

Painterly Rendering with Coherence for Augmented Reality

Jiajian Chen*

Greg Turk†

Blair MacIntyre‡

School of Interactive Computing
Georgia Institute of Technology

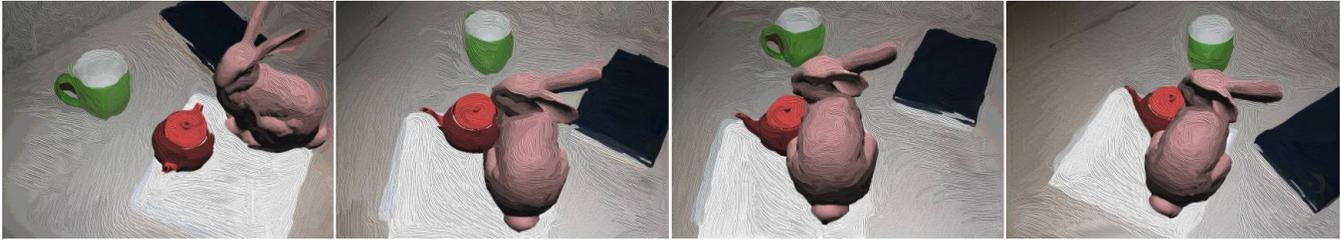


Figure 1: A sequence of painted AR frames. A real green cup, a real black cover book, a virtual teapot and a virtual bunny on a table.

ABSTRACT

A seamless blending of the real and virtual worlds is key to increased immersion and improved user experiences for augmented reality (AR). Photorealistic and non-photorealistic rendering (NPR) are two ways to achieve this goal. Non-photorealistic rendering creates an abstract version of both the real and virtual world by stylization to make them indistinguishable. We present a painterly rendering algorithm for AR applications. This algorithm paints composed AR video frames with bump-mapping curly brushstrokes. Tensor fields are created for each frame to define the direction for the brushstrokes. We use tensor field pyramids to interpolate sparse tensor field values over the frame to avoid the numeric problems caused by global radial basis interpolation in existing algorithms. Due to the characteristics of AR applications we use only information from the current frame and previous frame to provide temporal coherence in two ways for the painted video. First, brushstroke anchors are warped from the previous frame to the current frame based on their neighbor feature points. Second, brushstroke appearances are reshaped by blending two parameterized brushstrokes to achieve better temporal coherence. The major difference between our algorithm and existing NPR work in general graphics and AR/VR areas is that we use feature points across AR frames to maintain coherence in the rendering. The use of tensor field pyramids and extra properties of brushstrokes, such as cut-off angles, are also novel features that extend exiting NPR algorithms.

Index Terms: H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, Augmented, and Virtual Realities; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual Reality

1 INTRODUCTION

Photorealistic and non-photorealistic rendering are two major approaches to seamlessly blend the real and virtual world in AR.

*e-mail: jchen30@mail.gatech.edu

†e-mail: turk@cc.gatech.edu

‡e-mail: blair@cc.gatech.edu

Compared with photorealistic rendering [24], non-photorealistic rendering may be more appropriate to some AR applications. For example, an abstract stylization in AR hides unnecessary and overwhelming details of the world, and reveals and emphasizes important information of the scene. This could be particularly useful in some applications, e.g., AR/VR aided machine repair, or for virtual medical surgery.

Several NPR algorithms have been proposed for AR/VR. A major challenge for these rendering algorithms is maintaining temporal coherence for the composed AR video. Based on the literature in the AR/VR community we believe that this problem has not been solved yet. Most NPR techniques for AR/VR simply render each AR frame independently and do not consider temporal coherence at all. The resulting videos usually appear to flicker and are visually distracting.

The remainder of this paper is organized as follows: We briefly review NPR algorithms with coherence in general graphics and AR/VR areas in Section 2. We propose the framework of our algorithm and describe each important step in Section 3. We show results in Section 4 and we conclude the paper in Section 5.

2 RELATED WORK

There are a large number of NPR algorithms in the general graphics and AR/VR literature. The most applicable approaches related to our work are video NPR and real-time NPR, in which coherence is a key problem. Based on the type of input data and the solution methods for maintaining coherence, we divide the existing algorithms in the video and real-time NPR literature into two categories: algorithms in model space or in image space. In the image space approach the input is a sequence of camera images (video) or synthesized images. We do not have much knowledge about the geometry of the scene. In contrast, we have the 3D geometry of the scene as input in the model space approach. Different methods of maintaining temporal coherence are used in these two categories. We will discuss each of these in turn.

Model space algorithms create NPR effects for 3D models. There are two major types of algorithms in the model space approach for maintaining coherence: particle based algorithms and non-photorealistic textures. Particle based algorithms are studied by [23, 19, 22, 17]. Barbara Meier first presented an influential particle based NPR system that achieves high quality temporal coherence for animation. In her algorithm 3D particles are selected on the 3D model surfaces. Each particle is associated

with a 2D brushstroke texture. During rendering, these particles with brushstrokes are rendered from back to front. Kowalski et al. extended Meier’s work and uses stroke based procedural texture (“graftals”) to render fur, grass and trees without full geometry. Markosian et al. placed static graftals on surfaces during the modeling phase and the graftals are not redistributed each frame. Combining the work of Meier and Kowalski, Kaplan et al. proposed an interactive NPR system for model rendering with better coherence support.

Pre-built non-photorealistic textures can also be used in the model space approach when the 3D geometry is given. Klein et al. used a variation of mip-map texture (“Art Map”) to render a virtual environment with coherence [18]. Their approach applies NPR filters to stylize each level of the mip-map texture, and then maps the pre-built stylized textures on surfaces in rendering. Praun et al. presented a finer “Tonal Art Map” to extend Klein’s method [25].

In the image space approach, some researchers extended NPR techniques from still images to video and used optical flow to achieve temporal coherence [20, 16]. Some other researchers treated the video as a spatiotemporal volume and analyzed the video volume globally to obtain better coherence [3].

Litwinowicz first proposed the use of optical flow to improve coherence for video NPR. His algorithm uses the optical flow vector field as a displacement field to move the brushstrokes to new locations. This algorithm can provide a certain degree of temporal coherence for video NPR. However, new added strokes can appear in front of old strokes so the final result may scintillate.

Litwinowicz’s algorithm is the first approach in the literature that tries to use optical flow to maintain temporal coherence for video NPR. In general, this algorithm does fairly well but cannot be improved further without any advanced knowledge of the scene. Hertzmann et al. placed varied brushstrokes on still images by following normals of image gradients [13]. He extended his algorithm and Litwinowicz’s work to video and also used optical flow to maintain temporal coherence [16]. This paper presented two methods for keeping temporal coherence: a crude region based paint over method, and an optical flow based improved paint-over method. He also presented an energy function as a guide to place brushstrokes on canvas, and used optical flow to warp brushstroke control points to achieve coherence [14].

Optical flow has inspired many researchers to maintain temporal coherence for video NPR. Many NPR algorithms involve textures, such as brushstroke textures, that simulate the appearance of various NPR effects. Bousseau et al. produced a watercolor effect for video [1]. It uses two different approaches to achieve temporal coherence. To preserve coherence of the watercolor texture itself, it advects a set of pigmentation textures along lines of an optical flow field computed for the video.

Hays and Essa also used an improved optical flow method to process video with different artistic styles [12]. The key difference between their approach and Litwinowicz’s work is they add more properties such as opacity to the brushstroke. Also, their approach places temporal constraints on the brushstroke properties by interpolating gradient values to improve coherence.

Recently, researchers in graphics and computer vision have treated video sequences as spatiotemporal volumes by stacking sequential frames along the time-axis for analysis and post-processing. Video volume is then divided to semantic regions with coherence which can be further refined by user input. This approach improves coherence using global optimization, compared to optical flow that typically finds only pixel motion correlation between consecutive frames. Comanicu et al. developed a spatiotemporal segmentation technique (“kernel mean shift”) for images [4]. Wang et al. proposed “anisotropic kernel mean shift” to extend this algorithm to video [26]. The segmentation divides the video volume into pixel clusters with similar visual properties and these

clusters/regions have a low spatiotemporal coherence. He also presented a semi-automatic system that can transform a video into a cartoon-like style [27]. Their approach overcomes coherence problems by accumulating the video frames to a 3D volume and cluster pixels in spatiotemporal space. Collomosse et al. improved Klein and Wang’s approaches and presented a more general NPR system (“Stroke Surface”) that can create abstract animation from video with high temporal coherence [3].

NPR has also been studied and used in the AR/VR community. Fischer et al. presented several NPR styles, such as cartoon style, for AR applications [6, 5]. Haller et al. compared photorealistic rendering and NPR for MR [10, 8]. He also presented a sketch style rendering for AR [9]. Chen et al. presented a watercolor-like style with some degree of coherence for AR [2]. However, not much research has been done for providing temporal coherence for real-time AR NPR. A frame from an AR video is typically a mixture of 3D virtual image and a real world image, so naturally the algorithms in the image space and model space approaches discussed in section 2 should work. However, there are several differences between NPR for AR and NPR for video and 3D models. First, AR applications usually require real-time methods and cannot directly use the existing video NPR algorithms, which need the information beyond the current frame. Second, it is difficult to recover the scene geometry from AR video.

We propose a painterly rendering algorithm that demonstrates significant coherence for augmented reality. Our algorithm creates a painterly rendering effect for composed AR video by generating a tensor field and placing bump-mapping curly brushstrokes on the frames. It uses the correspondences of feature points across frames to warp and reshape the brushstrokes to maintain the frame-to-frame coherence. We choose the painterly rendering style in our system, but the algorithm of maintaining temporal coherence may be extended to apply to other NPR styles for AR.

3 PAINTERLY RENDERING WITH COHERENCE

The flowchart of our painterly rendering algorithm is shown in Figure 2 below.

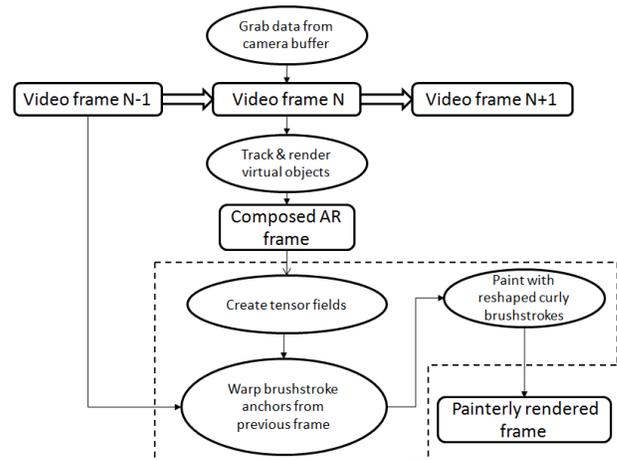


Figure 2: Algorithm flowchart. Three major steps are enclosed by the dash lines.

We use multiple layer painting to compose the final result. In the tensor field creation, we divide the input frame into regions and label each region as ‘highly-detailed’, ‘middle-detailed’ and ‘texture-less’ based on each region’s overall edge magnitude. We use different sets of properties for brushstrokes to paint these regions from the coarse layer to fine layer. During the painting, we allow a brushstroke from a coarse layer to cut across regions

in the fine layer, but do not allow in the opposite direction. In the following sections, we will discuss the creation of the tensor field, the warping and the reshaping of brushstrokes in turn.

3.1 Tensor Field Creation

To guide the orientation of brushstrokes, we first compute the tensor field on a composed AR frame. One of the major advantages of a tensor field over a vector field is it allows direction ambiguity. The tensor field at a pixel does not distinguish between the direction of θ and $\theta + \pi$ [28]. In the vector field, the magnitude P and orientation θ of a pixel can be computed by the following image gradient operator:

$$G_x = 2I(x+1,y) - 2I(x-1,y) + I(x+1,y+1) - I(x-1,y+1) + I(x+1,y-1) - I(x-1,y-1) \quad (1)$$

$$G_y = I(x-1,y+1) + 2I(x,y+1) + I(x+1,y+1) - I(x-1,y-1) - 2I(x,y-1) - I(x+1,y-1) \quad (2)$$

$$P = \sqrt{G_x^2 + G_y^2} \quad (3)$$

$$\theta = \arctan(G_y/G_x) \quad (4)$$

$I(x,y)$ is the gray scale intensity of a pixel at (x,y) . Note that many other edge detection algorithms in computer vision literature, such as the Harris corner detector [11], will also work. In our experiment, this operator from Canny edge detection is relatively fast and produces good enough initial tensor fields. Once the edge magnitude P and orientation θ of a pixel is computed, its tensor field M is defined as follows:

$$M = P \begin{bmatrix} \cos 2\theta & \sin 2\theta \\ \sin 2\theta & -\cos 2\theta \end{bmatrix} \quad (5)$$

The eigenvector of this matrix M 's major eigenvalue gives the orientation θ for this pixel.

To reduce the image noise we do a thresholding on pixel edge magnitude P . After thresholding, we have a tensor field map associated with the video frame. Each strong edge pixel whose magnitude is over the threshold has been assigned a tensor field value M .

For the pixels whose magnitude is below the threshold, we can use global radial basis interpolation to compute their tensor fields [12]. A linear interpolation of the vector field usually generates the singularity points, where the magnitude of the vector is zero if two vector fields have opposite directions. Tensor fields can eliminate this problem because the interpolation of two tensor fields with opposite orientations does not produce zero. The interpolated tensor field value for a non-edge pixel is computed as follows:

$$M = \sum_{i \in \text{edge pixels}} e^{-\frac{d}{\sigma}} P_i \begin{bmatrix} \cos 2\theta_i & \sin 2\theta_i \\ \sin 2\theta_i & -\cos 2\theta_i \end{bmatrix} \quad (6)$$

The exponential component is the weight that controls the contribution of a strong edge pixel to the tensor field. d is the Euclidean distance between an edge pixel $P(x_i, y_i)$ and the non-edge pixel $P(x, y)$. σ controls the width of the attenuation window. Small σ makes the tensor field smoother, but it is possible to lose local details. Also the radial basis interpolation needs to loop through all strong edge pixels to compute the weighted sum.

Usually this interpolation will become a computational bottleneck if there are a large amount of edge pixels initially detected. In practice, it may also give a tensor field of zero for non-edge pixels due to large textureless regions without strong edges. For example, if the distance between a non-edge pixel and all edge pixels is too big, its tensor field M will be zero. In the final painting

these unassigned regions become holes that are not covered by any brushstrokes. Also, the initial tensor field data usually is non-uniformly distributed. A region that has dense edge pixels (e.g., a tree) may strongly bias the interpolated results.

To address these problems in the global radial basis interpolation, we borrow an idea from Gortler et al. [7]. In this paper an image pyramid is built and a pull-push algorithm is used to interpolate the sparse lumigraph data. Similarly, we build a pyramid of the tensor field map. The algorithm is listed below.

1 Divide the original video frame to grid-based regions, and sum and normalize the tensor values in each region.

2 For a region that's not assigned a tensor value, use radial basis interpolation to compute its tensor, but we search only a small 5×5 window patch centered at this region. Apply low-pass Gaussian filter to the tensor value, and then sub-sample to create a new tensor field map at a higher level.

3 Repeat Step 2, until we have reached the top of the pyramid and built the tensor field of size 1.

Note the difference between our method and Gortler's original interpolation method in step 2. Gortler's original method averages and normalizes data in regions at each level. In our method at each level of the tensor field pyramid, first we search a 5×5 window centered at a region to compute its tensor field. We then apply a Gaussian filter to create a new higher level of tensor field map. The purpose is that we want to keep fine local details for later painting. Local tensor values near to a strong edge should all follow the direction of that edge. This makes the brushstrokes in the area look smoother. Figure 3 illustrates the creation of the tensor field pyramid.

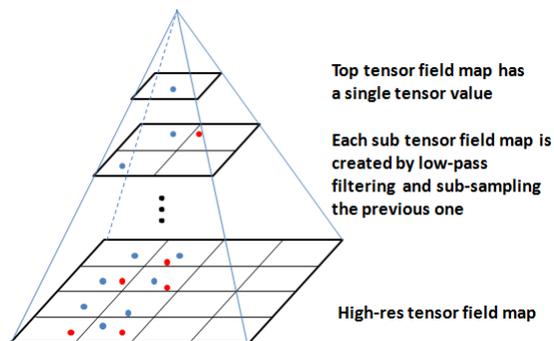


Figure 3: Three levels of the tensor field pyramid, created from bottom to top. At each level, blue dots are the tensor fields from strong edge pixels in regions, and red dots are the tensor fields created by interpolation in the region's local patch window. A higher level is created by applying a Gaussian low pass filter to the lower level and sub-sampling.

Once the pyramid is created, we can get the tensor field for any pixel in a frame. During the painting, if we need a tensor field of a pixel, first we check if it already has a tensor value assigned. If it does, we just use the local tensor, otherwise we go up in the tensor field pyramid, and check if the corresponding region containing this pixel has a tensor value in this upper level.

This tensor field pyramid algorithm has several advantages over the original global radial basis interpolation. First, we reduce the influence of non-uniformly distributed data by dividing the tensor map into regions and normalize the tensor field in each region. Second, it is much faster than a simple global interpolation, since it only needs to check a 5×5 window for the unassigned regions at each level, instead of all edge pixel/regions. The small window size also produces better interpolation results for our painting purpose

(e.g., the tensor field of pixels that are close to a strong edge will strictly follow its tensor value). Hence it preserves fine local details better. Third, it avoids the problem of holes in textureless regions.

In our multi-layer painting, we divide a frame into regions based on the layer index, and then sum the tensor values in each region. The sum of the tensor values in a region gives the direction of a brushstroke across this region.

Figure 4 shows a visualization of the tensor field for a static image. A short line segment is drawn along the direction of the tensor field at each anchor point in the image.

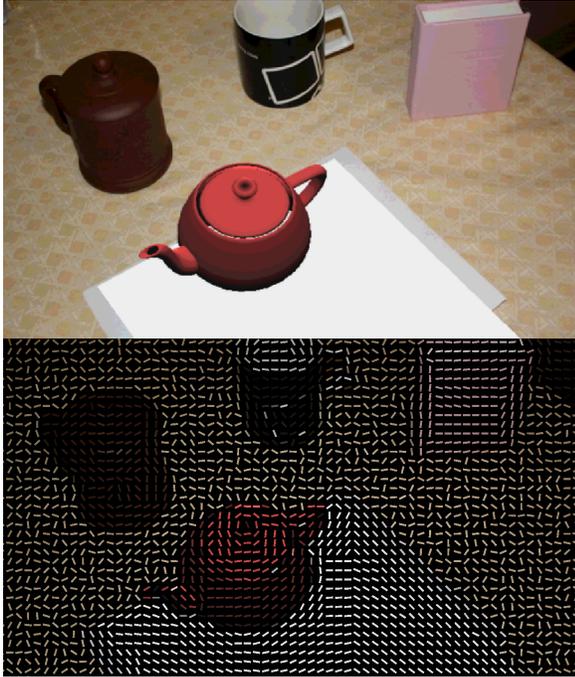


Figure 4: Tensor field visualization of an AR frame. There are a virtual teapot, two real cups and an album on the table.

3.2 Feature Point Based Brushstroke Anchor Warping

In order to maintain coherence we need to move and repaint brushstrokes based on the correspondences across frames. We track feature points across frames. For the non-feature pixels in the frame, we warp the brushstrokes that are located at these pixels from frame to frame. The warping is done using barycentric coordinates.

3.2.1 Initialization of Brushstrokes at the First Frame

For multi-layer painting, we place brushstrokes in each layer. The first frame is evenly divided into $M \times N$ grid regions. The initial anchor points of the brushstrokes at each layer are the centers of these regions.

3.2.2 Feature Points Tracking

We use video feature tracking to help produce brushstrokes that are coherent between frames. In general any tracking methods that generate feature points across frames can be used. In our system we use SIFT to build gradient-histogram descriptors for tracking feature points from frame to frame [21]. The features points we choose are Harris corners [11].

3.2.3 Brushstroke Anchor Warping

A major constraint for coherence processing due to the characteristics of AR is we cannot go further beyond the current frame or apply global coherence optimization for the whole video [3]. Most of the time we can only use information from the previous frame and current frame.

To maintain the temporal coherence for brushstrokes we warp their anchor points from the previous frame to the current frame in barycentric coordinates. Assume there are a set of matched feature points between two consecutive frames. For a brushstroke starting at position P in the previous frame, we find its nearest three neighbor feature points, and compute P 's barycentric coordinates C with respect to these three points. We then find the correspondent matches of these three points in the second frame. We warp this point P to the current frame, whose position P' is given by $[P'_1 \ P'_2 \ P'_3] \times C$. P' is the new location of the brushstroke. The procedure is shown below, assuming the anchor is P and its 3 nearest feature points are P_1, P_2, P_3 .

$$P = uP_1 + vP_2 + wP_3, \text{ where } u + v + w = 1 \quad (7)$$

$$\begin{bmatrix} P_1 & P_2 & P_3 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} P \\ 1 \end{bmatrix} \quad (8)$$

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} P_1 & P_2 & P_3 \\ 1 & 1 & 1 \end{bmatrix}^{-1} \begin{bmatrix} P \\ 1 \end{bmatrix} \quad (9)$$

A close form solution of the barycentric coordinates of P is:

$$u = \frac{(y_2 - y_3)(x - x_3) + (x_3 - x_2)(y - y_3)}{\det T} \quad (10)$$

$$v = \frac{(y_3 - y_1)(x - x_3) + (x_1 - x_3)(y - y_3)}{\det T} \quad (11)$$

$$w = 1 - u - v, \text{ where } T = \begin{bmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{bmatrix} \quad (12)$$

Figure 5 illustrates the warping for a single brushstroke.

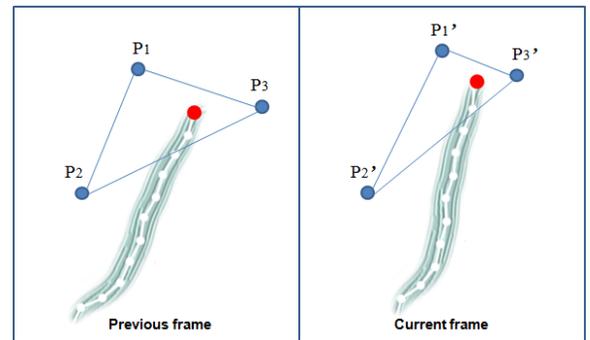


Figure 5: Warping of a brushstroke anchor point P (red dot).

When the scene is in motion, the warped brushstrokes could become over dense or sparse in some area. The over dense brushstrokes in an area may cause unnecessary repainted in this area, while the sparse brushstrokes may not cover the area and produce holes. We need to remove or add new brushstrokes in these cases. In the multi-layer painting, the frame is divided into regions. We use an array to record the number of brushstrokes in each region. If there are not enough brushstrokes we evenly-subdivide

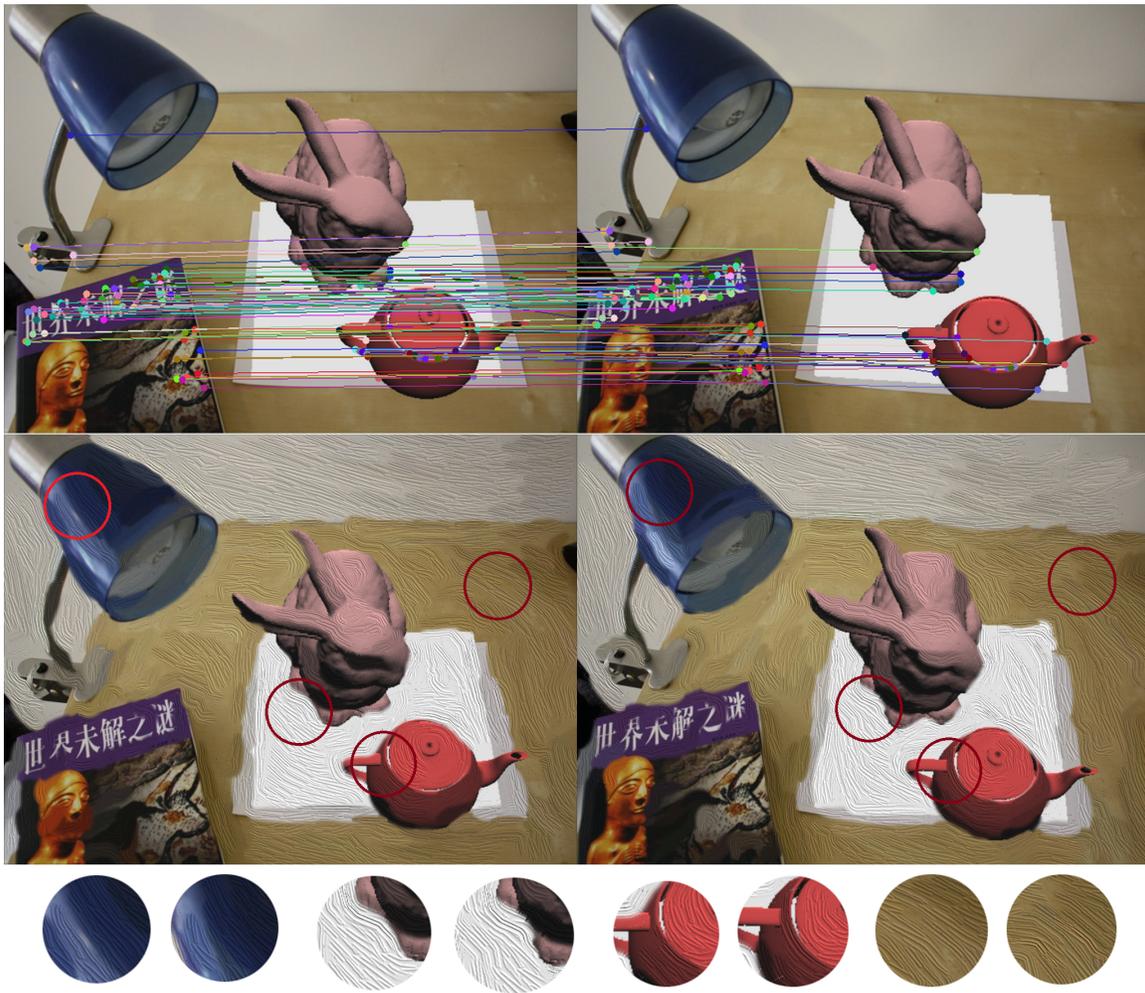


Figure 6: Top row: Matched SIFT feature points in two frames. Feature points are Harris corners. 256 features detected in the left frame, and 206 features detected in the right frame. ($CornerThreshold = MaxCornerStrength/5$). 80 matches are found. Matched features are linked with colored lines. Bottom row: Painted results for these two frames. Brushstrokes in four correspondent areas are shown at the bottom. Notice that in some areas the brushstroke anchors are inside the triangles formed by neighbor feature points, as well in some areas the anchors are outside the feature point triangles.

this region, and then add new brushstrokes to each sub region, if needed. Similarly we can remove brushstrokes in over dense areas. After adding or removing brushstrokes we correspondingly update the list of brushstroke anchors.

The correctness of the anchor point warping is based on the following observation: when the camera is far enough from the scene, if a non-feature point P is co-planar with some feature points in the previous frame, we can compute the correspondence of P in the current frame by warping. The warping gives the new anchor position of an existing brushstroke, and makes it coherent from frame to frame. Optical flow can also warp brushstroke anchors by estimating the movement of any individual pixel. Optical flow based brushstroke anchor warping is suitable for arbitrary video where precise tracking could be unavailable, as shown in [3, 12]. The feature point tracking based brushstroke anchor warping gives more stable results in AR applications since we directly have feature tracking in the scene. In our algorithm, a brushstroke anchor P could be outside of the triangle formed by its three nearest neighbor feature points. In this case the warping still works if P is close to these neighbors (i.e., these four points are approximately co-planar). Otherwise it may give undesirable results. Figure 6

shows an example from two consecutive AR frames. The matched SIFT feature points across frames are linked by colored lines. The anchors of painted brushstrokes are correspondingly warped based on these matches. Four corresponding areas from the painted results are shown at the bottom. Notice that in some areas the brushstroke anchors are inside of the triangle formed by neighbor feature points, but in some other areas they are outside of the feature point triangles. Although the later case is undesirable, the warping still works when the camera is far enough from the scene.

Notice that we use SIFT tracking to find matches feature points in the previous and current frames. Tracked feature points in frame n and $n + 1$ may disappear in frame $n + 2$, but we are still able to warp brushstroke anchors since we can find another set of matches in frame $n + 1$ and $n + 2$. If the tracking is temporarily lost during the application we can use the latest list of brushstroke anchors as the starting points to paint the new frames. Once the tracking works again we continue warping the brushstroke anchors if a large number of matched feature points are found in the new frame, or re-initialize all brushstroke anchors and restart warping.

3.3 Bump-mapping Curly Brushstroke

We create long curly brushstrokes based on the algorithm of Hertzmann [13]. The difference between our algorithm and Hertzmann’s original one is that we use bump mapping to give the appearance of textured brushstrokes. Hertzmann also proposed to use a height map to create lighting effects, but it is not directly used in the creation of a single curly brushstroke [15]. Another difference is we choose different cut-off angles as the stop condition for curly brushstrokes in each painting layer, compared to [12].

For each brushstroke we define a set of properties such as the width, length of a single travel step, minimum and maximum travel steps, and color difference threshold, similar to [16, 12]. To reduce the curvature change in the brushstroke we also define a cut-off angle as an extra property. We start from an anchor point to compute each curly brushstroke. To elongate a brushstroke we travel a short distance in the tensor field direction and this brings us to a new region. The tensor field direction is ambiguous by a half-turn, so we select the direction that will not cause a kink in the stroke, thus minimizing stroke curvature. We repeatedly extend the stroke until the color in the new region varies too much from the color in the starting region. We also stop if the difference of two tensor field directions is bigger than the cut-off angle. A coarse (fine) layer has a smaller (bigger) cut-off angle and bigger (smaller) color difference threshold. The color difference can be measured in RGB or HSV space.

In each step along the stroke we record the position of the regions as control points. Once we have all the control points $C_0, C_1, \dots, C_{n-1}, C_n$, we compute a cubic B-spline curve based on these control points. To force the result curve to interpolate at the start and end points, we use triple knots at C_0 and C_n . As a result, the input control point set for the cubic B-spline interpolation is $C_0, C_0, C_0, C_1, C_2, \dots, C_{n-2}, C_{n-1}, C_n, C_n, C_n$. Correspondingly we will get $n+2$ segments. In each segment we choose 4 sample points at $t = 0, \frac{1}{4}, \frac{1}{2}$ and $\frac{3}{4}$. In the final painting we will have $4(n+2) + 1$ sample points. We divide the curve to $4(n+2)$ subcurves and draw each subcurve as a quad. Figure 7 shows a parameterized brushstroke with 11 control points.

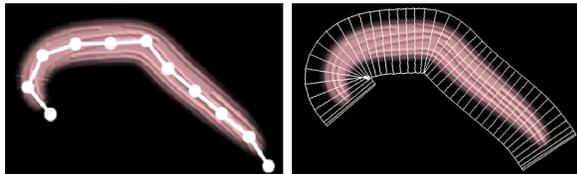


Figure 7: Left: A curly brushstroke with 11 control points (triple knots at two ends). Right: The brushstroke is divided to subcurves and rendered.

Figure 8 shows two of the various styles of brushstrokes used in our paper, each of which has an alpha mask and brushstroke texture.

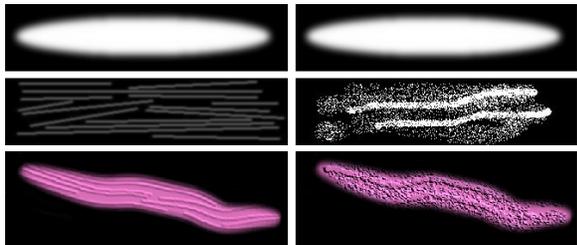


Figure 8: Each column is a type of a brushstroke. From top to bottom: an alpha mask, brushstroke texture, and example of a final composed brushstroke on screen.

Figure 9 visualizes the brushstroke skeleton curves. Each brushstroke is rendered as a colored curve.



Figure 9: Brushstroke skeleton visualization on the coarse layer

To obtain better visual results, we use bump mapping to paint each brushstroke, instead of texture mapping. Each brushstroke also has an alpha mask to mimic the smooth and natural brushstroke shape from a human artist’s painting. This is also used in [12]. Bump-mapping can simulate the subtle lighting effects for the brushstroke and drastically improves the visual quality of the final painting. We implement bump mapping using GLSL on the GPU for speed. To get correct lighting, we pass the orientation of each painting segments (“subcurve quad”) as a tangent vector to the vertex shader. In the fragment shader, the surface normal is estimated from a gray-scale brushstroke texture (e.g., Figure 8 middle row).

$$\text{vec3 } BumpNorm = \text{vec3}((\text{left}.x - \text{right}.x), (\text{bottom}.x - \text{top}.x), 1);$$

$$BumpNorm = \text{normalize}(BumpNorm);$$

The ‘left, right, bottom, top’ variables are the colors of the adjacent pixels in the texture. Multiple brushstrokes could be placed across a same area on a frame, which causes unnecessary repainting. To avoid this problem, we choose a proper width for each brushstroke. Once a brushstroke is drawn, we mark the regions covered by this brushstrokes as ‘painted’.

Figure 10 shows the result of the final painting. Note bump-mapping simulates the lighting effect of artistic painting and adds fine details to the result.

3.4 Coherent Curly Brushstrokes

In order to keep temporal coherence we warp the anchor points of brushstrokes from the previous frame to current frame. However, repainting a brushstroke at the corresponding anchor points at the current frame sometimes produces bad coherent results. The computation of a brushstroke curve is decided by the local tensor field and color difference of regions around the anchor. The shape of a curly brushstroke in two frames could vary significantly, even if its anchor points are ‘coherent’. Hence it is also necessary to change the shape of a brushstroke in the current frame to make it similar to the previous one.

Our assumption is that a coherent brushstroke in two consecutive frames should have similar shapes that do not vary too much. To make the shape of a brushstroke in the current frame resemble the previous one, we first compute a new curly brushstroke in the current frame, and then we blend it with the corresponding brushstroke from the previous frame that has the same anchor point. Both brushstrokes are parameterized. We compute the average of the two parameterized brushstrokes, and then use the blended brushstroke for painting in the current frame. If the two



Figure 10: Final painting with curly brushstrokes. Note the fine details in the painting. Three different styles of brushstrokes at the coarse and fine level are shown at the bottom row.

brushstrokes do not have a same number of control points, we resample both of them to obtain a same number of sample points, and then compute the average. Figure 11 shows the blending method.

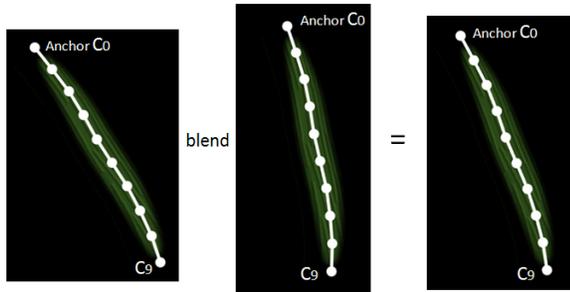


Figure 11: Blending of two parameterized brushstrokes. The left two brushstrokes locate at two correspondent anchors C_0 on two consecutive frames. Each brushstroke has 10 control points.

4 RESULTS

We applied our painterly rendering algorithm to both static images and AR videos. The results are shown in Figure 12 and Figure 13 in the color plate page at the end. The static images are comparable to Hays and Essa’s work [12], which is one of the highest quality methods that we know. We also show a sequence of frames from an AR video to demonstrate the coherent painting effects.

Currently our algorithm is focused on producing coherent painterly rendering effects for AR and it is not running in realtime. In our system the average processing time per AR frame is divided into four major stages: tensor field creation, Harris corner detection and SIFT matching, brushstroke anchor warping, and final painting (placing reshaped bump mapping brushstrokes on to the frame). We tested the speed on a PC with Core2Duo 2GHz CPU and ATI5730 graphics card. The result is shown in Table 1. The bottleneck is the SIFT matching where we run ratio test for all detected corner pairs, and brushstroke anchor warping where we find nearest neighbors for each brushstroke anchor and compute the barycentric coordinates. However, the original SIFT ratio test can be replaced with a faster version. The neighbor finding can also be accelerated

by using KD-tree-like data structure. These improvements may lead to the realtime performance in the future.

Table 1: Average Processing Time per AR frame. (Unit: ms)

Frame Resolution	Stage 1	Stage 2	Stage 3	Stage 4	Total
1440 × 1050	75	420	215	30	740
1024 × 768	50	173	125	27	375

5 CONCLUSION

We have presented an NPR algorithm that creates painterly rendering effect for AR applications. Our method improves the rendering result by providing visual coherence in two aspects. The final painting result is created by placing bump-mapping curly brushstrokes on a composed AR frame. The placement of brushstrokes is guided by tensor fields for each frame. Temporal coherence is maintained by warping the brushstroke anchors and reshaping the brushstrokes from the previous frame to current frame. We track feature points across frames, and brushstroke anchor warping is done by using the three nearest neighbor feature points in barycentric coordinates. To our knowledge this is the first attempt to achieve coherence of NPR stylization for AR.

The algorithm processes AR video with coherence in image space, as we categorized in Section 2. We choose painterly rendering style in our system, but the algorithm may be extended to apply to other NPR styles for AR. In the future we want to try methods that operate in model space. This could be done by recovering simple scene geometry and processing both the real and virtual scenes in model space. Investigating other NRP styles with coherence is another possible research direction.

REFERENCES

- [1] A. Bousseau, F. Neyret, J. Thollot, and D. Salesin. Video watercolorization using bidirectional texture advection. *ACM Transaction on Graphics*, 26(3):104, 2007.
- [2] J. Chen, G. Turk, and B. MacIntyre. Watercolor inspired non-photorealistic rendering for augmented reality. In *VRST*, pages 231–234. ACM, 2008.
- [3] J. P. Collomosse, D. Rowntree, and P. M. Hall. Stroke surfaces: Temporally coherent artistic animations from video. *IEEE TVCG*, 11(5):540–549, Sept./Oct. 2005.
- [4] D. Comanicu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE PAMI*, 24(5):603–619, 2002.
- [5] J. F. Dirk and D. Bartz. Real-time cartoon-like stylization of AR video streams on the GPU. In *TR WSI-2005-18*, University of Tübingen.
- [6] J. Fischer, D. Bartz, and W. Straßer. Artistic reality: fast brush stroke stylization for augmented reality. In *VRST*, pages 155–158, 2005.
- [7] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *SIGGRAPH*, pages 43–54, 1996.
- [8] M. Haller. Photorealism or/and non-photorealism in augmented reality. In *ACM International Conference on Virtual Reality Continuum and its Applications in Industry*, pages 189–196, 2004.
- [9] M. Haller and F. Landerl. A mediated reality environment using a loose and sketchy rendering technique. In *ISMAR*, pages 184–185, Washington, DC, USA, 2005. IEEE Computer Society.
- [10] M. Haller and D. Sperl. Real-time painterly rendering for mr applications. In *International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia*, 2004.
- [11] C. Harris and M. Stephens. A combined corner and edge detector. *4th Alvey Vision Conference*, pages 147–151, 1988.
- [12] J. Hays and I. A. Essa. Image and video based painterly animation. In *NPAR*, pages 113–120. ACM, 2004.
- [13] A. Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *SIGGRAPH*, pages 453–460, 1998.

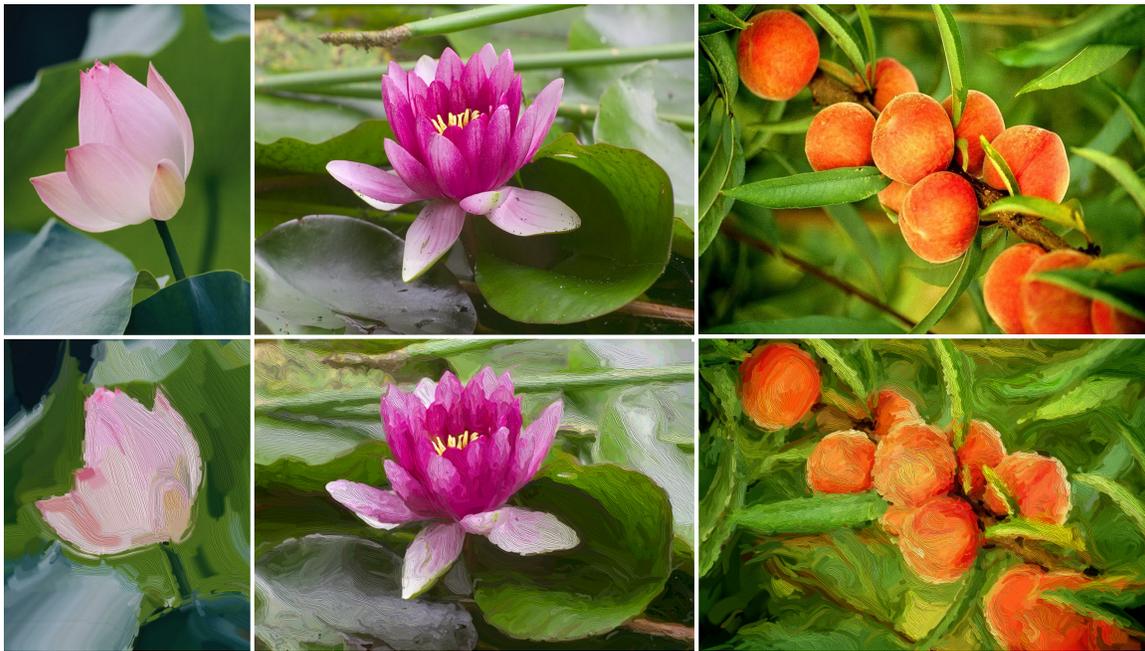


Figure 12: Painting results: static images comparable with Hays and Essa's work [12].

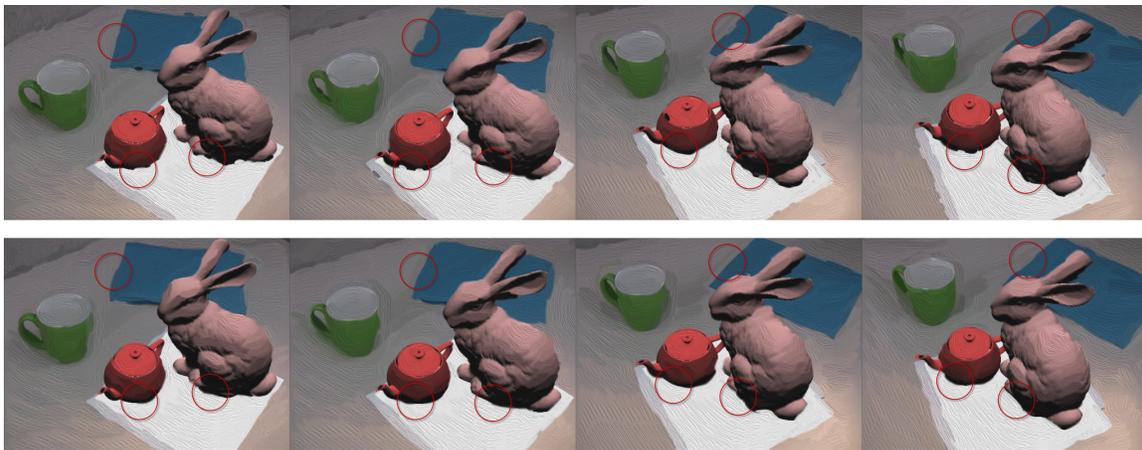


Figure 13: Top row: painted AR frames without coherence processing. Notice the brushstrokes in circled areas appear and then disappear between frames. Bottom row: the same sequence of frames with coherence processing. Notice coherent brushstrokes in circled areas.

- [14] A. Hertzmann. Paint by relaxation. In *Proceedings of Computer Graphics International Conference*, pages 47–54, July 3–6 2001.
- [15] A. Hertzmann. Fast paint texture. In *NPAR*, page 91, 2002.
- [16] A. Hertzmann and K. Perlin. Painterly rendering for video and interaction. In *NPAR*, pages 7–12, 2000.
- [17] M. Kaplan, B. Gooch, and E. Cohen. Interactive artistic rendering. In *NPAR*, pages 67–74, 2000.
- [18] A. Klein, W. Li, M. M. Kazhdan, W. T. Corrêa, A. Finkelstein, and T. A. Funkhouser. Non-photorealistic virtual environments. In *SIGGRAPH*, pages 527–534, 2000.
- [19] M. A. Kowalski, L. Markosian, J. D. Northrup, L. Bourdev, R. Barzel, L. S. Holden, and J. Hughes. Art-based rendering of fur, grass, and trees. In *SIGGRAPH*, pages 433–438, 1999.
- [20] P. Litwinowicz. Processing images and video for an impressionist effect. In *SIGGRAPH*, pages 407–414, 1997.
- [21] D. G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, pages 1150–1157, 1999.
- [22] L. Markosian, B. J. Meier, M. A. Kowalski, L. Holden, J. D. Northrup, and J. F. Hughes. Art-based rendering with continuous levels of detail. In *NPAR*, pages 59–66, 2000.
- [23] B. J. Meier. Painterly rendering for animation. In *SIGGRAPH*, pages 477–484, 1996.
- [24] S. A. Pessoa, G. de S. Moura, J. P. S. M. Lima, V. Teichrieb, and J. Kelner. Photorealistic rendering for augmented reality: A global illumination and BRDF solution. In *IEEE VR*, pages 3–10, 2010.
- [25] E. Praun, H. Hoppe, M. Webb, and A. Finkelstein. Real-time hatching. In *SIGGRAPH*, page 581, 2001.
- [26] J. Wang, B. Thiesson, Y. Q. Xu, and M. Cohen. Image and video segmentation by anisotropic kernel mean shift. In *ECCV*, pages Vol II: 238–249, 2004.
- [27] J. Wang, Y. Xu, H.-Y. Shum, and M. F. Cohen. Video tooning. *ACM Transactions on Graphics*, 23(3):574–583, Aug. 2004.
- [28] E. Zhang, J. Hays, and G. Turk. Interactive tensor field design and visualization on surfaces. *IEEE TVCG*, 13(1):94–107, 2007.