To appear, First International Symposium on Non-Photorealistic Animation and Rendering, June 5-7, 2000

Painterly Rendering for Video and Interaction

Aaron Hertzmann   Ken Perlin[*]

Media Research Laboratory
Department of Computer Science
New York University

Abstract

We present new methods for painterly video processing. Based on our earlier still image processing technique, we "paint over" successive frames of animation, applying paint only in regions where the source video is changing. Image regions with minimal changes, such as due to video noise, are also left alone, using a simple difference masking technique. Optionally, brush strokes may be warped between frames using computed or procedural optical flow.

These methods produce video with a novel visual style distinct from previously demonstrated algorithms. Without optical flow, the video gives the effect of a painting that has been repeatedly updated and photographed, similar to paint-on-glass animation. We feel that this gives a subjective impression of the work of a human hand. With optical flow, the painting surface flows and deforms to follow the shape of the world.

We have constructed an interactive painting exhibit, in which a painting is continually updated. Viewers have found this to be a compelling experience, suggesting the promise of non-photorealistic rendering for creating compelling interactive visual experiences.

CR Categories: I.3.3 [Picture/Image Generation]: Computer Graphics—Display Algorithms; J.5 [Computer Applications]: Arts and Humanities—Fine Arts

Keywords: Non-photorealistic rendering, painterly rendering, video processing, animation

[*]WWW: http://www.mrl.nyu.edu/{hertzmann,perlin}

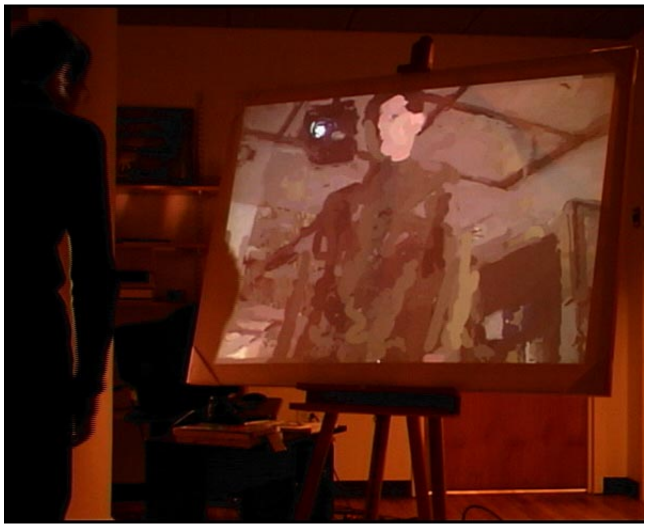

Figure 1: A viewer interacting with a "living" painting.

1 Introduction

Graphics researchers have demonstrated many techniques for producing dazzling and expressive still non-photorealistic images, allowing users to quickly create images that would previously require the labors of a skilled artist. Of course, these systems do not replace artists for the same reasons that cameras did not.

This paper builds on the still image research to address the problem of producing non-photorealistic animations and interactions. Here, instead of making a task faster and easier, we are focused on making a task *possible*. Some of the innovations required for non-photorealistic interaction are purely algorithmic (e.g. silhouette detection [13, 5]). In other cases, technological problems are tightly coupled to artistic problems (e.g. [2]). In particular, we are concerned with the question: what can a painterly animation look like? This paper presents several new tools that can be used in exploring this design space. Similar explorations have occurred in experimental animation, although such work is limited by the extreme amount of effort required to hand paint each frame.

Previous research can be categorized as either generating representations of a painted world, or painted representations of a world. The first of these approaches attaches stroke positions and sizes over time to 3D geometry [14, 12, 4]. This usually gives the appearance of a "Painted World," i.e. a world inhabited by brush strokes. In contrast, Litwinowicz [11] and Curtis [2] produce painted representations of the world, by allowing strokes to detach from geometry, in both space and time. We continue to explore this latter line of research.

Our goal in this research is to make tools to allow artists to create painterly animations and interfaces. For many applications, non-photorealistic animations and interfaces provide distinct advantages: visual appeal, the visual and emotional expression of physical media (e.g. brush strokes), informal and compelling interfaces, reduced requirements for geometric modeling (when processing 3D), and cultural references (e.g. animating existing paintings and styles).

1.1 Related Work

We now give a brief overview of research directly related to this paper; see [3] for a more thorough survey of non-photorealistic rendering. Haeberli [6] described automatic and semi-automatic painterly rendering algorithms, extensions of which are now commonly used in commercial desktop publishing software. Meier [14] attaches particles to 3D objects by an automatic procedure, and places brush strokes to coincide with the particles; the Deep Canvas system [4] allows a skilled artist to paint curved strokes onto a model, and then animate the painting. Litwinowicz [11] uses optical flow to push short brush strokes along scene movements, and provides various tools [12] for editing/correcting flow and layering. Hertzmann [7] automatically paints still images with various stroke sizes and shapes; in this paper, we extend this method to processing video.

Previous work has applied interactive non-photorealistic rendering to 3D modeling [18, 8], technical illustration [13, 5, 15], education and entertainment [9]. The methods in this paper are aimed at the last of these goals: building tools that will allow artists to create compelling, enjoyable, and expressive video and interfaces.

1.2 Overview

This paper is organized as follows. We begin by reviewing our still image processing system, and show how this system is extended to video. We then describe difference masking techniques for reducing flickering in video, and the use of optical flow to push strokes so as to follow the scene motion. Several video experiments and an interactive exhibit are described, followed by discussion and future work.

2 Still Images

In this section, we review a revised version of our still image processing algorithm; a detailed description can be found in [7]. This version uses a summed-area table [1] to quickly compute blurred images. Furthermore, the paint function takes an extra parameter, which allows us to choose whether to force the first paint layer to cover the canvas.

The algorithm paints a rough sketch of the image with large brush strokes, and then refines it with smaller brush strokes, but only in regions where the painting does not closely match the source image (e.g. where there is fine detail in the source).

```
function paint($I_s$, // source image
               $I_p$, // canvas; initially blank for still images
               $R_1 ... R_n$, // brush sizes
               firstFrame) // boolean; true for still images
  Create a summed-area table $A$ from $I_s$
  refresh ← firstFrame
  foreach brush size $R_i$, from largest to smallest, do
    Use $A$ to compute a blurred reference image $I_{R_i}$
    grid ← $f_g R_i$
    Clear depth buffer
    foreach position $(x, y)$ on a grid with spacing grid
      M ← the region $[x - grid/2 ... x + grid/2,$
                      $y - grid/2 ... y + grid/2]$
      areaError ← $\sum_{(i,j) \in M} ||I_p(i,j) - I_{R_i}(i,j)||$
      if refresh or areaError $> T$ then
        $(x_1, y_1) \leftarrow \arg\max_{(i,j) \in M} ||I_p(i,j) - I_{R_i}(i,j)||$
        paintStroke($x_1, y_1, I_p, R_i, I_{R_i}$)
    refresh ← false
```

paintStroke is the contour-following procedure defined in [7]. It produces a set of sparse control points, from which a smooth curve is computed by cubic B-spline subdivision. Strokes are painted as triangle strips using graphics hardware. Strokes are given random depth-buffer values to simulate the appearance of painting the strokes in random order. In this paper, $||\cdot||$ denotes Euclidean distance in RGB space. f_g is a constant factor defined by the painting style.

3 Video

3.1 Painting Over

The simplest method for generating painterly video is to apply a still image filter to each frame independently. As has been previously observed in the literature, subtle changes in the input can cause dramatic changes in the output, creating severe flickering in the output video. This flickering can be characterized as static areas of the scene that are painted differently in each frame. This flickering changes the character of the input and distracts from the action in ways that are usually undesirable.

The algorithm presented in the previous section leads to a natural approach to improving temporal coherence. The first frame of the video sequence is painted normally. For each successive frame, we "paint over" the previous frame[1]. This means that the painting of the first frame is used as the initial canvas for the second frame. The *paint* procedure is called with this initial canvas and *firstFrame* is set to **false**. Consequently, unchanging regions of the video frame will be left unchanged when the painting style is reasonably faithful to the source image.

This method produces video with the appearance of a painting that has been repeatedly painted over and photographed. This style is similar to the paint-on-glass style in experimental animation, such as the work of Alexander Petrov.

Though painting-over does improve temporal coherence, flickering remains a problem, when a static region of a video frame differs from the corresponding region of the previous painted frame. This discrepancy can occur in several cases. A significant source of error is video noise, which can cause static areas of the scene to be repainted each frame. Another source of difference is stylization and "sloppiness" in the painting: many painting styles produce images that radically diverge from the input. We address this problem by performing *difference masking*: we paint only in portions of the video which contain significant motion. In offline processing, we measure the sum of the differences between an image region for the current frame and the corresponding region for the previous frame. In terms of the above pseudocode, this amounts to adding the following test to the inner loop of *paint*:

$$\frac{1}{|M|} \sum_{(i,j)\in M} ||I_{t+1}(i,j) - I_t(i,j)|| > T_V$$

where I_t and I_{t+1} are successive video frames. If the test fails, then no stroke may be placed for the region M. Hence, the conditional in *paint* becomes: "**if** *refresh* **or** (*frameDiff* $> T_V$ **and** *areaError* $> T$)," where *frameDiff* is the sum of the differences over M. For interactive applications, we use a faster test that averages over an image region and compares this average to the previous frame's average:

$$\frac{1}{|M|} \left|\left| \sum_{(i,j)\in M} I_{t+1}(i,j) - \sum_{(i,j)\in M} I_t(i,j) \right|\right| > T_V$$

This test is computed in constant time using the summed-area table.

An example sequence is shown in Figure 2.

This difference masking method will not detect gradual changes over time, such as a fade-in or fade-out. We use a variation on this method for such input sequences, *cumulative difference masking*. A running cumulative sum of the region image difference is kept for each region M at each scale. The cumulative sum is updated at each frame. When a cumulative sum exceeds T_V, it is reset to zero and a stroke is painted.

[1]This method was demonstrated in the video portion of [7]. This method also appears to be employed by Studio Artist [17].

3.2 Frame Rate

Video frame rate is an important factor in producing painterly animation. In 30 Hz video, even minor flickering can become highly objectionable, and the paint-over method typically produces severe flickering in noisy or moving image regions. A more fundamental issue is that 30 Hz video can look "too real:" the underlying motion and shape is integrated so well by the human visual system that the video begins to take on the character of ordinary video with bad artifacts, rather than of a moving painting.

We find that 10-15 frames per second works well: enough of the underlying motion is preserved without too much realism, and flickering is much less objectionable. These frame rates lend a stronger subjective impression of being hand-painted, especially at 10 Hz, and especially for animations with very little motion. It is not entirely obvious why; perhaps we can mentally interpolate the human action of placing each stroke between frames, or perhaps we are used to seeing hand-made animations at lower frame rates (animation shot on "twos" is typically 12 fps or 15 fps). However, reducing the frame rate is not always an option, especially for live-action footage where a change in frame rate significantly alters the character of the movie.

The choice of frame rate is part of a larger tradeoff between abstraction and fidelity to the input. It appears that a similar frame rate was also used for "Impressions of San Francisco" [10]. In *What Dreams May Come,* [12] these problems were overcome by stylized motion and hand-edited motion vectors.

3.3 Optical Flow

In the above paint-over algorithm, brush strokes stay fixed on the image plane, even when the underlying object is moving. This gives the unusual, stylized effect of a continually-smudged image plane, and is sometimes undesirable. Alternatively, we can move the brush strokes with the objects, as suggested by Meier [14]. Litwinowicz [11, 12] proposed using optical flow to move brush strokes along the directions of scene motion. We use this idea by warping the image and painting-over. (Other features of Litwinowicz' method are more difficult to adapt.) Warping strokes also reduces flickering, since the image being painted over should be a closer match to the new video frame.

Optical flow is a measurement of object movement in a video sequence; it is defined as the projection of world motion vectors onto the image plane. Given an object that projects to a pixel (x, y) in frame t of a video sequence, let (x', y') be the projection of that same pixel in frame $t + 1$. If the object is visible in each frame, then the optical flow for that pixel is given by $F(x, y, t) = (x' - x, y' - y)$. We use Simoncelli et al.'s [16] probabilistic variant of coarse-to-fine differential estimation to compute optical flow.

The flow-based painting algorithm is as follows:

```
Paint the first frame
For each successive frame:
    Warp previous source frame to current for difference mask
    Warp all brush stroke control points to the current frame
    Paint the current frame over the warped painting
```

As with previous methods, we warp control points, not the bitmap. Warping the image can distort it in unpaintlike ways — given an ideal flow field, it would give the look of texture-mapping the painting onto the world.

Since we warp strokes by their control points and rerender, we must store all brush strokes in memory. Consequently, many brush strokes build up over time. We periodically cull strokes that are completely hidden, by rendering every stroke with a unique color and determining which colors are not shown.

Processing a sequence with optical flow produces a noticeably different effect than without. Instead of being continually repainted each frame, regions of the image move and shift to follow the surfaces in the scene. In our experiments, the look is of a wet, viscous canvas where brush strokes warp to match the video. The appearance is in large part due to the quality of the computed flow field — the flow algorithm does not detect discontinuities in the flow, producing a smooth flow field.

This painting style is somewhat similar to experimental animation with wet paint on glass (e.g. the work of Caroline Leaf), pinscreen, or sand animation. However, these techniques are limited by the media in their use of color, and are usually monochromatic.

Using a flow field other than the computed optical flow produces interesting effects. For example, processing video or even a still image with a nonzero constant flow field gives a sense of motion in a still world (Figure 5). Painting with user-defined flow fields is an intriguing avenue for artistic exploration. Note that the paint-over algorithm without optical flow is a special case of the optical flow algorithm, with $F(x, y, t) \equiv (0, 0)$.

4 Experiments

4.1 Music Video

We applied our techniques to footage of a jazz recording session. We broke the input footage into large segments, and processed each piece separately. Painting styles were chosen for visual appeal and to enhance the changing mood and intensity of the music. Although no specific formula was used, we found ourselves using some general design patterns: larger brush strokes (more abstract) were used during intense passages and meditative passages; smaller (less abstract) strokes were used in transitional passages; expressionistic styles (more active and abstract) were used mostly during the most intense passages and during solos. We processed almost all of the video at 15 frames per second, and did not use optical flow. The camera was moving at all times, so difference masking had relatively little effect. Cross-fades in the source video produced the effect of one moving shot being painted over another moving shot. When combined with a dramatic camera motion, a cross fade often appeard no longer as a fade, but as a smooth motion from one view to another. Excerpts are shown in Figure 4 and the accompanying video.

4.2 A “Living” Painting

We wanted to demonstrate our methods in an experience that would be immediately accessible to an outside visitor. We fed the output from a video camera into a PC, painted it, and projected it onto a large stretched canvas (Figure 1). The result appears on the canvas as a continually-updated painting that visitors can interact with and be a part of. The system runs on a 350 MHz Pentium II, with a Matrox video card and an Intergraph graphics board.

The system achieves a frame rate of 1-4 frames per second depending on the amount of motion in the scene; our system is not currently fast enough to include optical flow in the loop. However, at this frame rate, flickering cannot occur. We do not double buffer, allowing the visitor to watch the strokes appear on the canvas as they are created. Difference masking ensures that the painting only changes where there is motion in the scene. This is especially important for the more abstract painting styles in which the entire image would otherwise refresh each frame. Sample images are shown in Figure 6.

Our system was demonstrated at a recent exhibition. We found that users tended to spend a long time in front of the canvas creating various painterly renderings of their own faces and bodies. People also enjoyed watching other people create paintings. All participants seemed to immediately understand and accept the process.

5 Discussion and Future Work

We have presented new styles for painterly animation. The paint-over and difference masking methods give the subjective impression of a painting that is continually being painted over by a human hand. The use of estimated optical flow distorts the brush strokes to follow the scene motion. Procedural and hand-designed optical flow fields can be used to add motion and life to video.

These methods are designed to be simple and fast. High-end animation will require more sophisticated methods, as well as better artistic control over the animation.

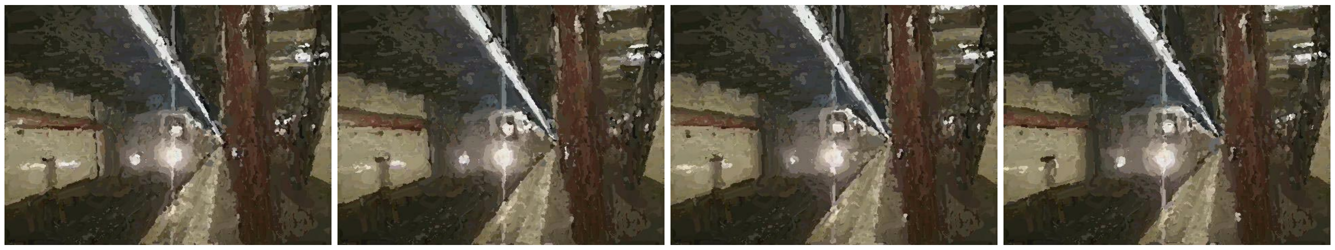

Figure 2: Consecutive frames of a video using paint-over and difference masking

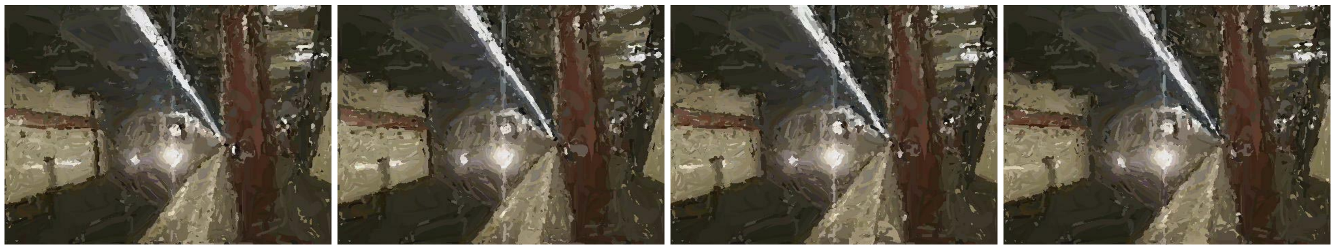

Figure 3: Consecutive frames from a painting using optical flow, paint-over and difference masking.

Acknowledgments

We are grateful for the contributions of the following individuals: Clilly Castiglia and Jay Konopka for video and systems support, Eero Simoncelli for optical flow code, and Erik Friedlander and Mike Landy of SIAM Records for music video footage. The first author is supported by NSF grant DGE-9454173.

References

[1] Franklin C. Crow. Summed-area Tables for Texture Mapping. *Computer Graphics (Proceedings of SIGGRAPH 84)*, 18(3):207–212, July 1984.

[2] Cassidy Curtis. Loose and Sketchy Animation. In *SIGGRAPH 98: Conference Abstracts and Applications*, page 317, 1998.

[3] Cassidy Curtis, Amy Gooch, Bruce Gooch, Stuart Green, Aaron Hertzmann, Peter Litwinowicz, David Salesin, and Simon Schofield. *Non-Photorealistic Rendering*. SIGGRAPH 99 Course Notes, 1999.

[4] Eric Daniels. Deep Canvas in Disney's Tarzan. In *SIGGRAPH 99: Conference Abstracts and Applications*, page 200, 1999.

[5] Bruce Gooch, Peter-Pike J. Sloan, Amy Gooch, Peter Shirley, and Richard Riesenfeld. Interactive Technical Illustration. In *Proc. 1999 ACM Symposium on Interactive 3D Graphics*, April 1999.

[6] Paul E. Haeberli. Paint By Numbers: Abstract Image Representations. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 207–214, August 1990.

[7] Aaron Hertzmann. Painterly Rendering with Curved Brush Strokes of Multiple Sizes. In *SIGGRAPH 98 Conference Proceedings*, pages 453–460, July 1998.

[8] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A Sketching Interface for 3D Freeform Design. *Proceedings of SIGGRAPH 99*, pages 409–416, August 1999.

[9] Michael A. Kowalski, Lee Markosian, J. D. Northrup, Lubomir Bourdev, Ronen Barzel, Loring S. Holden, and John Hughes. Art-Based Rendering of Fur, Grass, and Trees. *Proceedings of SIGGRAPH 99*, pages 433–438, August 1999.

[10] Peter Litwinowicz. Impressions of San Francisco. In *Electronic Theater Program*, number 120 in SIGGRAPH Video Review, 1997.

[11] Peter Litwinowicz. Processing Images and Video for an Impressionist Effect. In *SIGGRAPH 97 Conference Proceedings*, pages 407–414, August 1997.

[12] Peter Litwinowicz. Image-Based Rendering and Non-Photorealistic Rendering. In Stuart Green, editor, *Non-Photorealistic Rendering*, SIGGRAPH Course Notes. 1999.

[13] Lee Markosian, Michael A. Kowalski, Samuel J. Trychin, Lubomir D. Bourdev, Daniel Goldstein, and John F. Hughes. Real-Time Nonphotorealistic Rendering. In *SIGGRAPH 97 Conference Proceedings*, pages 415–420, August 1997.

[14] Barbara J. Meier. Painterly Rendering for Animation. In *SIGGRAPH 96 Conference Proceedings*, pages 477–484, August 1996.

[15] Ramesh Raskar and Michael Cohen. Image Precision Silhouette Edges. *1999 ACM Symposium on Interactive 3D Graphics*, pages 135–140, April 1999.

[16] Eero P Simoncelli, Edward H Adelson, and David J Heeger. Probability Distributions of Optical Flow. In *Proc. IEEE Conference of Computer Vision and Pattern Recognition*, June 1991.

[17] Synthetik Software. Studio Artist 1.1. Software package.

[18] Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. SKETCH: An Interface for Sketching 3D Scenes. In *SIGGRAPH 96 Conference Proceedings*, pages 163–170, August 1996.

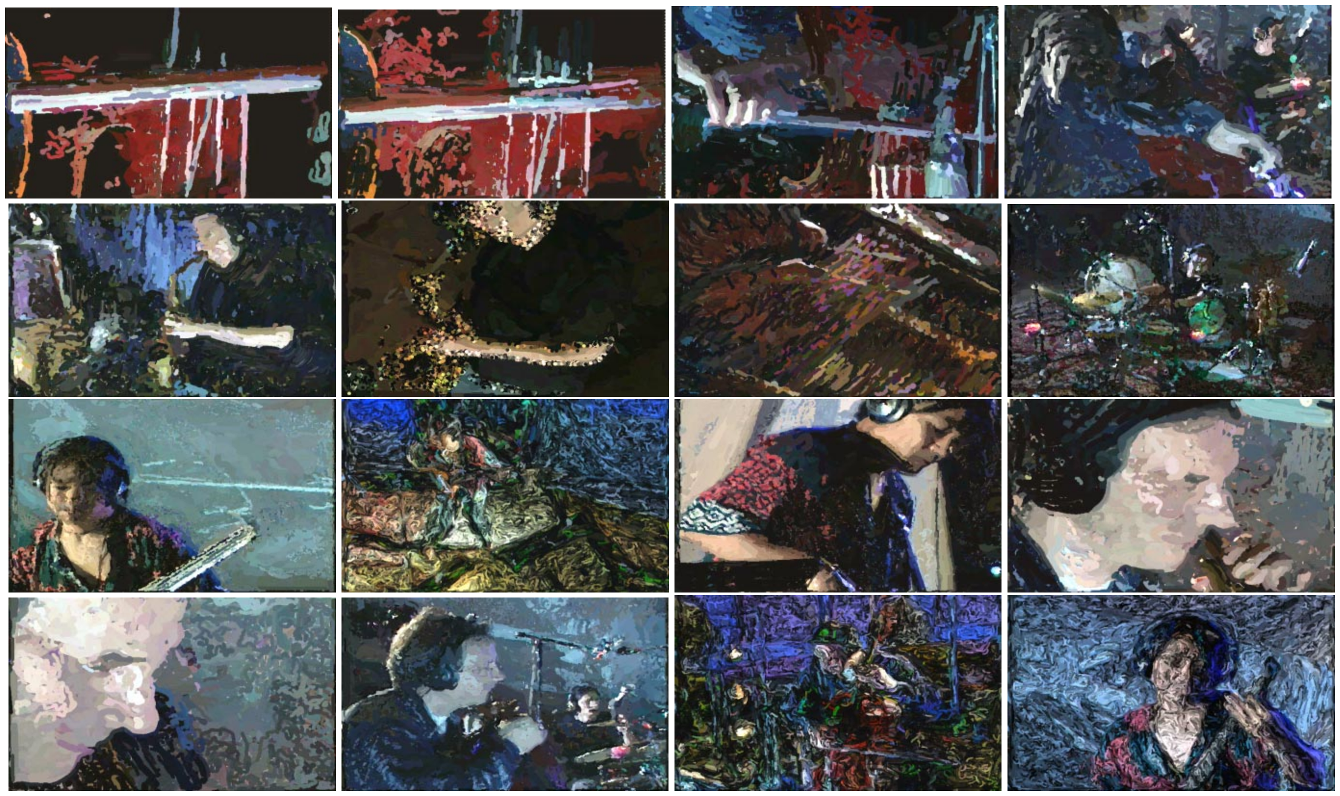

Figure 4: Frames from a music video, illustrating various painting styles and resulting effects.

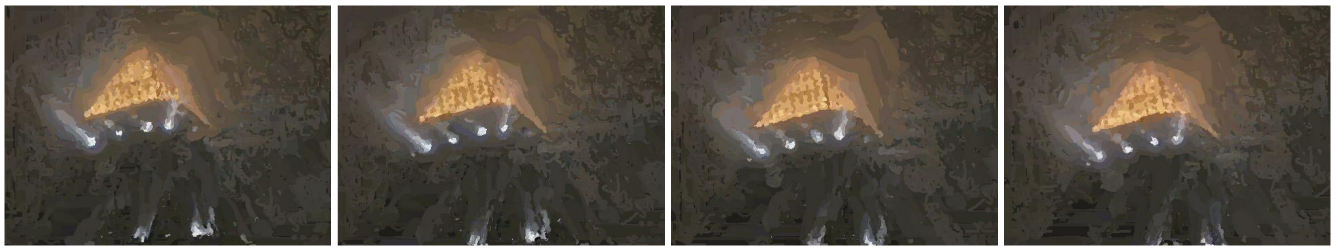

Figure 5: Consecutive frames of a video using a constant optical flow field ($F(x, y, t) \equiv (20 \text{ pixels}, 20 \text{ pixels})$). The video shows the top of a building on a foggy night, with camera motion. The flow field causes brush strokes in empty areas to rise toward the upper right corner of the image.

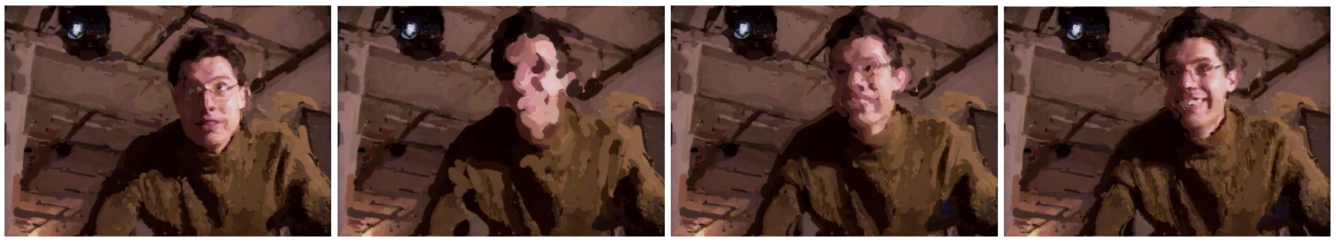

Figure 6: Paint layers from the live painting shown in Figure 1.