Painterly Animation Using Video Semantics and Feature Correspondence

Liang Lin[1,2] Kun Zeng[1] Han Lv[1] Yizhou Wang[3] Yingqing Xu[4] Song-Chun Zhu[1,5]

[1]Lotus Hill Research Institute, China
[2]School of Software, Sun Yat-Sen University, Guangzhou, China, { linliang@ieee.org}
[3]Nat'l Engineering Lab for Video Technology, Peking University, Beijing, China, { yizhou.wang@pku.edu.cn }
[4]Microsoft Research Asia, Beijing, China, { yqxu@microsoft.com }
[5]Department of Statistics, University of California, Los Angeles, USA, { sczhu@stat.ucla.edu }

Figure 1: *Two examples of painterly animation with the proposed stylization system. (Please view in high* 400% *resolution in Acrobat reader)*

Abstract

We present an interactive system that stylizes an input video into a painterly animation. The system consists of two phases. The first is an *Video Parsing* phase that extracts and labels semantic objects with different material properties (skin, hair, cloth, and so on) in the video, and then establishes robust correspondence between frames for discriminative image features inside each object. The second *Painterly Rendering* phase performs the stylization based on the video semantics and feature correspondence. Compared to the previous work, the proposed method advances painterly animation in three aspects: Firstly, we render artistic painterly styles using a rich set of example-based brush strokes. These strokes, placed in multiple layers and passes, are automatically selected according to the video semantics. Secondly, we warp brush strokes according to global object deformations, so that the strokes appear to be tightly attached to the object surfaces. Thirdly, we propose a series of novel teniques to reduce the scintillation effects. Results applying our system to several video clips show that it produces expressive oil painting animations.

CR Categories: I.3.6 [Computer Graphics]: Stroke-based Rendering—Interaction Techniques; I.4.8 [Image Processing and Computer Vision]: Semantic Labeling—Tracking

Keywords: non-photorealistic rendering; painterly animation; video parsing.

1 Introduction

We present an interactive system for stylizing an input video to create an artistic and expressive painterly animation. Fig.1 shows some sample frames of two video clips transformed into animated oil paintings. The animation community has long desired such effects, and animators have produced them using conventional frame-by-frame methods or the paint-on-glass technique, both of which are not only extremely laborious, but also require considerable artistic skills. As an example, it took over two years to manually produce the 22-minute Oscar-winning animation *Old Man and the Sea*. In comparison, our interactive system allows amateur players to produce painterly animations from real-life video clips with much less time and effort.

In the following, we briefly review the related work for painterly rendering and video stylization in the graphics community.

(I) Painterly rendering of a single image. In order to render expressive and vivid painterly styles, the essential problem is to extract useful image contents, which will guide the selection and placement of brush strokes to embody the artist's intention and interpretation. In the literature, [Collomosse et al. 2002] used image salience (contrast) and [Santella 2002] used eye-tracking data to determine the placement, ordering and attributes of brush strokes; [Hertzmann 1998] proposed curved strokes for rendering impressive oil painting style by tracing strong edges or boundaries; Particles and regions on 3D surfaces were extracted to guide stroke placement in [Meier 1996] and [Kolliopoulos et al. 2006], respectively.

The proposed rendering method is inspired by the painting procedure of artists, in which different stroke styles and placement patterns are used for different object categories in a scene. For example, the brush styles for wood, water, and rock are distinct from each other in oil paintings. In our method, we first explicitly categorize common objects according to their surface materials and correspondingly construct a dictionary of diverse brush examples by artists. These brushes exhibit much richer texture, shape, thickness than those used in the literature. Then we select and place these brush strokes according to image semantic contents. To simulate painting procedure of human artists, we propose two novel brush-based rendering techniques. (1) We perform a two-pass rendering procedure: a base-pass using generic brushes followed by a second pass using category-specific brush strokes. The base-pass rendering is to put on the base color for a region. The strokes in

second pass add object textures, structures and tactile feeling. (2) In addition to the commonly used brush stroke attributes such as opacity and height field, we augment the brushes with mixed colors for rich expression and contrast.

(II) Video stylization. There are two different categories of methods that have achieved remarkable success. The first category extracts and animates the image representations (i.e., regions or edges) from the input video clips, and directly stylizes them without using any brush strokes. The representative examples include [Klein et al. 2002], which abstracted the videos as space-time volume data; the roto-curves, contours and silhouette were utilized in [Agarwala et al. 2004], [Agarwala 2002], [Kalnins et al 2003], respectively; [Wang et al. 2004] and [Bousseau et al. 2007] transformed the object regions into cartoon style and watercolor style respectively. Winnemoller et al. abstracted regions and boundaries by modifying the contrast of luminance and color opponency [Winnemoller et al. 2006]. Collomosse et al. stylize spatiotemporal "video objects" by 3D segmentation [Collomosse et al. 2005]. The other category is stroke-based painterly animation, which artistically expresses object appearance and structure using exquisite physics-based or example-based brush strokes in [Meier 1996], [Litwinowicz 1997], [Hertzmann et al. 2000]. The proposed system belongs to this category, and it aims at generating expressive and vivid painterly effects.

In the process of painterly animation with strokes, the essential problem is to stick the brush strokes on object surfaces and maintain temporal coherence in the video. This is a nontrivial task for both artists and the computer aided systems. Litwinowicz et al. first introduced the brush strokes propagation with computed optical flow[Litwinowicz 1997]; Hertzmann et al. extended Litwinowcz' approach by making brush attributes adjustable based on the properties of the input video[Hertzmann et al. 2000]; Hays et al. further arrange brush strokes in motion layers and the motion information was also obtained by computing optical flow[Hays et al 2004].

Despite the impressive results, the existing methods still leave behind some challenging issues to solve: (1) sometimes the strokes drifting away from objects. This is also called the "shower door" effect[Meier 1996], and (2) scintillation (or flickering) of strokes. These two problems are more serious when there exist a large number of strokes (e.g., more than 2000) placed in multiple layers, as we show in Fig.1. In our system, we present two techniques to reduce the artifacts. Firstly, we tightly stick the brush strokes to the object and reduce their drifting by warping the brush strokes in consistent with the deformation of the objects, 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 feature correspondence for both textural and textureless regions, and adopt a TPS (thin-plate-spline) transformation to describe object deformation. Then the stroke warping and shifting are smoothly driven by the TPS transformation. Secondly, we strategically reduce the scintillation effects by the following methods: (1) confining the brush strokes inside each segmented region to prevent flickering along the boundaries. (2) Since the scintillation are often caused by adding/removing strokes suddenly in the process of brush stroke propagation, we propose a *deferred rendering and backward completion* strategy for adding new strokes. When a new area emerging, the system defers the rendering and leaves the area unpainted until it grow 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.

2 System overview

Our system consists of two phases: *Video Parsing* to obtain semantic contents and establish dense feature correspondence; *Painterly Rendering* to select, place and propagate brush strokes for stylized animations.

Phase I: Video Parsing To extract semantic content in a video, we segment the initial frame (or a keyframe) interactively by allowing user to draw scribbles on multiple regions. These segmented regions are further categorize into twelve semantic *material classes* according to their material properties, such as rock/mountain, sky, face/skin, etc. (see Tab.1). We call these regions with semantic labels as "semantic regions" (or objects). By using a video cutout algorithm [Bai et al. 2009], we propagate the segmentation and the semantic labels over frames, and thus each object is segmented as a space-time volume. Within each volume, the system then extracts a number of discriminative features. These features are tracked to compute dense correspondences over frames for the propagation of brush strokes in a later stage.

Phase II: Painterly rendering A dictionary of examplar brush strokes is first created by artists for different material (object) categories. At the keyframes, the system performs the two pass stroke rendering. In both two passes, the stroke placement is guided by the orientation field computed from the region. Once a keyframe is rendered, the brush strokes are then propagated over frames. We use a deferred rendering and backward completion strategy to add new strokes, and a damped spring system to stabilize the strokes in space and time by effectively minimizing a quadratic energy function using the Levenbergy-Marquardt algorithm.

In the rest of the paper, we introduce the video content parsing phase in Section 3, and the painterly rendering procedure in Section 4. We show experimental results in Section 5, and conclude this paper in Section 6.

3 Interactive Video Parsing

3.1 Interactive image labeling

The goal of image labeling is to segment each keyframe into a set of semantic regions (i.e., region with semantic labels). This is important for brush style selection and stroke placement.

Let $\mathbf{I}$ be a color image, say a keyframe from an input video. Our goal is to segment the image into K disjoint "semantic regions" R_i for $i = 1, 2, ..., K$. These semantic regions correspond to different objects with semantic labels, such as a sky, face, tree. Fig.2 shows a set of typical segmented and labeled regions for a keyframe. The semantic regions are the basic operating units in our system, since the painting of the strokes and the key features correspondence are all confined to the regions. The selection of brush style is also guided by the semantic label.

To segment the image $\mathbf{I}$, a user simply draws scribbles in each region R_i using different colors, as shown in Fig.2. Then we adopt an α-expansion algorithm [Boykov et al. 2001] to segment the image simultaneously into K regions.

After the segmentation, each region R_i is further assigned a semantic label ℓ_i corresponding to twelve material categories. A recently proposed method, namely "texton boost"[Shotton et al. 2008], is employed for image classification. In fact, as the regions are already well segmented, the classification is easier and more accurate.

Given the annotated training images (474 for 12 categories), we learn a strong classifier with various discriminative features, including texton filters, HSV color histogram and Histogram of Gradient

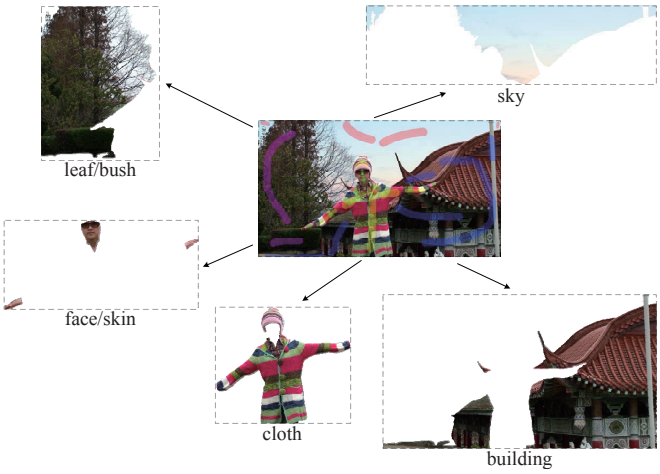

Figure 2: *Segmenting a keyframe into several regions simultaneously by placing user scribbles in them; these regions are classified into twelve categories then. Face and hands can be further decomposed according to the different motion.*

using the boosting framework [Shotton et al. 2008]. For a segmented frame with size 720×480, one shot of classification costs around $3 \sim 5$ seconds. The classification errors can be easily corrected by user interventions.

Table.1 lists the twelve common material classes.

Table 1: *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

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.

3.2 Space-time video cutout

In the graphics literature, the video cutout algorithms in [Wang et al. 2005] treat video as space-time volumes and use mean-shift clustering to classify pixels into foreground and background based on user scribbles in the volume. In our system, we adopt a recently proposed algorithms using localized classifiers [Bai et al. 2009].

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 on-line learned shape prior. We refer more detail to [Bai et al. 2009].

In our experiments, the cutout is propagated automatically for every $\tau = 10 \sim 20$ frames depending on the complexity of motion. Then a new key frame is specified, and the user draw new scribbles to continue the cutout process. Note that the system also allows a user to correct errors in this process.

3.3 Key feature extraction and correspondences

One may view the video cutout as a coarse correspondence at the region (object) level. 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.

A vast variety of image features (key points, patches) have been developed in recent years, and a consensus is that we should track different features in different types of regions.

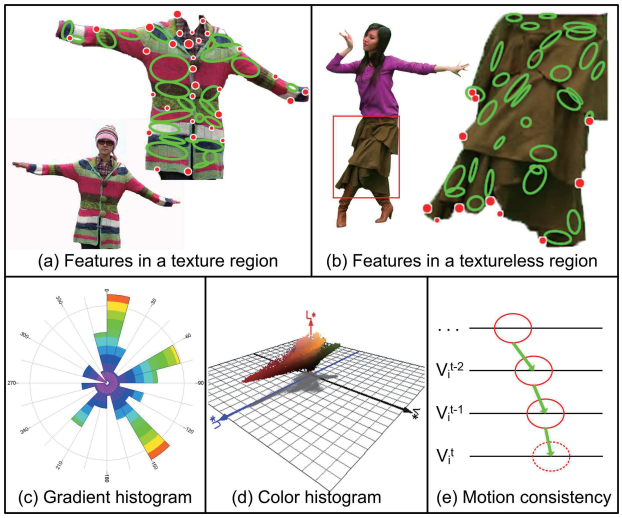

Figure 3: *Discriminative feature points are extracted for texture regions (a) and textureless regions (b). The gradient histogram for a SIFT feature (c), and the color histogram for an MSER feature (d), which are then used for tracking in (e) with a prior probability favoring smooth motion.*

For each segmented region R_i, we compute two types of features and allow them overlap. They are complementary to each other and provide dense matches for correspondence.

- The SIFT-like feature [Lowe 2004] is suitable for areas with rich textures; it is indicated by the red spots in Fig.3 (a) and (b) and described by a histogram h_s of image gradients in the neighborhood of a keypoint, discretized into 72 bins (see Fig.3 (c)).
- The MSER (Maximally Stable Extremal Region) feature [Matas et al. 2002] is a good descriptor for texture-less areas. They are symbolized as ellipses (see Fig.3 (a) and (b)) and described by color-histogram h_c in Luv space (see Fig.3 (d)) collected from the pixels within the ellipse. The three axes are quantized in 17, 45 and 40 bins for the L, u, v dimensions respectively.

These two types of features are distinctive under viewing-angle, scale, and illumination transformations. Their invariant properties make the tracker robust and drive the brushes stroke propagation 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 R_i, and we denote them by

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

For two matched points, we measure their similarity by appearance histogram, and adopt a constant penalty for the remaining unmatched points. We define

$$D(X_m, Y_n) = \begin{cases} \alpha d(A_m, A_n) + \mathcal{KL}(h_m || h_n), & X_m \to Y_n \\ \beta, & \text{otherwise} \end{cases} \quad (1)$$

where $d(A_m, A_n)$ is a quadratic distance between their geometric attributes, and $\mathcal{KL}(h_m||h_n)$ is the Kullback-Leibler divergence to measure appearance variations. Note that the distance between different types of features is set to ∞. In our experiments, the parameter α and β are set empirically, $\alpha = 0.35$ for non-rigid motion and $\alpha = 0.85$ for affine motion, and $\beta = 0.05$.

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

$$\Phi^*_{t,i} = \arg\min_{\Phi_t} \sum_{X_m \in \mathcal{X}_i, Y_n \in \mathcal{Y}_i} D(X_m, Y_n), \quad (2)$$

which can be interpreted as the optimal assignment problem based on a bipartite graph, which can be analytically solved by the Hungarian Marriage algorithm (also referred to as the Kuhn- Munkres algorithm) [Jonker et al. 1987].

We argue that our feature correspondence with video cutout is more effective for driving the stroke propagation than the optical flows used in [Hays et al 2004; Bousseau et al. 2007; Hertzmann et al. 2000], for three reasons: (1) the number of features in each frame is much smaller than the number of pixels, and they have more distinctive feature for computing correspondences; (2) our features are extracted from both texture and textureless areas, while the optical flow estimation is often difficult for flat regions; and (3) the region boundaries eliminate many cross-region feature mismatches.

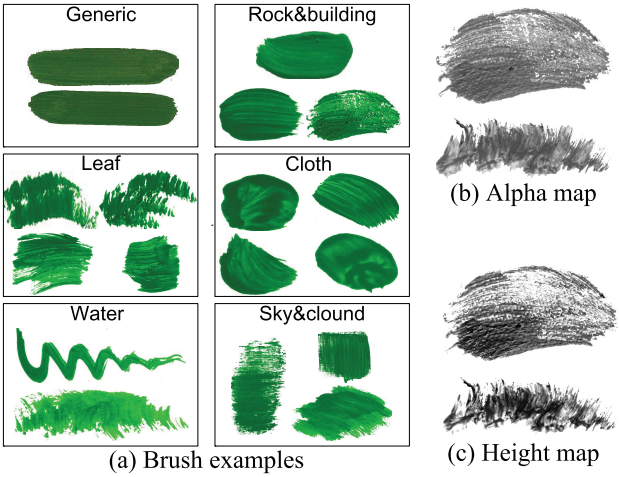

Figure 4: *(a) Brush examples for two-pass rendering; the generic brushes (in the left top cell), often large and semi-transparent, are used in the first pass rendering; the second pass selects brushes according to the material properties. (b) Two alpha maps of brushes where the darker pixels have higher opacity values. (c) Two height maps of brushes, where the darker color indicates higher thickness.*

4 Video Painterly Rendering

Based on the video semantics and correspondence in *Phase I*, the following *phase II* stylizes a video into a painterly animation with three functional modules. The system allows user interaction for style manipulation.

1. Painterly rendering of keyframes with artistic brush strokes, according to the image semantics.
2. Temporal brush stroke propagation driven by the dense feature correspondence.
3. Stabilizing the brush strokes for deflickering by a damped brush system.

4.1 Brush-based Rendering

There are two key components for rendering keyframes: (1) semantic-driven brush design and selection; (2) brush placement guided by the orientation field. For enhancing the animation efficiency, the system allows a user to specify keyframes at one time according to the motion of objects before rendering, (e.g., every 10 frames).

To enrich the painterly rendering styles, inspired by the previous work [Hertzmann et al. 2000; Xu et al. 2006; Zeng et al. 2009], we use over 800 example-based strokes manually produced by several artists who were asked to draw 12 *material classes*, shown in Table1. Each class forms a brush dictionary Δ_{ℓ_i}, five of which are illustrated in Fig.4 (a).

Each brush $\mathbf{B} = (\ell, \Lambda, \mathbb{C}, \alpha, \mathbb{H}, \{c_i\})$ is characterized by its label ℓ for 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.4 (a), the original colors of the strokes are close to green, and in the rendering process, the brush color is taken from the input image where the brush is placed. The alpha map and height map are part of the brush design. As shown in Fig.4 (b) and (c), the height map is used for rendering when illumination is defined in the finishing stage; the alpha map corresponds to the opacity when rendering a brush into image. The height map and alpha map are created by the artists with some painting & drawing softwares. The control points include key points on the backbone of the brush and around the boundary (see Fig.7).

Our brush-based rendering method differs from [Zeng et al. 2009] in two aspects: (1) We adopt a two-pass rendering strategy to enhance the painterly rendering visual effect; (2) We enrich brush strokes with mixed colors to improve color contrast within individual strokes and simulate a real brush stroke appearance.

(I) The two-pass rendering We paint each semantic region R_i in two passes. The first pass uses some generic brush strokes (see the top-left cell in Fig.4 (a)), which are often flat and semi-transparent. The generic brushes are from a shared dictionary Δ_g to paint all categories of material. This pass is inspired by human artists who use large brush strokes first to put on base colors for a region. This process abstracts unimportant details in textural areas and introduces new colors to enhance contrast in flat areas, such as sky and wall. On top of the first-pass rendering, the second pass is a semantic-driven rendering process. It places category-specific brush strokes to reflect object surface material properties in diverse textures, opacity, and height fields. According to the semantic label ℓ_i, the system selects strokes from the corresponding brush stroke dictionary Δ_{ℓ_i}.

In both passes of rendering, the system places brush strokes according to the computed orientation field of a 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" inside R_i. These "sketches" are strong edges and bars. 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 [Chen et al. 2006] to fill orientation of the rest of the pixels and obtain a smooth flow field. Note that the diffusion process can be interfered by user interactions, i.e., a user may draw additional scribbles in the region so as to change the orientation flow. The manually placed scribbles are treated as the strong edges in the diffusion process. The computation of the orientation field $\Theta_{\mathbf{B}_j}$ for a brush B_j is the same.

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.

$$\mathbf{B}^* = \arg\min \mathcal{M}(\Theta_{\mathbf{B}}, \Theta_r), \tag{3}$$

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 brush painting, the system randomly selects a small subset ($5 \sim 8$) of candidate brushes from the dictionary Δ_{ℓ_i}, then finds the best match and paint it. Our interface allows user refine some of the brush strokes after the rendering, including adding, removing, and editing brush strokes (i.e., rotating and translating). More results of two-pass rendering can be found in Fig.6.

(II) 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.5 illustrates a colorful brush with color map transformation. Intuitively, when a brush is placed in the image where it picks the image color, its color map must be transformed coherently so that the local neighborhood in the color-map is pre served, so is the relative brightness between pixels.

In Fig.5 (b), (c), (d), we present an example of painting with colorful brush strokes from a source image as well as a comparison with normal strokes. It can be observed that the rendering effect is more vivid by the colorful ones.

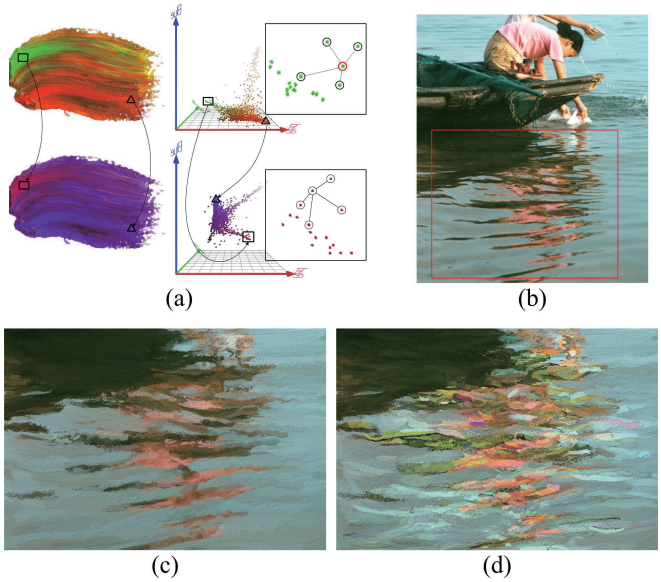

Figure 5: *Painting by the colorful brushes. (a) The brushes (left) have rich color distributions plotted in the RGB space (right). A brush in the dictionary can change its color distribution (from top to bottom) in order to fit the color in the input image. (b) The source image with a highlighted region (in the red box). (c) The painting result of the region with normal brushes. (d) The painting result with colorful brushes.*

Suppose we have a colorful (mixed color) brush $\mathbf{B}$ from the brush dictionary, in which the color map on all pixels is $\mathbb{C} = Q_s$. These colors are clustered using Gaussian mixture models (GMMs) of k components (typically $k = 2 \sim 5$). The dominant (component) colors $\{q_1, q_2, \ldots, q_k\}$ of the brush (e.g., green and orange in Fig. 5 (a)) are the mean colors of the GMMs. Let z_1 be the image color at the position where the brush $\mathbf{B}$ being placed, we can further 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 place 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 an LLE algorithm [Roweis and Saul 2000] in the following steps.

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

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

$$W^* = \arg\min \sum_i |q_i - \sum_{q_j \in \mathcal{N}_i} w_{ij} q_j|, \tag{4}$$

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

(3) Compute the embedded colors Z that best preserves the local manifold structure represented by the reconstruction weights.

$$Z^* = \arg\min \sum_i |z_i - \sum_{z_j \in \mathcal{N}_i} w_{ij} z_j|, \tag{5}$$

subject to the constraint of fixing initial color correspondence, $\{q_1, q_2, \ldots, q_k\} \rightarrow \{z_1, z_2, \ldots, z_k\}$. Fig.5 (a) illustrates a colorful brush with a color map transformation.

4.2 Temporal brush propagation

After rendering a keyframe, the system propagates the brush strokes to the following frames. It also removes some strokes and introducing new ones. There is a non-trivial deferred rendering and backward completion strategy for adding new brush strokes in the propagation process in order to avoid visual artifacts. Note that the new brush strokes are added only at the keyframes and propagated backward to fill unpainted regions in the previous frames.

Brush stroke propagation For a semantic region R_i at frame t, we have a number of key points $\mathcal{X}_i = \{X_{ij}\}$ for $j = 1, 2, ..., M_i$ and $i = 1, 2..., K$. We have computed a matching matrix $\Phi_(t, i)$ which maps these feature points to a set of points $\mathcal{Y}_i = \{Y_{ij}\}$ in the next frame. This mapping will distort the image lattice with an elastic deformation, as Fig. 7 shows. The warping of the image domain is accounted by the Thin-Plate Spline (TPS) model [Bookstein 1989]. That is, pixels at image features $\{X_{ij}\}$ are directly mapped to the counterpart feature positions $\{Y_{ij}\}$, and the non-feature pixels are warped to minimize the TPS smoothness constraint energy.

Although the underlying lattice is elastic, for example a region of cloth, our brush strokes are treated as *rigid* to preserve the brush textures. If deforming strokes, it may cause undesirable rendering effect on its texture and height field. As Fig.7 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.

Some brush strokes become smaller during the propagation in the video or because of occlusion. We eliminate a stroke if its size is

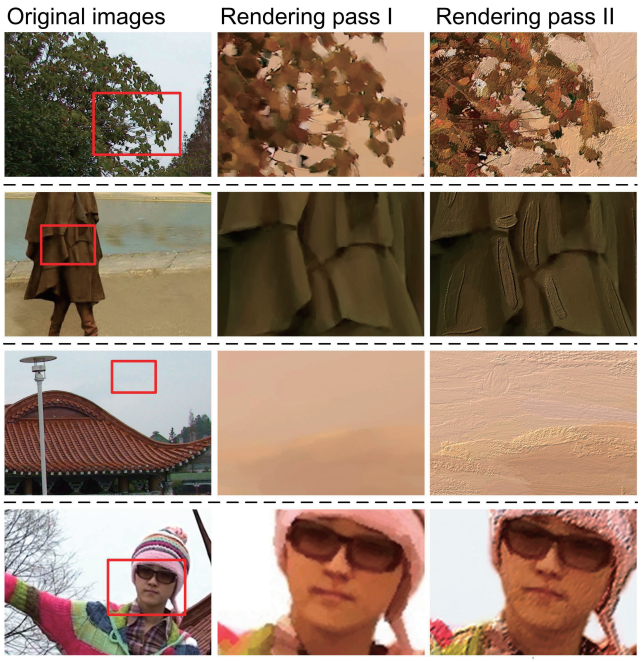

Figure 6: *Samples of two-pass brush-based rendering: the original images (in the left column), the results of first pass rendering (in the middle column), and the results of the second pass rendering (in the right column).*

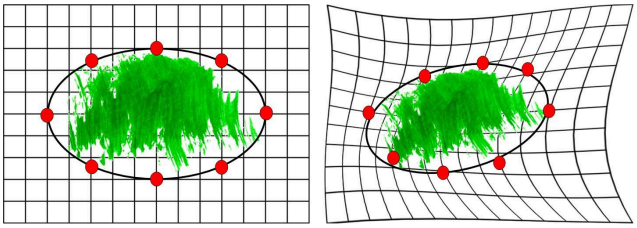

Figure 7: *Propagating a brush stroke between frames. The lattice for the semantic region undergoes a plastic deformation following the tracked feature points with a TPS transform, while the stroke is rigid and follows an affine transform with its control points.*

under a threshold. In addition, strokes propagated out of the region boundary are also eliminated.

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 the location of the neighboring strokes. If the uncovered area is larger than a certain threshold, the system will not paint new emerging regions immediately until a new keyframe is specified. New brush strokes are automatically rendered (by the keyframe rendering algorithm) on the unpainted regions at the new keyframe, and then transformed backward frame by frame so as to fill the all the corresponding gaps in between the two keyframes. In addition, the newly added strokes are added underneath the existing ones. This deferred rendering and backward completion process can reduce scintillation effects and other unwanted visual artifacts by avoiding frequent brush stroke changes.

4.3 A damped system for de-flickering

Once the brush strokes are rendered for all the frames in a video, we attach springs in between brush strokes adjacent in space and time to simulate a damped system, as shown in Fig.8. By minimizing the energy of this system, the strokes are adjusted by an iterative algorithm to remove flickering effect.

For the i-th 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 brush stroke system has three terms weighted by two parameters λ_1 and λ_2,

$$E_{\text{damp}} = E_{\text{data}} + \lambda_1 E_{\text{smooth 1}} + \lambda_2 E_{\text{smooth 2}}, \tag{6}$$

In experiments, we set $\lambda_1 = 2.8$ and $\lambda_2 = 1.1$.

The first term urges that the strokes should stick to its initial position,

$$E_{\text{data}} = \sum_{i,t} (A_{i,t} - A^o_{i,t})^2. \tag{7}$$

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

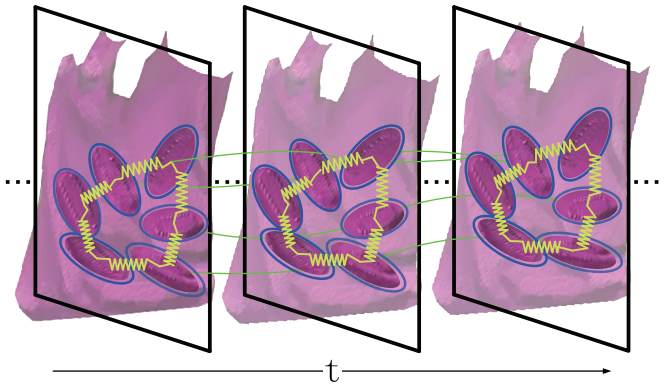

Figure 8: *The damped brush stroke system for de-flickering, where the springs are attached between strokes adjacent in space and time.*

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

$$E_{\text{smooth 1}} = \sum_{i,t} (A_{i,t+1} - 2A_{i,t} + A_{i,t-1})^2. \tag{8}$$

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

$$E_{\text{smooth 2}} = \sum_{i,t} \sum_{j \in \mathcal{N}_{i,t}} (\delta A_{i,j,t} - \delta A_{i,j,t-1})^2. \tag{9}$$

The energy E_{damp} is in a quadratic form, even though the neighborhood of each brush stroke may change over time. It can be solved using the Levenbergy-Marquardt algorithm [Nocedal and Wright 1999] iteratively.

Figure 9: *A few sample frames from the animations from our system; The original frame and stylized results are shown in each cell. (Please view in high* 400% *resolution in Acrobat reader).*

Table 2: *Rough timing for each stage of the algorithm running on 3.6GHz CPU with 4GB memory.*

Seq. name	Quantity	Parsing	Rendering
Cartoon	147 fr.	10 min.	10 hours
Lady Walk	360 fr.	20 min.	15 hours
Girl Lena	239 fr.	15 min.	13 hours

5 Experimental Results

We apply our system on several video clips and compare the visual effects with the other state-of-the-art stylization methods, which are presented in our supplementary material that can be found at the first author's homepage. These videos include non-rigid human motion, camera motion, and large scale scene rotation and shifting. Fig.9 shows a few frames of painterly animations produced by our system. In order to produce these animations, a user can specify a keyframe out of every 10∼20 frames, and there are about 1800∼2500 brush strokes in each keyframe. The time expense of rendering a keyframe is about 15∼20 minutes on a PC with 3.6GHz CPU and 4GB memory. Besides, adding new strokes to another keyframe usually takes less than 5 minutes. Propagating strokes is relatively fast, which takes about 1∼2 minutes per frame. Tab.2 summarizes the system performance.

6 Conclusion: contributions and limitations

In this paper, we propose an interactive system for painterly animation. Our system has the following contributions: (1) It renders artistic style animation using a diverse set of example-based brush strokes, and these strokes are automatically selected according to the object classes. (2) It sticks the strokes tightly to the object surface in the animations by warping and shifting strokes in accordance with the transformation of the object with dense feature correspondence in both textured and textureless areas. (3) It reduces the scintillation effects by several techniques: (i) confining the strokes inside each object; (ii) the deferred rendering and backward completion for newly birth strokes, and (iii) a damped system to stabilize strokes in space and time.

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

Acknowledgement

This work is done at the Lotus Hill Research Institute and is supported by China 863 Program (Grant No. 2007AA01Z340 and No. 2009AA01Z331) and National Natural Science Foundation of China (Grant No. 60970156 and No. 60728203). The authors would like to thank Michael F. Cohen for providing the Girl Lena sequence, in Figure 1 (right), and Figure 9 (right).

References

AGARWALA, A. 2002. SnakeToonz: A Semi-Automatic Approach to Creating Cel Animation from Video. *Int'l Symposium on Non-Photorealistic Animation and Rendering*. 139–146.

AGARWALA, A., HERTZMANN, A., SALESIN, D. H., And SEITZ, S. M. 2004. Keyframe-based Tracking for Rotoscoping and Animation. *ACM Trans. Graphics (Proc. SIGGRAPH)*. 23, 3, 584–591.

BAI, X., WANG, J., SIMONS, D., And SAPIRO, G. 2009. Video SnapCut: Robust Video Object Cutout Using Localized Classifiers. *ACM Trans. Graphics (Proc. SIGGRAPH)*.

BOOKSTEIN, L. 1989. Principal Warps: Thin-plate Splines and the Decomposition of Deformations. *IEEE Trans. on Pattern Anal. and Mach. Intell*. 11, 6, 567–585.

BOUSSEAU, A., NEYRET, F., THOLLOT, J., And SALESIN D. 2007. Video Watercolorization using Bidirectional Texture Advection. *ACM Trans. Graphics (Proc. SIGGRAPH)*. 26, 3, 104:1–104:7.

BOYKOV, Y., And JOLLY, P. 2001. Fast Approximate Energy Minimization via Graph Cuts. *IEEE Trans. on Pattern Anal. and Mach. Intell*. 23, 11, 1222–1238.

BRESLAV, S., SZERSZEN, K., MARKOSIAN, L., BARLA, P., And THOLLOT, J. 2007. Dynamic 2D Patterns for Shading 3D Scenes. *ACM Trans. Graphics (Proc. SIGGRAPH)*.

CHEN, H., And ZHU, S. C. 2006. A Generative Sketch Model for Human Hair Analysis and Synthesis. *IEEE Trans. on Pattern Anal. and Mach. Intell*. 28, 7, 1025–1040.

COLLOMOSSE, P. And HALL, M. 2002. Painterly rendering using image salience. *Eurographics*. 122–128.

COLLOMOSSE, P., ROWNTREE, D., And HALL, M. 2005. Stroke Surfaces: Temporally Coherent Artistic Animations from Video. *IEEE Trans. on VCG*. 11, 5, 540–549.

HAYS, J., And ESSA, I. 2004. Image and Video Based Painterly Animation. *Int'l Symposium on Non-Photorealistic Animation and Rendering*, 113–120.

HERTZMANN, A. 1998. Painterly Rendering with Curved Brush Strokes of Multiple Sizes. *Proc. SIGGRAPH*. 453-460.

HERTZMANN, A., And PERLIN, K. 2000. Painterly Rendering for Video and Interaction. *Int'l Symposium on Non-Photorealistic Animation and Rendering*, 7–12.

JONKER, R., And VOLGENANT, A. 1987. A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems. *Computing*. 38, 325–340.

KALNINS, R. DAVIDSON, P. MARKOSIAN, L. and FINKELSTEIN, A. 2003. Coherent Stylized Silhouettes. *ACM Trans. Graphics (Proc. SIGGRAPH)*. 22, 3, 856–861.

KLEIN, W., SLOAN, J., FINKELSTEIN, A., And COHEN, F. 2002. Stylized Video Cubes. *ACM SIGGRAPH Symposium on Computer Animation*.

KOLLIOPOULOS, A., WANG, J. M., And HERTZMANN, A. 2006. Segmentation-Based 3D Artistic Rendering. *Proc. Eurographics Symposium on Rendering*. 361–370.

LITWINOWICZ, P. 1997. Processing Image and Video for An Impressionist Effect. *Proc. of SIGGRAPH*. 407–414.

LOWE, D. 2004. Distinctive Image Features from Scale-Invariant Keypoints. *Int'l Journal of Computer Vision*. 60, 2, 91–110.

MATAS, J., CHUM, O., URBAN, M. And PAJDLA, T. 2002. Robust Wide Baseline Stereo from Maximally Stable Extremal Regions. *Proc. British Machine Vision Conference*. 384–393.

MEIER, B. 1996. Painterly Rendering For Animation. *ACM Trans. Graphics (Proc. SIGGRAPH)*. 477–484.

NOCEDAL, J., And WRIGHT, S. J. 1999. *Numerical Optimization*. Springer.

ROWEIS, S. And SAUL, L. 2000. Nonlinear dimensionality reduction by locally linear embedding. *Science*. 290, 5500, 2323–2326.

SANTELLA A. And DECARLO D. 2002. Abstracted painterly renderings using eye-tracking data. *Int'l Symposium on Non-Photorealistic Animation and Rendering*. 769–776.

SHOTTON, J., WINN, J., ROTHER, C., And CRIMINISI, A. 2009. TextonBoost for Image Understanding: Multi-Class Object Recognition and Segmentation by Jointly Modeling Texture, Layout, and Context, *International Journal of Computer Vision*. 81, 1, 2–23.

WANG, J., XU Y., SHUM, H.Y., And COHEN, F. 2004. Video Tooning. *ACM Trans. Graphics (Proc. SIGGRAPH)*. 23, 3, 574–583.

WANG, J., BHAT, P., COLBURN, R., AGARWALA M. And COHEN, M.F. 2005. Interactive Video Cutout. *ACM Trans. Graphics (Proc. SIGGRAPH)*. 24, 3, 585–594.

WINNEMOLLER, H., OLSEN, S. C., And GOOCH, B. 2006. Real-Time Video Abstraction. *ACM Trans. Graphics (Proc. SIGGRAPH)*. 25, 3, 1221–1226.

XU, S., Xu, Y., KANG, S. B., SALESIN, D. H., PAN, Y., AND SHUM, H.-Y. 2006. Animating Chinese Painting Through Stroke-based Decomposition. *ACM Trans. Graphics*. 25, 2, 239–267.

ZENG, K., ZHAO, M.T., And ZHU, S.C. 2009. From Image Parsing to Painterly Rendering. *ACM Trans. on Graphics*. 29. 1, 1–11.