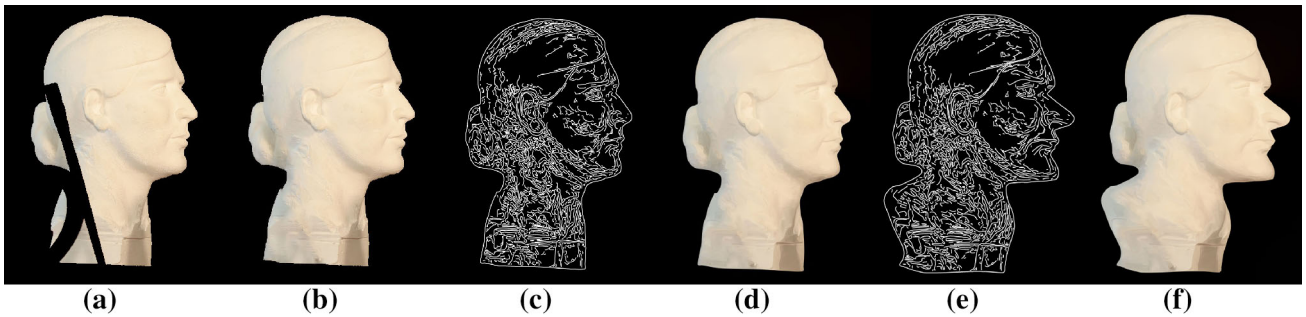

Fig. 10 Object segmentation, image completion, and object-level editing. Given the input RGB image and recovered depth map from the top row of Fig. 7, **a** is the depth-aware image segmentation result, **b** is the image completion result, and **c**–**f** are the diffusion curves and reconstructions before and after object-level editing

Fig. 11 Motor editing example. From left to right, the columns show the diffusion curves and reconstructions before and after object-level editing

based on diffusion curves only performs editing tasks such as detail removal and image stylization without support of shape editing. By comparison, our method is able to extract and manipulate the objects in the vector image, which greatly reduces the time and efforts to create photorealistic editable shapes of real-world complex objects. Figures 9, 10, and 11

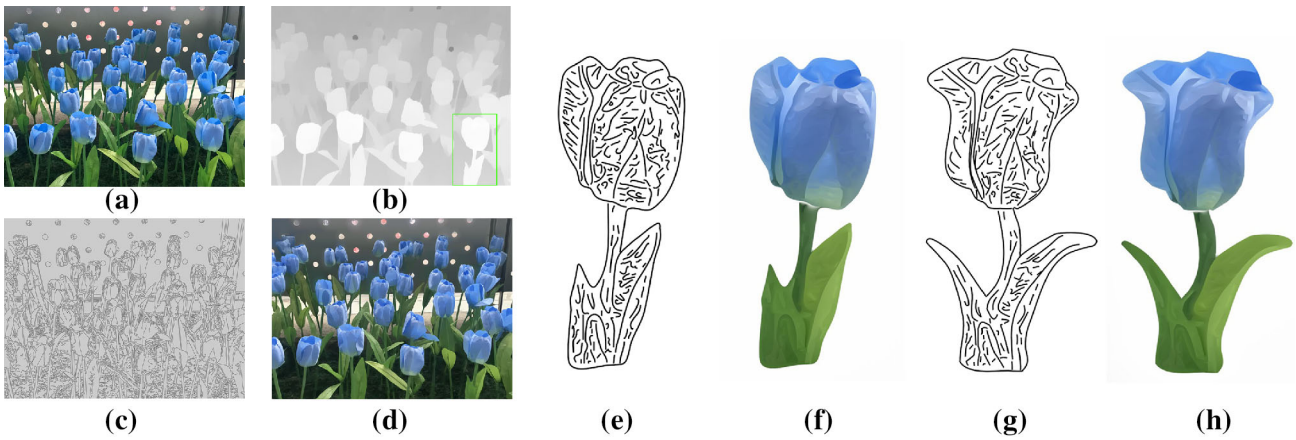

Fig. 12 A vectorized image and an object-level editing result based on an RGB-D image captured using the dual-lens system in an iPhone 7 Plus. The color and depth images are shown in (**a**) and (**b**). We generate the diffusion curves shown in (**c**), yielding the reconstructed vectorization in (**d**). We segment out one flower with its diffusion curves and reconstruction shown in (**e**) and (**f**). The editing results are shown in (**g**) and (**h**)

show another three examples of depth-aware segmentation and object-based diffusion curves image editing.

We also applied our depth-aware vectorization and object-level editing method on RGB-D images captured by the dual lens of the iPhone 7 Plus smartphone; results are shown in Fig. 12.

Limitations Although our approach works well in most cases, there are still some aspects that could be improved. First, our diffusion curve representation includes only color attributes and therefore cannot mimic the full character of some source image textures. Many natural images and photographs contain textural information, which we could simulate by adding noise parameters to our current diffusion curve image representation [11]. Furthermore, the depth edges used for depicting the contours of an object should ideally form closed curves. We would like to develop a strategy that links unconnected depth curve segments into longer closed curves, which would further improve the quality of object-level selection and editing. In addition, to make the object-level editing more user-friendly, in the first stage of depth-aware segmentation method, we would like to estimate an initial depth range based on the statistics within the rectangle instead of tuning the parameters manually.

6 Conclusions

We have introduced a vectorization method that can produce a diffusion curve image from an RGB-D input. By incorporating depth information into the vectorization process, our approach achieves better reconstruction quality in image vectorization. Curves extracted from the input depth map help to preserve the geometric shapes of objects, and support higher-level editing operations. Our experiments and comparison results indicate that our approach produces better reconstruction results than previous diffusion curve image vectorization methods on RGB-D image datasets. We also developed an object-level editing tool, which allows users to edit the content of a diffusion curve image at a higher semantic level than was possible with prior techniques based on diffusion curves.

Acknowledgements This work is supported by Zhejiang Provincial Natural Science Foundation of China (No. LY19F020027), the National Natural Science Foundation of China (No. 61402410), the Key Research and Development Program of Zhejiang Province (No. 2018C03055), and the National Natural Science Foundation of China (No. 61732015).

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Barnes, C., Shechtman, E., Finkelstein, A., Goldman, D.B.: Patchmatch: a randomized correspondence algorithm for structural image editing. ACM Trans. Graph. (TOG) **28**(3), 24 (2009)
2. Boyé, S., Barla, P., Guennebaud, G.: A vectorial solver for free-form vector gradients. ACM Trans. Graph. (TOG) **31**(6), 173 (2012)
3. Cheng, M.M.: Curve structure extraction for cartoon images. In: 5th Joint Conference on Harmonious Human Machine Environment, 13–25 (2009)
4. Finch, M., Snyder, J., Hoppe, H.: Freeform vector graphics with controlled thin-plate splines. ACM Trans. Graph. (TOG) **30**(6), 166 (2011)
5. Ge, L., Liang, H., Yuan, J., Thalmann, D.: Robust 3D hand pose estimation in single depth images: from single-view CNN to multi-view CNNs. In: IEEE Conference on Computer Vision and Pattern Recognition, 3593–3601 (2016)

6. Gupta, S., Arbelaez, P., Malik, J.: Perceptual organization and recognition of indoor scenes from RGB-D images. In: IEEE Conference on Computer Vision and Pattern Recognition, 564–571 (2013)
7. Hou, F., Sun, Q., Fang, Z., Liu, Y.J., Hu, S.M., Hao, A., Qin, H., He, Y.: Poisson vector graphics (pvg). In: IEEE Transactions on Visualization and Computer Graphics (2018)
8. Ilbery, P., Kendall, L., Concolato, C., McCosker, M.: Biharmonic diffusion curve images from boundary elements. ACM Trans. Graph. (TOG) **32**(6), 219 (2013)
9. Jeschke, S.: Generalized diffusion curves: an improved vector representation for smooth-shaded images. Comput. Graph. Forum **35**(2), 71–79 (2016)
10. Jeschke, S., Cline, D., Wonka, P.: A gpu laplacian solver for diffusion curves and poisson image editing. ACM Trans. Graph. (TOG) **28**(5), 116 (2009)
11. Jeschke, S., Cline, D., Wonka, P.: Estimating color and texture parameters for vector graphics. Comput. Graph. Forum **30**(2), 523–532 (2011)
12. Lai, Y.K., Hu, S.M., Martin, R.R.: Automatic and topology-preserving gradient mesh generation for image vectorization. ACM Trans. Graph. (TOG) **28**(3), 85 (2009)
13. Liao, Z., Hoppe, H., Forsyth, D., Yu, Y.: A subdivision-based representation for vector image editing. IEEE Trans. Vis. Comput. Graph. **18**(11), 1858–1867 (2012)
14. Lindeberg, T.: Edge detection and ridge detection with automatic scale selection. Int. J. Comput. Vis. **30**(2), 117–156 (1998)
15. Newcombe, R.A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A.J., Kohi, P., Shotton, J., Hodges, S., Fitzgibbon, A.: Kinectfusion: Real-time dense surface mapping and tracking. In: 10th IEEE international symposium on Mixed and augmented reality, 127–136. IEEE (2011)
16. Orzan, A., Bousseau, A., Barla, P., Thollot, J.: Structure-preserving manipulation of photographs. In: 5th International Symposium on Non-photorealistic Animation and Rendering, 103–110. ACM (2007)
17. Orzan, A., Bousseau, A., Winnemöller, H., Barla, P., Thollot, J., Salesin, D.: Diffusion curves: a vector representation for smooth-shaded images. ACM Trans. Graph. (TOG) **27**(3), 92 (2008)
18. Perona, P., Malik, J.: Scale-space and edge detection using anisotropic diffusion. IEEE Trans. Pattern Anal. Mach. Intell. **12**(7), 629–639 (1990)
19. Price, B., Barrett, W.: Object-based vectorization for interactive image editing. Vis. Comput. **22**(9–11), 661–670 (2006)
20. Rother, C., Kolmogorov, V., Blake, A.: Grabcut: interactive foreground extraction using iterated graph cuts. ACM Trans. Graph. (TOG) **23**(3), 309–314 (2004)
21. Ruderman, D.L., Bialek, W.: Statistics of natural images: scaling in the woods. Phys. Rev. Lett. **73**(6), 814–817 (1994)
22. Scharstein, D., Hirschmüller, H., Kitajima, Y., Krathwohl, G., Nešić, N., Wang, X., Westling, P.: High-resolution stereo datasets with subpixel-accurate ground truth. Ger. Conf. Pattern Recognit. **8753**, 31–42 (2014)
23. Schneider, P.J.: An algorithm for automatically fitting digitized curves. In: Graphics Gems I, 1st edn., pp. 612–626. Academic Press Professional, Cambridge, Massachusetts, USA (1990)
24. Shen, J., Wang, D., Li, X.: Depth-aware image seam carving. IEEE Trans. Cybern. **43**(5), 1453–1461 (2013)
25. Song, S., Xiao, J.: Sliding shapes for 3d object detection in depth images. In: European Conference on Computer Vision, 634–651. Springer (2014)
26. Song, S., Xiao, J.: Deep sliding shapes for amodal 3D object detection in RGB-D images. In: IEEE Conference on Computer Vision and Pattern Recognition, 808–816 (2016)
27. Sun, D., Sudderth, E.B., Pfister, H.: Layered RGBD scene flow estimation. In: IEEE Conference on Computer Vision and Pattern Recognition, 548–556 (2015)
28. Sun, T., Thamjaroenporn, P., Zheng, C.: Fast multipole representation of diffusion curves and points. ACM Trans. Graph. (TOG) **33**(4), 53 (2014)
29. Wang, C., Zhu, J., Guo, Y., Wang, W.: Video vectorization via tetrahedral remeshing. IEEE Trans. Image Process. **26**(4), 1833–1844 (2017)
30. Wang, T.C., Srikanth, M., Ramamoorthi, R.: Depth from semi-calibrated stereo and defocus. In: IEEE Conference on Computer Vision and Pattern Recognition, 3717–3726 (2016)
31. Xia, T., Liao, B., Yu, Y.: Patch-based image vectorization with automatic curvilinear feature alignment. ACM Trans. Graph. (TOG) **28**(5), 115 (2009)
32. Xie, G., Sun, X., Tong, X., Nowrouzezahrai, D.: Hierarchical diffusion curves for accurate automatic image vectorization. ACM Trans. Graph. (TOG) **33**(6), 230 (2014)
33. Yang, J., Ye, X., Li, K., Hou, C., Wang, Y.: Color-guided depth recovery from RGB-D data using an adaptive autoregressive model. IEEE Trans. Image Process. **23**(8), 3443–3458 (2014)
34. Ye, X., Yang, J., Huang, H., Hou, C., Wang, Y.: Computational multi-view imaging with Kinect. IEEE Trans. Broadcast. **60**(3), 540–554 (2014)
35. Zhao, S., Durand, F., Zheng, C.: Inverse diffusion curves using shape optimization. IEEE Trans. Visual. Comput. Graph. **24**(7), 2153–2166 (2018)

Shufang Lu is an associate professor at College of Computer Science and Technology, Zhejiang University of Technology. She received her BSc degree in software engineering from Wuhan University, and MSc and PhD degrees in computer science and technology from Zhejiang University. Her research interests include computer graphics and computer vision.

Wei Jiang is a PhD student in the Visual Computing Group, University of Victoria. He received his BE degree from Zhejiang University of Technology and MS degree from Boston University. His research interests include computer vision and deep learning.

Xuefeng Ding is a postgraduate student of the College of Computer Science & Technology, Zhejiang University of Technology. He received his BSc degree in software engineering from Zhejiang University of Technology. His research interests include computer graphics and computer vision.

Craig S. Kaplan is an associate professor of computer science at the University of Waterloo. He has a BMath in pure mathematics and computer science from Waterloo, and an MS and PhD in computer science from the University of Washington. Kaplan studies the application of computer graphics and mathematics to problems in art, architecture and design, and is an expert on topics such as Islamic geometric patterns and computational applications of tiling theory. He also enjoys experimenting with computer-aided art and design using modern technology like 3D printing.

Xiaogang Jin is a professor of the State Key Lab of CAD&CG, Zhejiang University. He received his BSc degree in computer science in 1989, and MSc and PhD degrees in applied mathematics in 1992 and 1995, all from Zhejiang University. His current research interests include virtual try-on, insect swarm simulation, traffic simulation, implicit surface modeling and applications, creative modeling, sketch-based modeling, and image processing. He received an ACM Recognition of Service Award in 2015. He is a member of the IEEE and ACM

Fei Gao is a professor at the College of Computer Science and Technology at Zhejiang University of Technology. He received his PhD degree in mechanical engineering from Zhejiang University in 2004. His research interests include image processing, computer vision, and computer-aided design

Jiazhou Chen is an associate professor in Zhejiang University of Technology. He received his double PhD degrees in INRIA Bordeaux Sud-Ouest, France, and the State Key Laboratory of CAD and CG at Zhejiang University, China, in 2012. His research interests include computer graphics and visual media computing.