Video Stylization: Painterly Rendering and Optimization With Content Extraction

Liang Lin, Kun Zeng, Yizhou Wang, Ying-Qing Xu, *Senior Member, IEEE,* and Song-Chun Zhu

***Abstract*—We present an interactive video stylization system for transforming an input video into a painterly animation. The system consists of two phases: a content extraction phase to obtain semantic objects, i.e., recognized content, in a video and establish dense feature correspondences, and a painterly rendering phase to select, place, and propagate brush strokes for stylized animations based on the semantic content and object motions derived from the first phase. Compared with the previous work, the proposed method has the following three advantages. First, we propose a two-pass rendering strategy and brush strokes with mixed colors in order to render expressive visual effects. Second, the brush strokes are warped according to global object deformations, so that the strokes appear to be naturally attached to the object surfaces. Third, we propose a deferred rendering and backward completion method to draw brush strokes on emerging regions and simulate a damped system to reduce stroke scintillation effect. Moreover, we discuss the graphics processing unit-based implementation of our system, which is demonstrated to greatly improve the efficiency of producing stylized videos. In experiments, we verify this system by applying it to a number of video clips to produce expressive oil-painting animations and compare it with the state-of-the-art approaches.**

***Index Terms*—Digital art, graphics processing unit (GPU) processing, painterly animation, temporal coherency, video stylization.**

I. Introduction

THIS PAPER proposes an interactive system for producing painterly animation from video clips. Fig. 1 shows an example produced by our system. Although similar oil-painting effects can be generated manually by the paint-on-glass technique, such animation production is not only laborious but also requires considerable artistic skills. For example, it took over two years for artists to manually produce the 22-min Oscar-winning animation *Old Man and the Sea*. In comparison, our interactive system allows an amateur player to produce painterly animations from real-life video clips with far less time and effort.

In the following, we review the related work for painterly rendering and video stylization in the literature and provide an overview of our method accordingly.

A. Painterly Rendering of a Single Image

In order to render expressive and vivid painterly styles, the essential problem is to extract useful image content, which will guide the selection and placement of brush strokes to embody the artist's intention and abstraction [18], [21]. For example, Collomosse and Hall used image salience (contrast) [9] and Santella and DeCarlo used eye-tracking data [34] to determine the placement and order of brush strokes, while Hertmann proposed curved strokes for rendering an impressive oil-painting style by tracing strong edges or boundaries [13]. Particles and regions on 3-D surfaces were extracted to guide stroke placement in [20] and [31], respectively.

The proposed rendering method is inspired by the painting procedure of artists, in which different stroke styles and placement patterns are applied to different object categories in a scene. For example, the brush styles for wood, water, and rock are distinct from each other in oil paintings. Inspired by the semantic-driven rendering systems [16], [36], [39], in our method, we first categorize objects according to their surface materials and correspondingly construct a dictionary of diverse brush examples by artists. These brushes exhibit rich texture, shape, and thickness, in contrast to those used in the literature. Then we select and place these brush strokes according to image semantic contents. To simulate the painting procedure of human artists, we propose a novel two-pass rendering strategy: a base pass using generic brushes followed by a second pass using category-specific brush strokes. The first pass renders the base color for a region. The strokes in the second pass add object details, such as textures, structures, and tactile feelings.

The painterly rendering style we aim to achieve is the popular oil-painting effect. In fact, it is not a sharp boundary to define the oil-painting style in the research of image and video rendering. Compared to daily pictures and other

Manuscript received January 11, 2012; revised May 2, 2012; accepted June 16, 2012. Date of publication July 30, 2012; date of current version April 1, 2013. This work was supported in part by the National Basic Research Program of China, under Grant 2012CB725300; in part by the Hi-Tech Research and Development Program of China, National 863 Program, under Grant 2012AA011504; in part by the National Natural Science Foundation of China, under Grant 60970156; in part by the Guangdong Natural Science Foundation, under Grant S2011010001378; in part by the Guangdong Science and Technology Program, under Grant 2011B040300029; and in part by the SYSU-Sugon high performance computing typical application project. This paper was recommended by Associate Editor J. Cai.

L. Lin and K. Zeng are with Sun Yat-Sen University, Guangzhou 510275, China, and also with the Lotus Hill Research Institute for Computer Vision and Information Science, Hubei 430074, China (e-mail: linliang@ieee.org; zengkun@gmail.com).

Y. Wang is with the National Engineering Laboratory for Video Technology, Key Laboratory of Machine Perception, School of Electrical Engineering and Computer Science, Peking University, Beijing 100871, China (e-mail: yizhou.wang@pku.edu.cn).

Y.-Q. Xu is with Tsinghua University, Beijing 100084, China (e-mail: yqxu@tsinghua.edu.cn).

S.-C. Zhu is with the Department of Statistics, University of California at Los Angeles, CA 951554 USA (e-mail: sczhu@stat.ucla.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2012.2210804

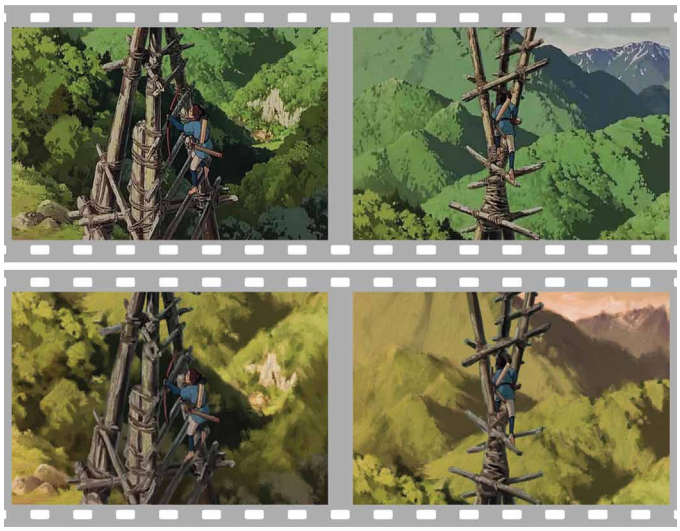


Fig. 1. Example of painterly animation in oil-painting style (the lower frames) from the original movie (the upper frames).

rendering effects, we consider the desirable style to have the following characteristics that are consistent with the previous works [13], [23], [39]: 1) more rich and expressive colors; 2) salient structures and shapes that are emphasized while some homogeneous textures are abstracted; and 3) identity of image that is preserved after rendering. Fig. 2 compares our rendering effects with other state-of-the-art painterly animation algorithms in [11] and [37].

B. Video Stylization

There are two different categories of methods that have achieved remarkable success. The first category extracts image primitives (e.g., regions or edges) from input video clips, and directly stylizes and animates them without using brush strokes. The representative examples include [19], which abstracted the videos as space–time volume data; the roto-curves, contours, and silhouettes that were utilized in [1], [2], and [17], respectively; Wang *et al.* and Bousseau *et al.* transformed the object regions into cartoon style [37] and watercolor style [5], respectively. Winnemoller *et al.* abstracted regions and boundaries by modifying the contrast of luminance and color opponency [38]. Collomosse *et al.* stylized spatiotemporal "video objects" by 3-D segmentation [10]. The other category is stroke-based painterly animation, which artistically expresses object appearance and structure using exquisite physics-based or example-based brush strokes [14], [16], [24], [27], and [31]. The proposed system belongs to this category and it aims at generating expressive animations with painterly brushes.

For stroke-based painterly animation, the essential problems are to stick the brush strokes on object surfaces and to maintain their temporal coherence in the video. This is a nontrivial task for both human artists and the computer-aided systems. Litwinowicz *et al.* first introduced the brush strokes propagation with computed optical flow [27]; Hertzmann *et al.* extended Litwinowcz' approach by making brush attributes adjustable based on the properties of the input video [14]. Hays *et al.* further arranged brush strokes in motion layers and the motion information was also obtained by computing optical flow [11].

Despite the impressive results, the existing methods still leave behind some challenging issues to solve: 1) strokes sometimes drift away from objects during temporal propagation (called the "shower door" effect) [31] and 2) strokes scintillate (or flicker). The two problems become even more serious when using a large number of strokes (e.g., more than 2000) to render a dynamic scene, such as the scene shown in Fig. 1. In our system, we present several techniques to reduce these artifacts. First, we tightly stick the brush strokes to the object by transforming and warping the brush strokes consistently with the object motion and deformation, i.e., the local stroke transformation conforms to the object global transformation. We use two types of robust and distinctive features inside each object to establish dense temporal feature correspondence for both textural and textureless regions between frames of an input video. We adopt a thin-plate-spline (TPS) transformation to describe object deformation. Then the strokes are propagated temporally according to the feature correspondences and warped smoothly by the TPS transformation. Second, we strategically reduce the scintillation effects by the following methods.

1) We confine the brush strokes inside each segmented region to prevent flickering along region boundaries.
2) Since the scintillation is often caused by adding strokes suddenly during brush stroke propagation, we propose a deferred rendering and backward completion strategy for adding new strokes. When a new area emerges, the system defers its rendering and leaves the area unpainted until it grows to certain size. Then, new strokes are added and propagated back to fill the gaps in the previous frames.
3) A damped system is built to stabilize all the strokes in space and time. We simulate the system by attaching springs between brush strokes and minimize the energy of the system by adjusting the rendered strokes so as to enforce coherent motion.

In summary, our system consists of two phases, as shown in Fig. 3: Phase I—video content extraction and Phase II—painterly rendering. The contributions of our method are as follows. First, we propose a novel two-pass rendering strategy and brush strokes with mixed colors, to simulate the painting procedure of human artists for expressive visual effects. Second, for sticking the brush strokes naturally to the object surface during object motion, we present an effective algorithm of matching robust and distinctive features over frames. Last, we reduce the scintillation effects by several techniques: 1) confinement of the strokes inside each object; 2) the deferred rendering and backward completion of new strokes; and 3) a damped system to stabilize strokes in space and time. The user interface is shown in Fig. 4. Note that a preliminary version of this paper was introduced in [23].

In the remainder of this paper, we introduce the video content extraction step in Section II and the painterly rendering procedure in Section III. The GPU-based implementation for our system is discussed in Section IV. We show the

Fig. 2. Comparison of different painterly animation effects. (a) Rendering results of Hays *et al.* [11] (left), Wang *et al.* [37] (middle), and Winnemoller *et al.* [38] (right). (b) Our rendering results with clean boundaries, richer colors, and diverse brush shapes and height fields.

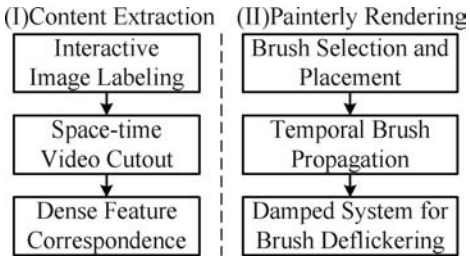


Fig. 3. Key steps in our two-phased painterly animation system.

experimental results in Section V and conclude this paper in Section VI.

II. Interactive Video Content Extraction

A. Interactive Image Labeling

The objective of image labeling is to segment each keyframe into a set of regions with semantic (categorical) labels, called semantic regions. These labels will guide the selection of brush style and stroke placement.

Let **I** be a keyframe from an input video. Our goal is to segment **I** into K disjoint "semantic regions" R_i for $i = 1, \ldots, K$. These semantic regions correspond to different types of recognized objects, such as sky, faces, and trees. Fig. 5 shows a set of segmented typical semantic regions in a keyframe. The semantic regions are the basic operating units in our system, since the strokes and feature correspondences are all confined to the regions. The selection of brush style is also guided by the semantic labels.

To segment a keyframe **I**, a user simply draws a few scribbles in each region R_i using different colors, as shown in Fig. 5. Then we adopt the α-expansion algorithm [6] to segment the image simultaneously into K regions.

The segmentation energy is defined with a pixel-based graphical representation that incorporates the color model D_v and pairwise spatial model $V_{u,v}$ as

$$E_L = \sum_{v} D_v(l_v) + \sum_{\{u,v\}\in\mathcal{N}} V_{u,v}(l_u, l_v) \tag{1}$$

where $\mathcal{N}$ is the set of interacting (adjacent) pairs of pixels and a pixel v can be assigned a multiple possible label, $l_v \in \mathcal{L}$. The color model D_v is defined as a multi-Gaussian function in the Luv-space and the spatial model $V_{u,v}$ is a Euclidean distance over the image domain. Please refer to [6] for details.

TABLE I
Twelve Material Classes of Semantic Regions

Mountain	Water	Rock/building	Leaf/bush/grass
Face/skin	Hair/fur	Flower/fruit	Sky/cloud
Cloth	Trunk/twig	Abstract background	Wood/plastic

We perform a sequence of binary two-way cuts to approximately solve this multilabeling problem and the procedure is described as follows.

1) Manually place scribbles of multiple labels, $\mathcal{L}$, in the image, implying different motion and material properties.
2) Assign an initial label for each pixel based on placed scribbles and obtain an initial labeling energy.
3) Loop for each label $\alpha \in \mathcal{L}$:
   a) find an optimal α-expansion move, which leads cuts to partition α label from all other labels;
   b) decline the move if there is no energy decreases.
4) Segment the image with the final labeling.
5) Allow users to refine the segmentation by adding new scribbles.

After the segmentation, each region R_i is assigned a semantic label ℓ_i corresponding to 12 material categories. A recently proposed method, namely, "texton boost" [35], is employed for image classification. In fact, as the regions are already well segmented, the classification becomes easier and we can achieve more accurate results than [35].

We annotate 474 images for the 12 categories shown in Table I. Then we learn a strong classifier with various discriminative features, including texton filters, the hue, saturation and value color histogram, and the histogram of gradient using the boosting framework [35]. For a segmented frame of size 720×480, it takes about 3–5 s for region classification. We provide a friendly user interface to correct classification errors, if any, which can be found in the attached video.

For each input video in our experiments, we segment and label the first frame and then propagate the segments through the video until a new keyframe is specified by the user.

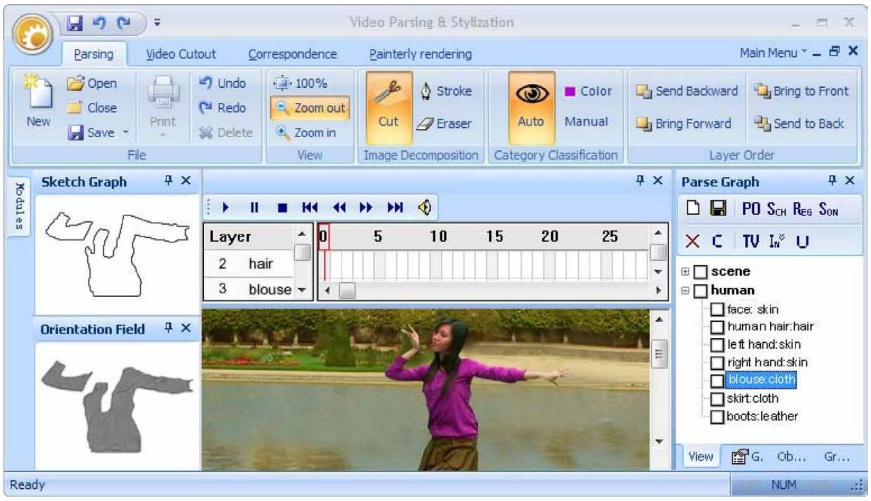


Fig. 4. User interface for the video content extraction and stylization. The right panel shows the semantic labeling for the scene, objects, and parts. The left panel shows the segmentation, sketch, and computed orientation field. These mentioned components will be introduced later.

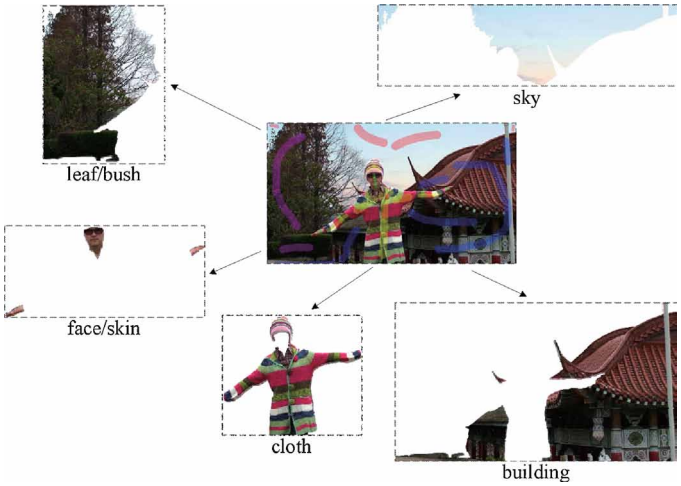


Fig. 5. Segmentation of a keyframe into several regions simultaneously with user scribbles; these regions are consequently classified into 12 categories. Face and hands can be further decomposed according to their different motions.

B. Space–Time Video Cutout

In the video-processing literature, the video cutout algorithms are extensively studied for classifying pixels into foreground and background in space–time volumes [12], [37]. In our system, we adopt an interactive video-segmentation algorithm using localized classifiers [3].

Given a labeled keyframe, a group of local classifiers are constructed around the boundaries of semantic regions, which are then propagated onto successive frames to segment the objects in space–time volumes. Each classifier adaptively integrates multiple local features, such as color, edge, and online learned shape prior. In the proposed system, we iteratively segment each semantic object by treating all surrounding regions as the background. In practice, the salient objects (e.g., people) in videos need to be elaborately segmented, while the segmentation for other scene objects (e.g., trees, buildings) can be obtained quickly by subtracting the objects that have been segmented.

Our interface allows a user to supervise the cutout process and specify the keyframes according to the segmentation results. The algorithm usually achieves good results even for difficult examples with enough user assistance. However, in some cases, which we consider "failures," too much user correction is needed. For example, the scene includes highly dynamic textured objects (e.g., fountains, fire, and heavy rain) or drastic object interactions. Please refer to [3] for more details and analysis.

In our experiments, the cutout is propagated automatically for every τ =10–20 frames depending on the complexity of motion. Then a new keyframe is specified and the user draws new scribbles to continue the cutout process.

The video cutout component will confine the brush strokes inside each segmented region to effectively reduce flickering along region boundaries.

C. Key Feature Extraction and Correspondences

One may view the video cutout as a coarse correspondence between regions in adjacent frames. Our next task is to establish finer correspondence at the feature level within each segmented region. Then the feature correspondence is used to propagate the brush strokes over frames.

A vast variety of image features (keypoints, patches) have been developed in recent years and the consensus is that we should track different features in different types of regions [22], [25], [28]. For each segmented region R_i, we compute two types of features. They are complementary to each other and provide dense matches for correspondence.

1) The SIFT-like feature [29] is suitable for textured areas; it is illustrated by the red dots in Fig. 6(a) and (b) and described by a histogram of image gradients in the neighborhood of a keypoint denoted as h_s. The features are quantized into 72 bins [see Fig. 6(c)].
2) The maximally stable extremal region (MSER) feature [30] is a good descriptor for textureless areas. It is symbolized as ellipse-like shapes [see Fig. 6(a) and (b)] and is described by color histograms in the Luv space

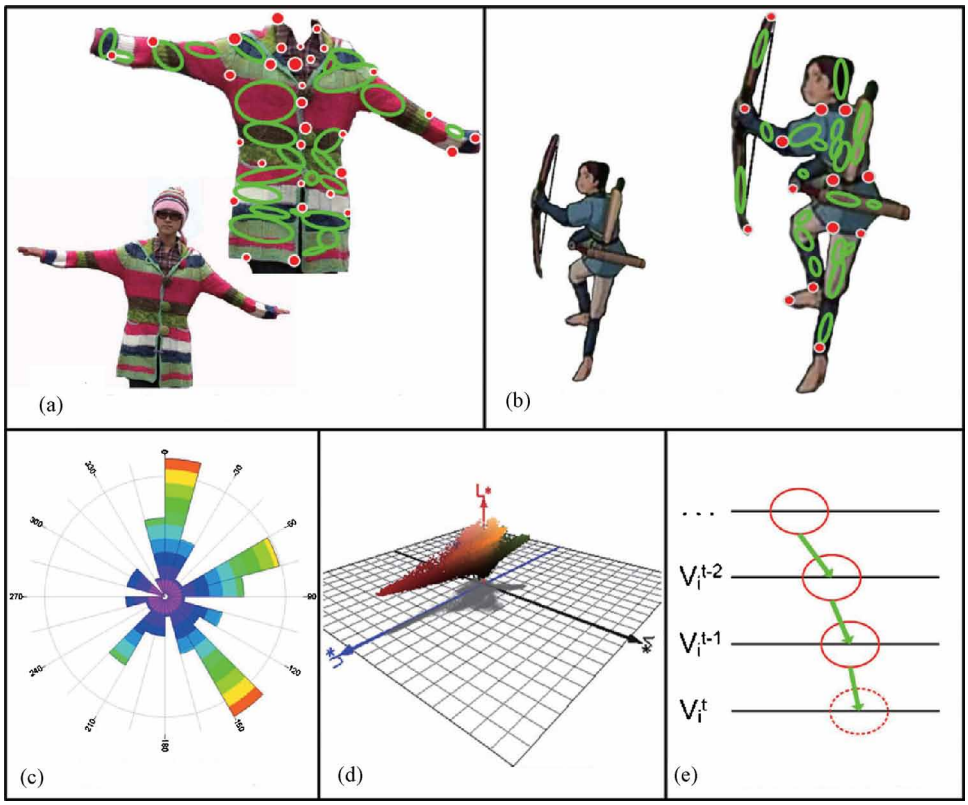


Fig. 6. Discriminative features extracted from (a) textured regions and (b) textureless regions. (c) Gradient histogram of a SIFT feature. (d) Color histogram for a MSER feature. (e) Trajectory of a tracked feature.

[see Fig. 6(d)] as collected from the pixels within the ellipse denoted as h_c. The three axes are quantized in 17, 45, and 40 bins for the L, u, and v dimensions, respectively.

These two types of features are discriminant against viewing angles, scales, and illumination transformations. Their invariant properties make the tracker robust and thus drive the brush strokes to propagate stably in our system.

Suppose a semantic region R_i has M_i feature points at frame t denoted by

$$\mathcal{X}_i = \{X_m = (A_m, h_m), m = 1, \ldots, M_i\}$$

where A_m represents its geometric attributes (location and scale) and h_m is the appearance histogram (h_c for MSER features and h_s for SIFT features). In the next frame, $t + 1$, suppose we detect N_i features in the corresponding region and we denote them by

$$\mathcal{Y}_i = \{Y_n = (A_n, h_n), n = 1, \ldots, N_i\}.$$

We define a similarity measure of two matched features as

$$D(X_m, Y_n) = \alpha d(A_m, A_n) + \mathrm{KL}(h_m||h_n), \quad \text{if } X_m \to Y_n \qquad (2)$$

where $d(A_m, A_n)$ is a quadratic distance between their geometric attributes and $\mathrm{KL}(h_m||h_n)$ is the Kullback–Leibler (KL) divergence to measure appearance variations. We assign a constant penalty β for the remaining unmatched points. Note that the distance between different types of features is set to ∞. α is a tuning parameter to account for matching deformation and β is correlated with occlusions in the video. In our experiments, α and β are set empirically according to our previous work on graph matching [22]: $\alpha = 0.35$ for nonrigid motion and $\alpha = 0.85$ for affine motion, and $\beta = 0.05$.

The tracking problem now becomes finding an optimal bijective mapping $\Phi_{t,i} : \mathcal{X}_i \to \mathcal{Y}_i$ by minimizing the global matching cost

$$\Phi^*_{t,i} = \arg\min_{\Phi_i} \sum_{X_m \in \mathcal{X}_i, Y_n \in \mathcal{Y}_i} D(X_m, Y_n). \qquad (3)$$

This can be interpreted as the optimal assignment problem on a bipartite graph. We adopt the Hungarian marriage algorithm (also referred to as the Kuhn–Munkres algorithm) [15] to solve this optimization problem.

We argue that the proposed feature correspondence with a video cutout method is more effective in propagating the strokes than utilizing the optical flows introduced in [5], [11], and [14] for three reasons: 1) the number of features in each frame is much smaller than the number of pixels, and they are more reliable for computing their correspondences than using optical flows; 2) our features are extracted from both texture and textureless areas, while the optical flow estimation is often difficult in flat regions; and 3) the region boundaries eliminate many cross-region feature mismatches. A comparison of our approach to a method using optical flows is shown in Fig. 7.

III. Painterly Rendering

Based on the video content extraction in Phase I, the next step stylizes a video into a painterly animation with three

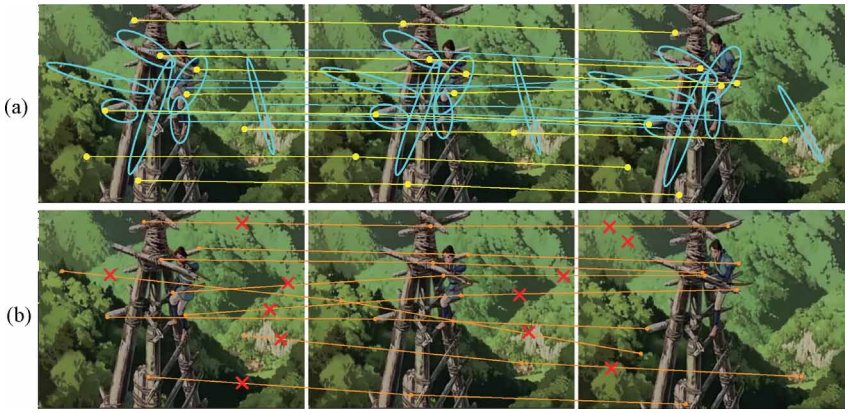


Fig. 7. Comparison of correspondence establishing. In three sequential cartoon frames, we exhibit (a) the correspondence with SIFT features (indicated by spots) and MSER features (indicated by ellipses) and (b) the correspondence with optical flows. The yellow spots and the incorrect correspondence are marked by the red crosses.

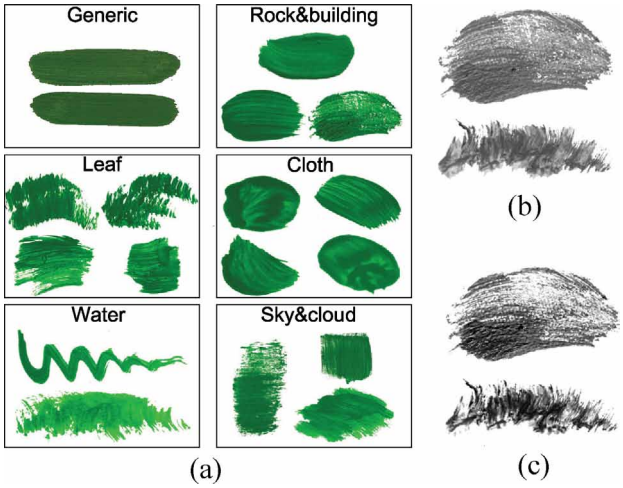


Fig. 8. Shown are (a) examples of brush strokes; the generic brushes (in the top-left cell) and the class-specific brushes; (b) two alpha maps of brushes, where the darker pixels have higher opacity values; and (c) two height maps of brushes, where the darker color indicates higher thickness.

modules, allowing user interactions:

1) keyframe brush stroke placement according to the region semantic label and an orientation field within each region;
2) temporal brush stroke propagation guided by dense feature correspondence;
3) brush stroke stabilization by a damped system.

A. Brush-Based Rendering

There are two important components for rendering keyframes: a semantic-driven brush selection and a brush placement guided by a region orientation field. For enhancing rendering efficiency, the system allows a user to specify keyframes, e.g., every 10–20 frames. The selection of keyframes relies on the object motions. In practice, we can set the keyframes for rendering by roughly estimating the motions of objects beforehand.

Inspired by the previous work [14], [39], we use over 800 exemplar strokes to enrich the painterly rendering styles. These strokes were manually produced by several artists, who were asked to draw strokes for 12 material classes, as shown in Table I. Each class forms a brush dictionary Δ_{ℓ_i}, five of which are illustrated in Fig. 8(a).

Each brush $\mathbf{B} = (\ell, \Lambda, \mathbb{C}, \alpha, \mathbb{H}, \{c_i\})$ is characterized by a label ℓ for its material class, the image lattice Λ, its color map $\mathbb{C}$, alpha map α, height map $\mathbb{H}$, and a number of control points $\{c_i\}$. In Fig. 8(a), the original colors of the strokes are set to green and in the rendering process the brush color is substituted by the color of the pixel where the brush is placed. The height map and alpha map are created by the artists with the drawing software Corel Painter. As shown in Fig. 8(b) and (c), the height map is used to indicate the texture thickness and the alpha map models the opacity of a brush stroke. The control points are the keypoints on the backbone of the brushes and around their boundaries (see Fig. 12).

The proposed brush-based rendering method differs from our previous work [39] in two aspects: 1) we adopt a two-pass rendering strategy to enhance the painterly effect and 2) we enrich brush strokes with mixed colors to improve color contrast of individual strokes and simulate a real brush stroke appearance.

1) *Two-Pass Rendering:* We paint each semantic region R_i with two passes. The first pass applies some generic brush strokes [the top-left cell in Fig. 8(a)], which are often flat and semitransparent. The generic brushes are from a shared dictionary Δ_g to paint all categories of materials. This pass is inspired by human artists who first use large brush strokes to put on base colors for a region. In our algorithm, we abstract unimportant details in textural areas and introduce new colors to enhance contrast in flat areas. On top of the first-pass rendering, the second pass is a semantic-driven rendering process. It places category-specific brush strokes to render different types of object surfaces of diverse textures, opacities, and height fields. According to the semantic label ℓ_i, the system selects strokes from the corresponding brush dictionary Δ_{ℓ_i}. Some results of the two-pass rendering can be found in Fig. 11.

Given a region to be painted, in both the passes, the system places brush strokes according to an orientation field of the region. To compute the orientation field $\Theta_i(x, y)$ at pixel (x, y) in region R_i at a keyframe, we first detect the "sketches"

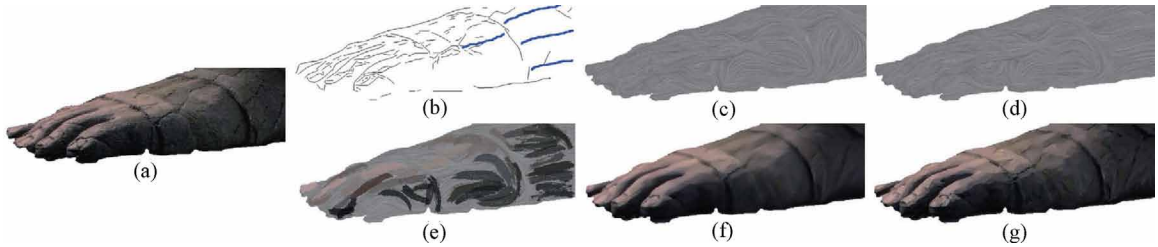


Fig. 9. Brush stroke placement guided by the orientation field. Given (a) a source image, we first (b) compute the strong edges and boundaries, where the user is allowed to place some oriented scribbles (blue curves) to manipulate the field. The computed orientation fields (c) without using user scribbles and (d) with scribbles are comparably shown. With the orientation, the system performs two-pass rendering. (e) Placing generic brush strokes at the first-pass rendering. (f) Result of the first-pass rendering. (g) Final result after the second-pass rendering.

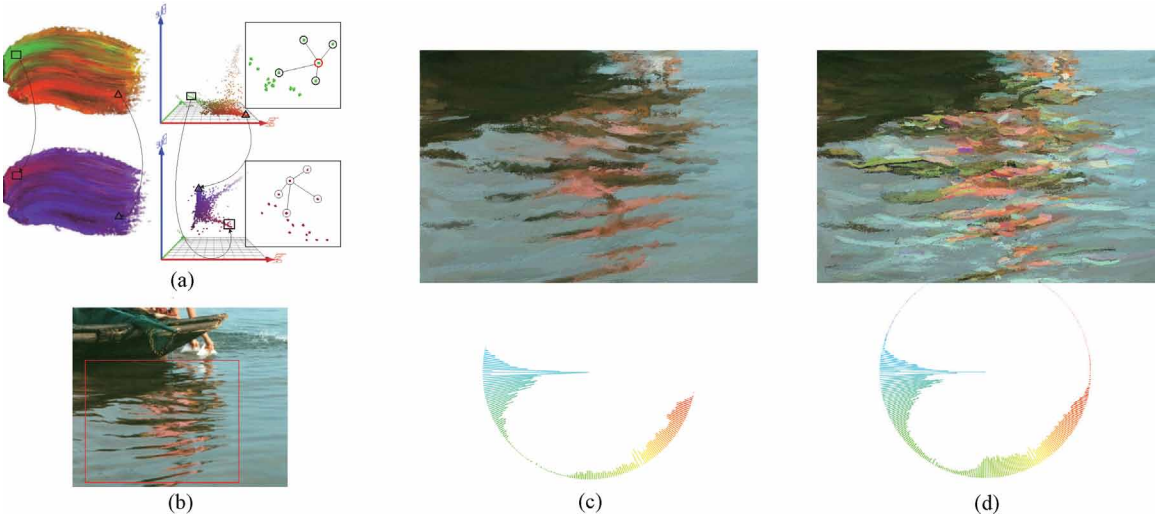


Fig. 10. Painting with colorful brush strokes. (a) Brushes (left) have rich color distributions plotted in the RGB space (right). A brush in the dictionary can change its color distribution (from top to bottom) in order to fit the corresponding color in the input image. (b) A source image with a highlighted region (in the red box) is shown. (c) A result with normal brushes and the corresponding hue distribution (d) and result with colorful brushes and the hue distribution are shown.

inside R_i. These "sketches" are strong edges and bars, as shown in Fig. 9(b). For the pixels on the interior sketches or on the boundary owned by the region, $\Theta_i(x, y)$ is set to be the orientation of the edge or boundary. Note that the boundaries owned by other regions that occlude R_i should not affect the orientation field of R_i. Then we run a diffusion process [8] to fill the orientation of the rest of the pixels and obtain a smooth flow field [see Fig. 9(c) and (d)]. Note that the diffusion process can be interfered with by user interactions, i.e., a user may draw additional scribbles in the region so as to change the orientation flow [as Fig. 9(b) shows]. The manually placed scribbles are treated as the strong edges in the diffusion process. In Fig. 9, we compare the orientation fields without using user scribbles and using scribbles in Fig. 9(c) and (d). It can be seen that the user scribbles are able to regulate the orientation field more smoothly in the existence of cluttered edges. It is the same procedure to calculate the orientation field $\Theta_{\mathbf{B}_j}$ for a brush B_j.

To place a brush stroke onto a small region r, the system finds the most suitable brush stroke by matching the orientation fields $\Theta_{\mathbf{B}_j}$ and Θ_r

$$\mathbf{B}^* = \arg\min \mathcal{M}(\Theta_{\mathbf{B}}, \Theta_r) \tag{4}$$

where the similarity measure $\mathcal{M}(\cdot, \cdot)$ for orientation fields is defined as the KL divergence of the two orientation histogram over all pixels.

To enhance the rendering efficiency and diversity, for each unpainted area the system randomly selects a small subset (5–8) of candidate brushes from the dictionary Δ_{ℓ_i}, and then finds the best match and paints it. The number of selected brushes is set empirically according to the painting effect we designed. In Fig. 9(e) and (f), we show some intermediate results of placing brush strokes; the final result is shown in Fig. 9(g). Our interface allows a user to refine some of the brush strokes after the rendering. The operations include adding, removing, and editing brush strokes (i.e., rotating and translating). More results of two-pass rendering can be found in Fig. 11.

2) *Brush Strokes With Mixed Colors:* We mix warm colors (e.g., yellow, orange) with cold colors (e.g., blue, purple) in order to simulate the real appearance of brush strokes and enhance their color contrast. Fig. 10 illustrates a colorful brush with color-map transformation. Intuitively, when a brush is placed in the image from which it picks the image color, its color map must be transformed coherently so that the local neighborhood in the color map is preserved, and so is the relative brightness between pixels.

In Fig. 10, we present an example of painting with colorful brush strokes. In Fig. 10(c) and (d), we show the hue distributions of the painted images. We compare the rendering result to that using strokes without color enhancement. It can be observed that the rendering effect is more vivid and artistic using strokes with color enhancement.

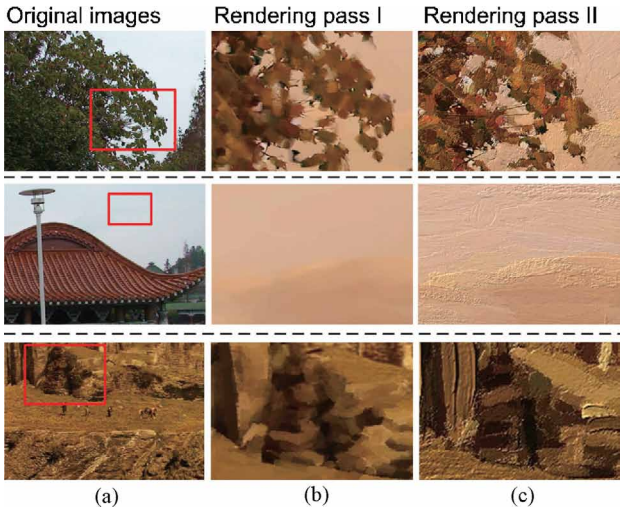


Fig. 11. Examples of two-pass brush-based rendering. (a) Original images. (b) Results of the first-pass rendering. (c) Results of the second-pass rendering.

Unlike the original brushes with a single color, e.g., green, each mixed-color brush from our dictionary **B** includes an additional color map on all pixels, $\mathbb{C} = Q_s$. These colors are clustered by using Gaussian mixture models (GMMs) of k components. We empirically set k =2–5 based on the observation that artists usually mix a few dominant colors into a stroke for vivid effects.[1]

The dominant colors (component) $\{q_1, q_2, \ldots, q_k\}$ of the brush [e.g., green and orange in Fig. 10(a)] are the mean colors of the GMMs. Let z_1 be the image color at the position where the brush **B** being placed; we can randomly select colors $\{z_2, z_3, \ldots, z_k\}$ around z_1 in RGB space. Then the transformed color map Z_t of the brush can be obtained by the transformation from $Q = Q_s \cup \{q_1, q_2, \ldots, q_k\}$ into $Z = Z_t \cup \{z_1, z_2, \ldots, z_k\}$. This transform can be analytically solved by a local linear embedding algorithm [33] in the following steps.

a) Compute the nearest neighbors $\mathcal{N}_i$ for each color $q_i \in Q$ in RGB color space.

b) Compute the reconstruction weights w_{ij} of the neighbors that minimize the error of reconstructing q_i

$$W^* = \arg\min \sum_i |q_i - \sum_{q_j \in \mathcal{N}_i} w_{ij} q_j| \tag{5}$$

subject to the constraints, $\sum_{q_j \in \mathcal{N}_i} w_{ij} = 1$.

c) Compute the embedded colors Z that best preserve the local manifold structure represented by the reconstruction weights

$$Z^* = \arg\min \sum_i |z_i - \sum_{z_j \in \mathcal{N}_i} w_{ij} z_j| \tag{6}$$

subject to the constraint of initial color mapping, $\{q_1, q_2, \ldots, q_k\} \rightarrow \{z_1, z_2, \ldots, z_k\}$. Fig. 10(a) illustrates a colorful brush with a color-map transformation.

We briefly summarize the brush-based rendering method for keyframes in Algorithm 1.

[1]Available at http://en.wikipedia.org/wiki/Oil_painting.

Algorithm 1 Two-pass rendering procedure

Input: An unpainted semantic region R_i; generic brush dictionary Δ_g and specific brush dictionary Δ_{ℓ_i}.

Output: A stylized region covered by brush strokes.

1) Compute orientation field $\Theta_i(x, y)$ for each pixel in R_i
2) Pass 1: Repeat loop: paint R_i with Δ_g.
   a) Randomly select a cluster of unpainted pixels r.
   b) Obtain the orientation field Θ_r.
   c) Find the most suitable generic brush by matching orientation fields.
   d) Render the selected brush on r.
   e) Mark all pixels covered by this brush as painted.
3) Pass 2: Repeat loop: paint R_i with Δ_{ℓ_i}.
   a) Randomly select a small number of brush strokes as candidates from Δ_{ℓ_i}.
   b) Randomly select a cluster of unpainted pixels r.
   c) Obtain the orientation field Θ_r.
   d) Find the most suitable brush from candidates by matching orientation fields.
   e) Render the selected brush on r.
   f) Mark all pixels covered by this brush as painted.
4) Remind user to check and refine the painting result.

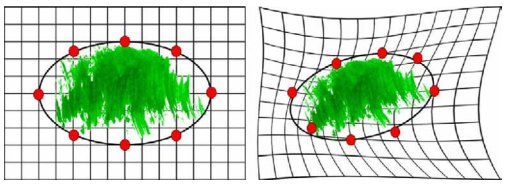


Fig. 12. Propagating a brush stroke between frames. The lattice for the semantic region undergoes a plastic deformation following the tracked feature points with a TPS transform, while the stroke is rigid and follows an affine transform. The red dots are control points of the stroke.

B. Temporal Brush Propagation

After rendering a keyframe, the system propagates the brush strokes to the following frames. It also removes some strokes that cover disappearing regions and introduces new ones for emerging regions. We propose a deferred rendering and backward completion strategy for adding new brush strokes in the propagation process in order to reduce the stroke scintillation artifact. Note that new brush strokes are added only at keyframes and propagated backward to fill unpainted regions in the previous frames.

1) *Brush Stroke Propagation:* For a semantic region R_i at frame t, we have a number of feature points $\mathcal{X}_i = \{X_{ij}\}$ for $j = 1, 2, \ldots, M_i$. We have computed the matching matrix $\Phi_(t, i)$ that maps these feature points to a set of points $\mathcal{Y}_i = \{Y_{ij}\}$ in the next frame. To propagate the brush strokes, the image lattice under the strokes may be distorted according to the corresponding features, as shown in Fig. 12. The warping of the image domain is accounted by the TPS model [4]. That is, pixels at image features $\{X_{ij}\}$ are directly mapped to their correspondence positions $\{Y_{ij}\}$, and the nonfeature pixels are warped to minimize the TPS smoothness constraint energy.

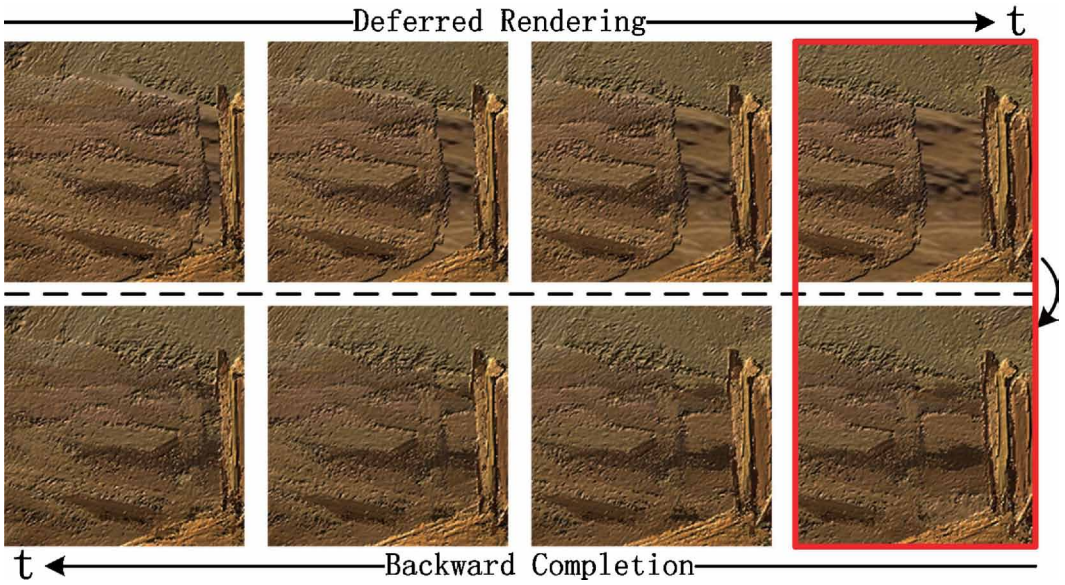


Fig. 13. Deferred rendering and backward completion. When a new area appears (see the top row), the system defers the rendering and lets the area unpainted until to a keyframe (highlighted by the red box), where four new brush strokes are placed and then propagated back to fill the gaps in the previous frames.

Although the underlying lattice is elastic, our brush strokes are treated as *rigid* to preserve the brush textures. If deforming strokes, it may cause an undesirable artifact on its texture and height field. As Fig. 12 shows, each brush has a number of control points, $\{c_i\}$. We fit an affine transformation between the two sets of correspondence features and then transform the associated brush stroke. This idea is inspired by the work in augmented reality [26].

Some brush strokes become smaller during the propagation in the video because of occlusion. We eliminate a stroke if its size is under a threshold. In addition, strokes propagated out of the region boundary are also eliminated.

2) *Deferred Rendering and Backward Completion:* When a new semantic region emerges, or an existing region grows larger, new brush strokes are introduced to cover the new area. To fill small seams between brush strokes, we simply perturb the size and location of the neighboring strokes. If the uncovered area is larger than a certain threshold, the system will not paint it immediately until a new keyframe is specified. For such large unpainted regions, new brush strokes are automatically rendered at the new keyframe by the keyframe-rendering algorithm. Then these strokes are transformed backward frame by frame so as to fill all the corresponding gaps up until the previous keyframe. In addition, the newly added strokes are put underneath the existing ones. This process is illustrated in Fig. 13. This deferred rendering and backward completion process can reduce scintillation effects and other unwanted visual artifacts by avoiding frequent brush stroke changes.

C. Damped System for Deflickering

Once the brush strokes are rendered for all the frames in a video, we attach springs in between the strokes adjacent in space and time to simulate a damped system, as shown in Fig. 14. By minimizing the energy of this system, the strokes are adjusted by an iterative algorithm to remove flickering effects.

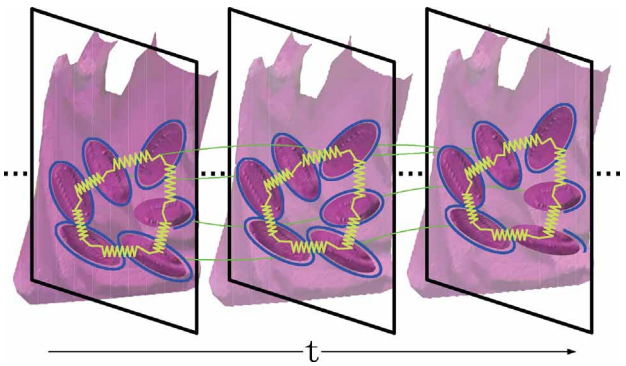


Fig. 14. Damped brush stroke system for deflickering, where the springs are attached between strokes adjacent in space and time.

For the ith stroke at frame t, we denote $A_{i,t}$ as its geometric attributes, including its central point and size. $A_{i,t}$ is a variable and it is initialized to $A^o_{i,t}$, which is its original state obtained from the rendering step. The energy function of the damped system has three terms weighted by two parameters λ_1 and λ_2

$$E_{\text{damp}} = E_{\text{data}} + \lambda_1 E_{\text{smooth}\,1} + \lambda_2 E_{\text{smooth}\,2}. \tag{7}$$

The first term urges that the strokes should stick to their initial positions

$$E_{\text{data}} = \sum_{i,t} (A_{i,t} - A^o_{i,t})^2. \tag{8}$$

Intuitively, this is like attaching a spring between a brush stroke and its initial location so that it does not deviate too far.

The second term enforces a smoothness constraint in time, i.e., the strokes should move smoothly

$$E_{\text{smooth}\,1} = \sum_{i,t} (A_{i,t+1} - 2A_{i,t} + A_{i,t-1})^2. \tag{9}$$

The third term imposes a smoothness constraint between adjacent strokes in space and time. Let $\mathcal{N}_{i,t}$ denote the neighbors of stroke i at frame t. For an adjacent stroke $j \in \mathcal{N}_{i,t}$, their difference $\delta A_{i,j,t} = A_{i,t} - A_{j,t}$ of relative distance and sizes should remain stable in time

$$E_{\text{smooth}\,2}=\sum_{i,t}\sum_{j\in\mathcal{N}_{i,t}}(\delta A_{i,j,t}-\delta A_{i,j,t-1})^2. \tag{10}$$

The energy E_{damp} is in a quadratic form, even though the neighbors of each brush stroke may change over time. It can be solved using the Levenbergy–Marquardt algorithm [32] iteratively. The effectiveness of the damped system is demonstrated in the experiments.

In experiments, we set $\lambda_1 = 2.8$ and $\lambda_2 = 1.1$ according to the object motions and interactions in the video. For some videos including extremely drastic motions and intrackable regions, we can increase the weights of the two smooth terms to further enhance the deflickering.

IV. Parallel Implementation With GPU

The key limitation of many video-stylization systems is the low processing efficiency, basically caused by painting a large number of brush strokes in the video. We propose a parallel implementation using GPU programming to solve this problem and make the system applicable. The benefit of using the parallel implementation is demonstrated in the experiments.

Our implementation uses the NVIDIA CUDA framework in which C functions can be executed multiple times by multiple threads, on multiple processors. We use a NVIDIA GeForce GTX 265 video card, which has 32 multiprocessors, each with four cores. It is thus able to run $32 \times 4 = 128$ threads at once.

In the following, we will discuss the architectures and working flows for stylizing videos with two steps, based on the rendering strategy introduced above: 1) painting brushes in the keyframes and 2) propagating brushes over frames.

A. GPU-Based Rendering on Keyframes

Given a keyframe to be painted, the segmented regions are processed in parallel by a number (i.e., 32) of GPU processors. In practice, the number of processors for one region is decided by the pixel number in the region, namely, the number of brush strokes to be painted.

Based on the two-pass rendering strategy for keyframes (summarized in Algorithm 1), we propose the parallel version of this algorithm that can be split into two parts: host and GPU processing, as shown in Fig. 15. In the host, the image data of the keyframe is created and partitioned into a number of semantic regions according to the content-extraction phase (refer to Section II) and the orientation fields for all regions are calculated as well. Then we transfer the data from the host into the GPU for parallel implementation. It is worth mentioning that the original image and the orientation fields are stored in the shared memory of the GPU, which all the processors can access. The two-pass rendering is then performed. There are two steps for the first-pass rendering.

Step 1) Generate a queue of brushes to be painted. For each pixel in the region, the most suitable brush is decided by matching the orientation fields of the image and the brush (4). Then we collect a queue of brushes to be painted for the region, where the brushes are ranked by the matching score. Since the generation of the queue is processed serially, we execute it in the host part. In practice, it is not necessary to place brushes for all the pixels, while we only constrain each pixel covered by at least one brush.

Step 2) Utilize GPU processors for brush-based rendering in the GPU part, based on the generated queue of brushes in Step 1. We first dispatch the brushes in the queue into a number of GPU processors, as illustrated in Fig. 16, and then each processor is able to control four threads of brush painting. The associations of the GPU processors with the brushes should be recorded for the temporal brush propagating.

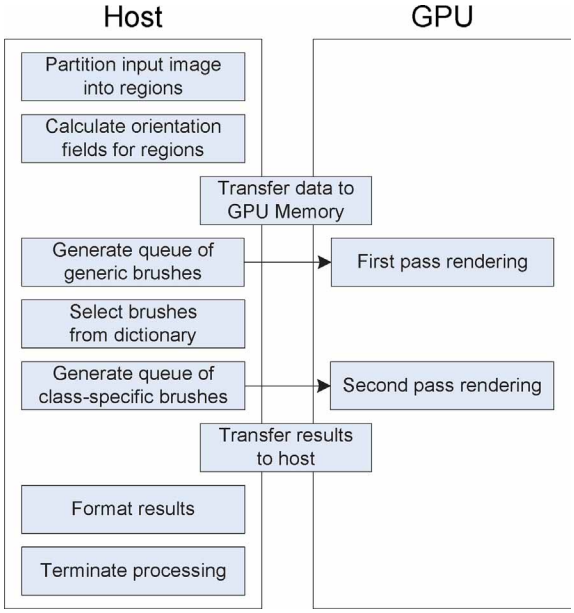


Fig. 15. Host and GPU processing for parallel rendering on a keyframe.

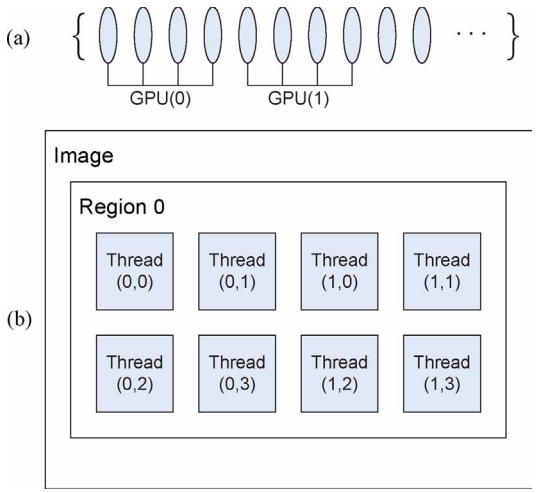


Fig. 16. Illustration of painting brushes by multiple threads, on multiple processors. (a) Queue of brushes (i.e., the ellipses) to be painted is assigned to multiple GPU processors. (b) In region 0 of the image, one GPU processor control a number of threads of painting (i.e., the dark rectangles).

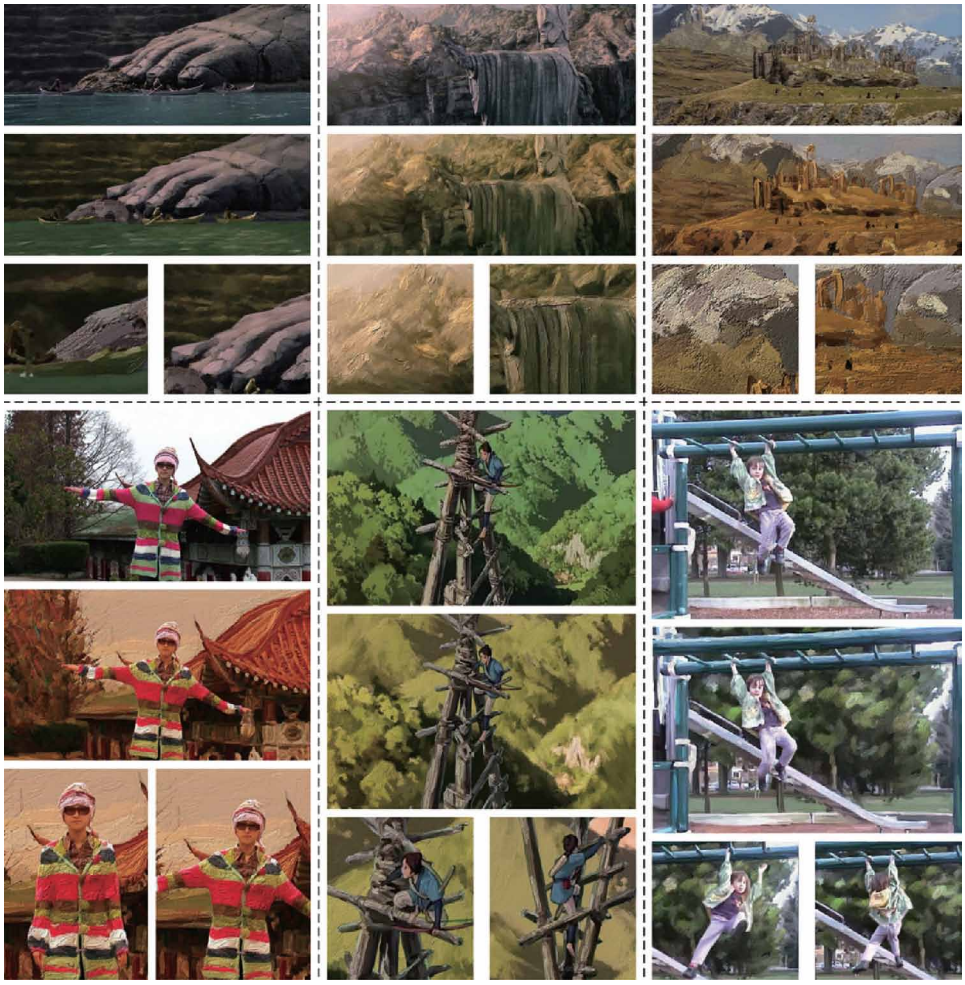


Fig. 17. Sample frames from the animations generated by our system; the original frame and stylized results are shown in each cell.

The rendering for the second pass is very similar with the first pass; the only difference is to select brushes to generate the queue according to region semantics.

B. GPU-Based Rendering With Brush Propagation

Our system propagates the brush strokes from one painted keyframe to the following sequential frames. We consider the sequential frames between two keyframes as a period of rendering. There are two steps, as discussed in Section III-B: 1) painting the brushes according to the object deformations and 2) deferred rendering and backward completion for new emerging areas.

Step 1) For each region to be painted, we first update the attributes of brushes in the queue that are generated in the keyframe rendering. Each brush is transformed based on the TPS model computed by feature correspondences, as illustrated in Fig. 12 and its attribute (image lattice, color map, alpha map, etc.) is updated accordingly. In addition, the brushes are eliminated, namely, marked for no further rendering, which are transformed out of the region or smaller than a threshold. Then we adopt the GPU to paint the brushes according to the associations between the brushes and the GPU processors, which is also kept in the rendering.

Step 2) For new birth areas, we perform the strategy of deferred rendering, as illustrated in Fig. 13, i.e., we stop rendering them until a new keyframe is specified. Thus, we use the algorithm for keyframe rendering (described in the previous section) to paint brush strokes on these areas of the new keyframe and then propagated back to paint the brushes in the previous frames. The processing of backward propagation is exactly the same as Step 1.

V. Experimental Results

We apply our system to several video clips and compare the visual effects with the other state-of-the-art stylization methods. These video sequences include nonrigid human motion, camera motion, and large scene rotation, and shifting. Fig. 17 shows a few frames of painterly animations produced by our system. The results are presented in our supplementary material (please contact the authors to acquire it), in which we also show the contributions of each module of the system by comparing the rendered results with and without the modules: two pass brush rendering, deferred rendering and backward

TABLE II
TIME EXPENSE OF EACH STAGE OF THE ALGORITHM

Seq. Name	Quantity (Frames)	Parsing (min)	Render (CPU) (h)	Render (GPU) (min)	Refining (h)
Cartoon	147	10	10	85	0.5
The Ring	450	20	19	126	1.0
Lady Walk	360	20	15	90	1.0
Girl Lena	239	15	13	98	0.5

completion, video cutout, and damped system for deflickering.

1) A *Lady Walk* sequence shot with a hand-held camera. This video includes nonrigid human motion and camera motion (i.e., shifting and scaling). In the animation, some flat areas (e.g., the sky) are enriched with new colors by the base-pass rendering. The impressive result by the mixed color brushes can be found on the stylized trees and leaves. The different materials (e.g., the clothes, face, and building) are well expressed by the diverse brushes. Due to the robust feature correspondence, we find the brush strokes are basically stuck with the lady as well as the background.
2) Two sequences from the movie *The Lord of the Rings*. One clip includes multilayer motion and large-scale view changes. The strokes are confined in the segmented objects (e.g., the rocks, wall, and water) in the video, and thus the boundary scintillation is removed. The advantage of the damped system can be found for the water animation, where the brush strokes move smoothly and consistently. For the other clip, it includes large scene rotation that leads to drastic emergence and the disappearance of regions. The benefit of the deferred rendering and backward completion is demonstrated. The flickering of the newly adding strokes are effectively removed and we find visual satisfaction on the appearing regions. The results based on these two sequences by Hays *et al.* [11] are proposed for comparison.
3) A *Cartoon* sequence with a climbing boy, in which most of the areas are textureless. This animation exhibits impressive and artistic effect by the depth of field, different object materials, and the enhancement of color diversity. Compared with the method using optical flow for propagating brush strokes [11], we show that the space–time segmentation and the feature-based correspondence stick the brush strokes more stably and tightly.
4) The *Girl Lena* sequence from [37]. Compared with other clips, this video is more challenging due to drastic motion and lower resolution. Since the girl is very small in the video, we treat her as a whole object in the content extraction phase, which causes a few jittering effects of the brush strokes. That could be improved by further segmenting the girl according to the different motions.

In order to produce these animations, a user can specify a keyframe out of every 10–20 frames, and there are about 3500–4800 brush strokes in each keyframe. We carry out the experiments on a personal computer with 3.6 GHz Duo-CPU, 8 GB memory, and a NVIDIA GeForce GTX 265 video card. In the painterly rendering phase, our video sequences are resized into the size of 1280×720 for processing.

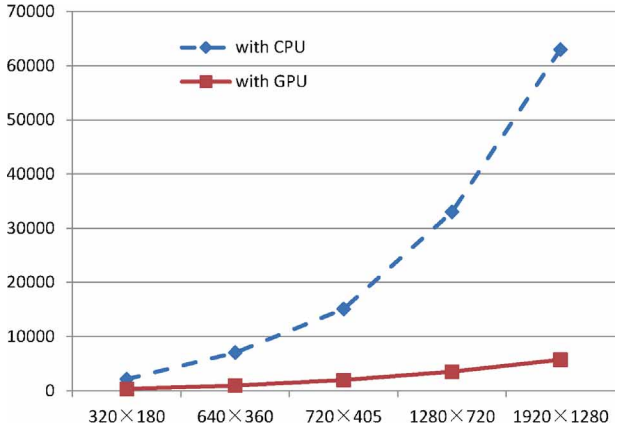


Fig. 18. Time expense for brush-based image rendering. The vertical axis and horizontal axis represent the time consumption (s) and the image sizes, respectively. The dashed curve and the real curve indicate the results with CPU and GPU, respectively.

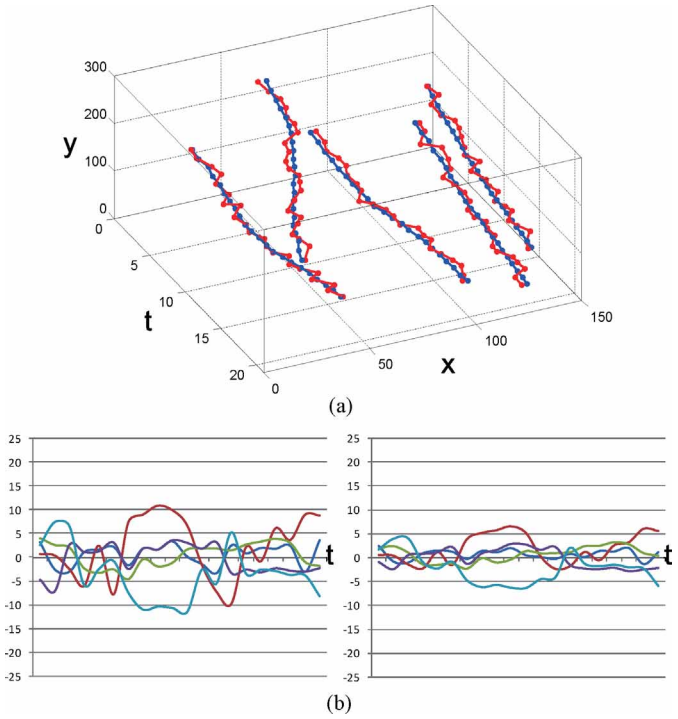


Fig. 19. Evaluation of the deflickering algorithm. (a) Trajectories of five brush strokes in the cloth of the *Lady* sequence before (red) and after (blue) the deflickering process. (b) Displacements of the five brush strokes with the color curves; the results before and after the deflickering process are shown in the left and the right, respectively, where the horizontal axis is time and the vertical axis is distance in pixels.

To demonstrate the efficiency improvement of the GPU-based parallel implementation well, we design an experiment to render with a series of parsed images of different sizes and compare with the traditional CPU performing. As Fig. 18 shows, the efficiency is increased by 10 times on average with the GPU-based implementation. Note that the improvement becomes more significant with images of larger size. Table II summarizes the overall system performance including three key phases, namely, content extracting (parsing), rendering,

and user refining, as well as the comparisons of using CPU or GPU for rendering.

In addition, we present a quantitative evaluation for the damped system discussed in Section III-C. In the *Lady Walk* sequence, we randomly select five brush strokes, and visualize their trajectories and relative displacements before and after the deflickering process, as shown in Fig. 19.

VI. Conclusion

In this paper, we proposed an interactive system for painterly animation. The system consisted of two phases: a content extraction phase to obtain semantic objects in a video and establish dense feature correspondences, and a painterly rendering phase to select, place, and propagate brush strokes for stylized animations based on the semantic content and object motions derived from the first phase. We applied our system to several video clips and achieved very vivid and expressive stylized results.

The limitation of our method is as follows. The TPS transform for the stroke propagation and the smoothness energy between the stokes assumed that the underlying motion was continuous and smooth. This is not always true for stochastic and drastic motions, such as dancing fires and breaking waves. These drastic events need other models after the segmentation process. The current representation also had problems in representing transparent objects, such as steam and wedding veils.

Acknowledgment

The authors would like to thank H. Lv for her assistance in the experiments.

References

[1] A. Agarwala, "SnakeToonz: A semi-automatic approach to creating cel animation from video," in *Proc. Int. Symp. Non-Photorealistic Animat. Render.*, 2002, pp. 139–146.

[2] A. Agarwala, A. Hertzmann, D. H. Salesin, and S. M. Seitz, "Keyframe-based tracking for rotoscoping and animation," *ACM Trans. Graphics*, vol. 23, no. 3, pp. 584–591, 2004.

[3] X. Bai, J. Wang, D. Simons, and G. Sapiro, "Video SnapCut: Robust video object cutout using localized classifiers," *ACM Trans. Graphics*, vol. 28, no. 3, pp. 70:1–70:11, 2009.

[4] L. Bookstein, "Principal warps: Thin-plate splines and the decomposition of deformations," *IEEE Trans. Patt. Anal. Mach. Intell.*, vol. 11, no. 6, pp. 567–585, Jun. 1989.

[5] A. Bousseau, F. Neyret, J. Thollot, and D. Salesin, "Video watercolorization using bidirectional texture advection," *ACM Trans. Graphics*, vol. 26, no. 3, pp. 104:1–104:7, 2007.

[6] Y. Boykov and P. Jolly, "Fast approximate energy minimization via graph cuts," *IEEE Trans. Patt. Anal. Mach. Intell.*, vol. 23, no. 11, pp. 1222–1238, Nov. 2001.

[7] S. Breslav, K. Szerszen, L. Markosian, P. Barla, and J. Thollot, "Dynamic 2D patterns for shading 3D scenes," *ACM Trans. Graphics*, vol. 26, no. 3, pp. 20:1–20:5, 2007.

[8] H. Chen and S. C. Zhu, "A generative sketch model for human hair analysis and synthesis," *IEEE Trans. Patt. Anal. Mach. Intell.*, vol. 28, no. 7, pp. 1025–1040, Jul. 2006.

[9] P. Collomosse and M. Hall, "Painterly rendering using image salience," in *Proc. Eurographics*, 2002, pp. 122–128.

[10] J. Collomosse, D. Rowntree, and P. Hall, "Stroke surfaces: Temporally coherent artistic animations from video," *IEEE Trans. Visualiz. Comput. Graphics*, vol. 11, no. 5, pp. 540–549, May 2005.

[11] J. Hays and I. Essa, "Image and video based painterly animation," in *Proc. Int. Symp. Non-Photorealistic Animat. Render.*, 2004, pp. 113–120.

[12] S. S. Huang, L. C. Fu, and P. Y. Hsiao, "Region-level motion-based foreground segmentation under a Bayesian network," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 19, no. 4, pp. 522–532, Apr. 2009.

[13] A. Hertzmann, "Painterly rendering with curved brush strokes of multiple sizes," in *Proc. ACM Siggraph*, 1998, pp. 453–460.

[14] A. Hertzmann and K. Perlin, "Painterly rendering for video and interaction," in *Proc. Int. Symp. Non-Photorealistic Animat. Render.*, 2000, pp. 7–12.

[15] R. Jonker and A. Volgenant, "A shortest augmenting path algorithm for dense and sparse linear assignment problems," *Computing*, vol. 38, no. 4, pp. 325–340, 1987.

[16] M. Kagaya, W. Brendel, Q. Deng, T. Kesterson, S. Todorovic, P. J. Neill, and E. Zhang, "Video painting with space-time-varying style parameters," *IEEE Trans. Visualiz. Comput. Graphics*, vol. 17, no. 1, pp. 74–87, Jan. 2010.

[17] R. Kalnins, P. Davidson, L. Markosian, and A. Finkelstein, "Coherent stylized silhouettes," *ACM Trans. Graphics*, vol. 22, no. 3, pp. 856–861, 2003.

[18] S. B. Kang, M. Wu, Y. Li, and H. Y. Shum, "Large environment rendering using plenoptic primitives," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 11, pp. 1064–1073, Nov. 2003.

[19] W. Klein, J. Sloan, A. Finkerlstein, and F. Cohen, "Stylized video cubes," in *Proc. ACM Siggraph Symp. Comput. Animat.*, 2002, pp. 15—22.

[20] A. Kolliopoulos, J. M. Wang, and A. Hertzmann, "Segmentation-based 3D artistic rendering," in *Proc. Eurographics Symp. Render.*, 2006, pp. 361–370.

[21] M. Lhuillier and L. Quan, "Image-based rendering by joint view triangulation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 11, pp. 1051–1063, Nov. 2003.

[22] L. Lin, X. Liu, and S. C. Zhu, "Layered graph matching with composite cluster sampling," *IEEE Trans. Patt. Anal. Mach. Intell.*, vol. 32, no. 8, pp. 1426–1442, Aug. 2010.

[23] L. Lin, K. Zeng, H. Lv, Y. Wang, Y. Xu, and S. C. Zhu, "Painterly animation using video semantics and feature correspondence," in *Proc. Int. Symp. Non-Photorealistic Animat. Render.*, 2010, pp. 73–80.

[24] T. Lin, L. Lin, and Q. Wang, "Robust stroke-based video animation via layered motion and correspondence," in *Proc. ACM Conf. MM*, 2012.

[25] L. Lin, P. Luo, X. Chen, and K. Zeng, "Representing and recognizing objects with massive local image patches," *Patt. Recog.*, vol. 45, no. 1, pp. 231–240, 2012.

[26] L. Lin, Y. Liu, Y. Wang, and W. Zheng, "Registration algorithm based on image matching for outdoor AR system with fixed viewing position," *IEE Proc. Vision, Image Signal Process.*, vol. 153, no. 1, pp. 57–62, 2006.

[27] P. Litwinowicz, "Processing image and video for an impressionist effect," in *Proc. ACM Siggraph*, 1997, pp. 407–414.

[28] X. Liu, L. Lin, S. Yan, H. Jin, and W. Jiang, "Adaptive object tracking by learning hybrid template on-line," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 21, no. 11, pp. 1588–1599, Nov. 2011.

[29] D. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[30] J. Matas, O. Chum, M. Urban, and T. Pajdla, "Robust wide baseline stereo from maximally stable extremal regions," in *Proc. Brit. Mach. Vision Conf.*, 2002, pp. 384–393.

[31] B. Meier, "Painterly rendering for animation," in *Proc. ACM Siggraph*, 1996, pp. 477–484.

[32] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York: Springer, 1999.

[33] S. Roweis and L. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[34] A. Santella and D. Decarlo, "Abstracted painterly renderings using eye-tracking data," in *Proc. Int. Symp. Non-Photorealistic Animat. Render.*, 2002, pp. 769–776.

[35] J. Shotton, J. Winn, C. Rother, and A. Criminisi, "TextonBoost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context," *Int. J. Comput. Vision*, vol. 81, no. 1, pp. 2–23, 2009.

[36] J. Suo, L. Lin, S. Shan, X. Chen, and W. Gao, "High resolution face fusion for gender conversion," *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 41, no. 2, pp. 226–237, Feb. 2011.

[37] J. Wang, Y. Xu, H. Y. Shum, and F. Cohen, "Video tooning," *ACM Trans. Graphics*, vol. 23, no. 3, pp. 574–583, 2004.

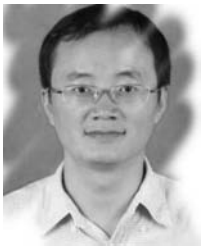
[38] H. Winnemoller, S. C. Olsen, and B. Gooch, "Real-time video abstraction," *ACM Trans. Graphics*, vol. 25, no. 3, pp. 1221–1226, 2006.

[39] K. Zeng, M. Zhao, S. C. Zhu, and C. Xiong, "From image parsing to painterly rendering," *ACM Trans. Graphics*, vol. 29, no. 1, pp. 1–11, 2009.

Liang Lin received the B.S. and Ph.D. degrees from the Beijing Institute of Technology, Beijing, China, in 1999 and 2008, respectively. From 2006 to 2007, he was a Ph.D. student with the Department of Statistics, University of California at Los Angeles (UCLA), Los Angeles.

He was a Post-Doctoral Research Fellow with the Center for Image and Vision Science, UCLA. From 2007 to 2009, he was a Senior Research Scientist with the Lotus Hill Research Institute, Hubei, China. He is currently an Associate Professor with the Software School, Sun Yat-Sen University, Guangzhou, China. He has authored or co-authored over 40 academic papers over a wide range of research topics. His current research interests include, but are not limited to, computer vision, pattern recognition, machine learning, and multimedia technology.

Dr. Lin was a recipient of a number of honors, including several scholarships while pursuing the Ph.D. degree, the Beijing Excellent Students Award in 2007, the China National Excellent Ph.D. Dissertation Award Honorable Mention in 2010, and the Best Paper Runner-Up Award in ACM NPAR 2010.



Kun Zeng received the Ph.D. degree from the National Laboratory of Pattern Recognition Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2008.

He is currently an Assistant Professor with Sun Yat-Sen University, Guangzhou, China. His current research interests include computer vision, multimedia, and nonphotorealistic rendering.

Yizhou Wang received the B.E. degree from the Department of Electrical Engineering, Tsinghua University, Beijing, China, in 1996, and the Ph.D. degree from the University of California at Los Angeles, Los Angeles, in 2005.

From 2005 to 2007, he was a Research Scientist with the Palo Alto Research Center, Palo Alto, CA. Currently, he is a Bairen Professor with the Department of Computer Science, School of Electrical Engineering and Computer Science, Peking University, Beijing. His current research interests include computer vision and visual arts.

Ying-Qing Xu (SM'08) received the Ph.D. degree in computer graphics from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 1997.

He is currently a Professor and the Chair with the Department of Information Art and Design, Tsinghua University, Beijing. From 1999 to 2011, he was a Lead Researcher with Microsoft Research Asia, Beijing. From 2006 to 2011, he was the Co-Director of the Microsoft Digital Cartoon and Animation Laboratory, Beijing Film Academy, Beijing. He has published over 70 papers in computer graphics, computer vision, multimedia, and interaction design. He holds over 20 granted U.S. patents. His current research interests include human–computer interaction, natural user interfaces, computer graphics, and e-heritage.

Dr. Xu is a member of the Association for Computing Machinery.