Paint By Relaxation

Aaron Hertzmann
New York University

Abstract

We use relaxation to produce painted imagery from images and video. An energy function is first specified; we then search for a painting with minimal energy. The appeal of this strategy is that, ideally, we need only specify what we want*, not how to directly compute it. Because the energy function is very difficult to optimize, we use a relaxation algorithm combined with search heuristics.*

This formulation allows us to specify painting style by varying the relative weights of energy terms. The basic energy function yields an economical style that conveys an image with few strokes. This style produces greater temporal coherence for video than previous techniques. The system allows as fine user control as desired: the user may interactively change the painting style, specify variations of style over an image, and/or add specific strokes to the painting.

1. Introduction

The time is ripe for novel computer-generated visual styles. The increasingly adventurous use of digital effects in movies and television, together with many recent advances in non-photorealistic rendering (NPR) make this an exciting time for NPR research. Painterly rendering, the subject of this paper, promises to merge the beauty and expressiveness of painting with the flexibility of computer graphics.

In this paper, we formulate painting as an energy minimization problem, given a source image or video sequence. This approach has many benefits. Most generally, it allows us to express the desired features of the painting style as energy terms, freeing us from the often-difficult task of devising a new or modified painting algorithm for each new consideration in a style.

This gives us the ability to produce paintings that closely match the input imagery. This is particularly important in processing video sequences, where imprecision can lead to unacceptable problems with temporal coherence; the only previous algorithm for automatically painting video with long, curved strokes [9] requires many small strokes, and still produces a degree of flickering that may be objectionable for some applications. We have attempted to recreate a somewhat "generic," modern painting style: realistic, but with visible brush strokes, inspired by artists such as Richard Diebenkorn, John Singer Sargent, and Lucian Freud.

Figure 1. A source image and painting

The energy minimization approach also allows us to bridge the gap between automatic painterly filters and user-driven paint systems. Users may specify requirements about a painting at a high-level (painting style), middle-level (variations of style over the image), and low-level (specific brush strokes); the user may work back and forth between these different levels of abstraction as desired. This feature is of critical importance for taking the best advantage of the skills of artists. This also allows us to accommodate a wide range of user requirements and skill levels, from desktop publishing to feature film production.

Color images and additional details from this work may be found in [8] (see http://www.mrl.nyu.edu/~hertzman).

1.1. Related Work

We now review previous work on painterly rendering. (A more general survey of NPR can be found in [4].) One thread of work are painterly filters, which

do not require an explicit 3D model of the world being painted. Haeberli [6] demonstrated an interactive painting system for quickly producing a painted representation of a still image. Several commercial packages (e.g. [1]) provide automatic painterly image filters based on these ideas, using fixed size and shape strokes. Shiraishi and Yamaguchi [16] use local image measurements to choose stroke size and orientation. Litwinowicz [12] demonstrated how these ideas could be extended for processing video, by moving short strokes with estimated optical flow. In previous work, we varied brush stroke sizes by layering and placed curved strokes by following normals of image gradients [7]. This method produces images that are loose but not very economical or precise; fine details are often lost even after several layers of painting. This method is extended to video processing in [9]. Although the existing techniques produce excellent imagery in a variety of styles, the range of styles available is limited by the particular painting process; this paper expands the range of styles in several dimensions, and provides a framework for the straightforward addition of further styles.

The availability of 3D geometry allows more flexibility in producing painterly animation. Meier [14] demonstrated an automatic particle-based approach for painting with short brush strokes. Extensions of this idea with interactively-placed strokes were used with great success in the recent feature films *What Dreams May Come* [13] and *Tarzan* [5]. Such systems naturally provide excellent temporal coherence and an economical painting style, if desired. However, exact 3D information is often unavailable. Klein et al. [11] renders objects with filtered textures. Also, the use of 3D typically yields a different aesthetic from painterly filters, because strokes that adhere closely to 3D shape over time usually give the impression of strokes attached to objects in space, rather than of a view through an animated painting. Which appearance is preferred will depend on the application.

Haeberli [6] introduced the use of relaxation for painterly rendering, and Turk and Banks [18] used relaxation to illustrate vector fields with streamlines. This paper extends their work.

2. Energy Function

The central idea of this paper is to formulate painting as an energy relaxation problem. Given an energy function $E(P)$, we seek the painting P^* with the least energy:

$$P^* = \arg\min_{P \in \mathcal{P}} E(P)$$

where P refers to a specific painting and $\mathcal{P}$ to the space of all paintings. In this paper, a painting is defined as an ordered collection of colored brush strokes, together with a fixed canvas color or texture. A brush stroke is a thick curve defined by a list of control points, a color, and a thickness/brush width. A painting is rendered by compositing the brush strokes in order onto the canvas. Brush strokes are rendered as thick cubic B-splines. Strokes are drawn in the order that they were created.

An energy function can be created as a combination of different functions, allowing us to express a variety of desires about the painting, and their relative importance. Each energy function corresponds to a different painting style. This formulation gives us a reasonably intuitive way of designing painting styles, in that we specify the desired features of the painting, rather than providing a direct method for computing the painting, as has been done in the past.

At a high level, our goal has been to seek concise paintings that match a source image closely and cover the image with paint, but use as few strokes as possible. To this end, we use the following energy function for a painting:

$$\begin{aligned}
E(P) &= E_{app}(P) + E_{area}(P) + \\
&\quad E_{nstr}(P) + E_{cov}(P) \\
E_{app}(P) &= \sum_{(x,y)\in\mathcal{I}} w_{app}(x,y)||P(x,y) - G(x,y)|| \\
E_{area}(P) &= w_{area} \sum_{S\in P} \text{Area}(S) \\
E_{nstr}(P) &= w_{nstr} \cdot (\text{number of strokes in } P) \\
E_{cov}(P) &= w_{cov} \cdot (\text{number of empty pixels in } P)
\end{aligned}$$

This energy is a linear combination of four terms. The first term, E_{app}, measures the pixelwise differences between the painting and a source image G. Each painting in this paper was created with respect to a source image. The second term, E_{area}, measures the sum of the surface areas of all of the brush strokes in the painting. In some sense, this is a measure of the total amount of paint that was used by the artist in making the image. The number of strokes term E_{nstr} can be used to bias the painting towards larger brush strokes. The coverage term E_{cov} can be used to force the canvas to be filled with paint, if desired, by setting w_{cov} to be very large. The weights w are user-defined values. The color distance $||\cdot||$ represents Euclidean distance in RGB space.

The first two terms of the energy function quantify the trade-off between two competing desires: the desire to closely match the appearance of the source image, and the desire to use as little paint as possible, i.e. to be economical. By adjusting the relative proportion of

Figure 2. *Top*: **Painterly renderings with the method of [7] with 1 and 2, layers, respectively. The algorithm has difficulty capturing detail below the stroke size. (The source is shown in Figure 1.)** *Bottom*: **Paint by relaxation, with 1 and 2 layers. Strokes are precisely aligned to image features, especially near sharp contours, such as the lower-right corner of the jacket.**

w_{app} and w_{area}, a user can specify the relative importance of these two desires, and thus produce different painting styles. Likewise, adjusting w_{nstr} allows us to discourage the use of smaller strokes.

By default, the value of $w_{app}(x, y)$ is initialized by a binary edge image, computed with a Sobel filter. This gives extra emphasis to the edges, although a constant weight often gives decent results as well. If we allow the weight to vary over the canvas, then we get an effect that is like having different energy functions in different parts of the image (Figure 3). The weight image $w_{app}(x, y)$ allows us to specify how much detail is required in each region of the image, and can be generated automatically, or hand-painted by a user (Section 7).

The particular form of these equations has no intrinsic meaning (that we are aware of); they are simply one possible formulation of the desires expressed above. When the user chooses a set of weights (usually by experimentation), the complete energy function encapsulates some aesthetic sensibility of the user.

3. Relaxation

The energy function presented in the previous section is very difficult to optimize: it is very discontinuous, it has a very high dimensionality and there does not appear to be an analytic solution.

Following Haeberli [6] and Turk and Banks [18], we use a relaxation algorithm. This is a trial-and-error approach, illustrated by the following pseudocode:

$P \leftarrow$ empty painting

while not *done*
  $C \leftarrow$ SUGGEST() *// Suggest change*
  if $(E(C(P)) < E(P))$ *// Does the change help?*
    $P \leftarrow C(P)$ *// If so, adopt it*

There is no guarantee that the algorithm will converge to a result that is globally optimal or even locally optimal. Nevertheless, this method is still useful for producing pleasing results.

The main question of interest is how make the suggestions. In our initial experiments, we used highly random suggestions, such as adding a disc stroke in a random location with a random size. Most of the computation time was spent on suggestions that were rejected, resulting in paintings poorly-matched to the energy. Because the space of paintings has very high dimensionality, it is possible to make many suggestions that do not substantially reduce the energy. Hence, it is necessary to devise more sophisticated ways of generating suggestions.

4. Stroke Relaxation

A basic strategy that we use for generating suggestions is to use a *stroke relaxation* procedure. This is an adaptation of snakes [10, 2] to the painting energy function: it searches for an approximate local minimum for a given stroke placement.

The high-level algorithm for modifying a stroke is:

1. Measure the painting energy
2. Select an existing stroke
3. Optionally, modify some stroke attribute
4. Relax the stroke
5. Measure the new painting energy

This algorithm is described in detail in Section 8.1. However, we mention two important aspects of the implementation here: First, the stroke search procedure

(a) (b) (c) (d)

Figure 3. Spatially-varying style. (a) Source image. (b) Interactively-painted weight image (w_{app}). (c) Painting with the given weights. More detail appears near faces and hands. (d) Another choice of weights; detail is concentrated on the rightmost figures.

evaluates the painting energy with the stroke deleted (for reasons described later). Hence, the modification suggestion may become a stroke deletion suggestion, at no extra cost. Second, the relaxation procedure would be very expensive and difficult to implement if done precisely. Instead, we use an approximation for the energy inside the relaxation loop, and measure the exact energy only after generating the suggestion.

4.1. Single Relaxation Steps

In this Section, we describe individual procedures for generating suggestions.

Add Stroke: A new stroke is created, starting from a given point on the image. The new stroke is always added in front of all existing strokes. Control points are added to the stroke, using the contour-approximating procedure described in [9]. The stroke is then relaxed (see previous section). The result of this search is then suggested as a change to the painting.

Reactivate Stroke: A given stroke is relaxed, and the modification is suggested as a change to the painting. However, if deleting the stroke reduces the painting energy more than the modification does, then the stroke will be deleted instead. Note that this procedure does not modify the ordering of strokes in the image. Thus, it will delete any stroke that is entirely hidden, so long as there is some penalty for including strokes in the painting (i.e. $w_{area} > 0$ or $w_{nstr} > 0$).

Enlarge Stroke: If the stroke radius is below the maximum, the stroke radius is incremented and the stroke is reactivated. The resulting stroke becomes a suggestion.

Shrink Stroke: If the stroke radius is above the minimum radius, the stroke radius is decremented and the stroke is reactivated.

Recolor: The color for a stroke is set to the average of the source image colors over all of the visible pixels of the stroke, and the stroke is reactivated.

4.2. Combining Steps

Individual suggestions are combined into loops in order to guarantee that every stroke is visited by a relaxation step.

Place Layer: Loop over image, *Add Strokes* of a specified radius, with randomly perturbed starting points.

Reactivate All: Iterate over the strokes in order of placement, and *Reactivate* them.

Enlarge/Shrink All: For each stroke, *Enlarge* until the change is rejected or the stroke is deleted. If the stroke is unchanged, then *Shrink* the stroke until the change is rejected or the stroke deleted.

Recolor All: Iterate over the strokes, and *Recolor* each one.

Script: Execute a sequence of relaxation loops. To make a painting from scratch, we use the following script:

foreach brush size R_i, from largest to smallest,
do N times:
Reactivate all strokes
Place Layer R_i
Enlarge/Shrink All
Recolor All

We normally use $N = 2$. This script usually brings the painting sufficiently close to convergence. Note that the reactivation loops apply to all brush strokes, not just those of the current size. For processing video, we reduce processing time by setting $N = 1$ and omitting the recolor and enlarge steps.

Creating a final image can take several hours, depending on the image size and the painting style. Most of the images in this paper were generated in a few hours on a 225 MHz SGI Octane R10000. Video was processed at low resolution (320x240) with the reduced script above to save time, yielding a processing rate of about 1 hour per frame.

5. Stroke Color, Texture, and Opacity

Our system provides a variety of different ways to render an intermediate or final painting. Brush strokes can be rendered with random color and intensity perturbations (as in [7]), with transparency, and/or with procedural textures. Texture is currently omitted from the main relaxation procedure solely for the sake of efficiency. Furthermore, separating the rendering step allows us to experiment with different stroke colors and textures without needing to perform relaxation anew. Our procedural stroke texturing is described in [8].

6. Video

The relaxation framework extends naturally to processing video. A variety of energy formulations can be employed for video. In the simplest formulation, the energy of the video sequence is the sum of the energy for each frame. The target image for each video frame is the corresponding image in the source sequence. For efficiency and temporal coherence, we can use the painting for one frame as the initial painting for the next frame. For speed, we currently process each frame sequentially; a more general extension would be to perform relaxation over the entire sequence in arbitrary order.

This method can be improved if optical flow information is available [12, 9]. Optical flow is a measure of the motion of scene objects projected onto the image plane. Warping brush strokes by optical flow gives the impression of brush strokes moving to follow the motions of objects in the scene. We compute flow using a variant of coarse-to-fine differential motion estimation [17]. In order to generate the initial painting for a frame, we warp the stroke positions by the optical flow before relaxation. This gives a good initialization for the next frame.

This yields a relatively clear video sequence, with much less flickering than occurs in previous algorithms. In particular, it is possible to paint a video sequence with much larger strokes than before. However, the temporal coherence is far from perfect, and more work remains to be done in this area.

7. Interaction and Metapainting

The energy formulation is well-suited for an interactive painting application. In this view, the software functions as a high-level paintbox, allowing the user to make artistic decisions at any stage and at any level of abstraction in the painting process. In particular, the user may:

- Select overall painting styles.
- Select different styles for different image regions.
- Place individual brush strokes, as suggestions or by decree.

We refer to this approach as *metapainting*. (A related system was described by Salisbury et al. for pen-and-ink illustration [15].)

In our implementation, the user has complete freedom to make these choices before, during, and after relaxation. For example, the user might select an overall style for an image and perform a relaxation. After seeing the result, the user decides that a particular part of the image should be emphasized, paints a new appearance weight function, and then performs relaxation again. The user then realizes that she desires specific brush strokes in one image region, and then paints them in.

We believe that interactive metapainting can be a very powerful tool for image-making, combining the ease of an automatic filter with the fine control of a digital paint system. The artist holds absolute control over the appearance of the final painting, without being required to specify every detail. The appeal is even greater for video processing, where one can automatically propagate artistic decisions between video frames.

The system is currently too slow to provide tight interactivity. With faster hardware, the user feedback loop can be made tighter. We look forward to the day when it is possible to visualize changes in painting styles at interactive rates.

8. Implementation Details

In this section, we describe some of the algorithms and data structures used in our implementation to avoid redundant computation.

Lazy evaluation and partial result caching are used throughout our code. The values of each subterm in the energy function are cached, and are only updated when they change. For example, one image encodes the per-pixel value of the weighted color difference (WCD) between the painting and source image (i.e. the summand of E_{app}). Whenever a pixel value changes, the corresponding value in the WCD image is updated. The sum of the WCD image is also updated by subtracting the old value and adding the new value. Similar methods are used for other the energy terms.

Figure 4. Consecutive frames from a painterly animation rendered at low-resolution (320x240).

8.1. Stroke Relaxation

We now describe the details of the stroke relaxation procedure introduced in Section 4.

Performing a search for a locally optimal set of stroke control points would be prohibitively expensive even with dynamic programming; the cost is a product of the number of per-pixel operations, the cost of scan-converting a curve section, and an exponential in the size of the search neighborhood. A key idea of the relaxation procedure is to perform relaxation over a simpler energy function that approximates the true energy, and has roughly the same minima as the true energy. The true stroke energy will only be evaluated once, for the resulting suggestion.

Our goal is to replace initial stroke S with a stroke T, where T has the same color and thickness as S, and T minimizes the new energy $E(P-S+T)$. This is equivalent to minimizing the change in energy $E(P-S+T)-E(P-S)$, because $E(P-S)$ is constant with respect to T. This change can be evaluated faster than the energy.[1]

We build an approximate energy function $I(T) \approx E(P-S+T)-E(P-S)$ over which to search. In this new energy function, a brush stroke is modeled as the union of discs, each centered at a control point. This simpler shape will allow us to use a faster dynamic programming algorithm, because the shape of the approximation depends locally on only one control point, and a computation over a disc can easily be cached.

The search is restricted in order to ensure that adjacent control points maintain a roughly constant distance from each other. This is done by adding an extra term to the painting energy function: $E_{mem}(T) = w_{mem}\sum_{v_i \in T}(||v_i - v_{i+1}|| - R)^2$, for a stroke T of radius R with control points v_i. This is a variant of membrane energy [10]. This energy is included in the painting energy by summing it over every paint stroke.

[1] We use $+$ and $-$ in the set theoretic sense when applied to paintings and strokes, e.g. $P+T = P \cup \{T\}$ = painting P with stroke T added. $E(P)$ is a scalar, and thus $E(P_1)-E(P_2)$ is a scalar subtraction.

All of the terms of the true energy function $E(P)$ have corresponding terms in the approximation $I(T)$:

$$I(T) = I_{app}(T)+I_{area}(T)+I_{nstr}(T)+I_{cov}(T)+I_{mem}(T)$$

The number of strokes and spacing terms are measured exactly: $I_{str}(T) = w_{nstr}$, $I_{mem}(T) = E_{mem}(T)$. The area energy I_{area} is approximated as the product of the stroke's radius and its control polygon arc length.

The appearance (I_{app}) and coverage improvements (I_{empty}) are approximated by making use of auxiliary functions $I_R(x,y)$ and $I_p(x,y)$. This formulation allows us to use lazy evaluation to avoid redundant computation of the auxiliary functions. $I_p(x,y)$ measures the energy improvement due to changing pixel (x,y) to the color of the stroke, unless the stroke would be hidden by another stroke at that pixel. $I_R(x,y)$ sums $I_p(x,y)$ over a circular disc around (x,y). These functions are given by:

$$\begin{aligned} I_R(x,y) &= \sum_{||(u,v)-(x,y)|| \le R} I_p(u,v) \\ I_p(x,y) &= h(x,y)w_{app}(x,y)I_C(x,y) - c(x,y)w_{empty} \\ I_C(x,y) &= ||C - G(x,y)|| - \\ &\quad ||(P-S)(x,y) - G(x,y)|| \end{aligned}$$

C is the color of the brush stroke. $c(x,y)$ and $h(x,y)$ are indicator functions computed from the fragment buffer: $c(x,y)$ is 0 if (x,y) is covered by any paint, 1 otherwise; $h(x,y)$ is 0 if (x,y) is covered by a stroke that was created after S. (The new stroke will take the same place as the old stroke in the painting; e.g. if S is completely hidden, then T may be as well.) Summing $I_R(x,y)$ over the control points for a stroke gives an approximation to the appearance and coverage improvement terms due to placing the stroke:

$$I_{app}(T)+I_{empty}(T) = \sum_{(x,y) \in \text{control points of } T} I_R(x,y)$$

With these functions, we can modify a stroke as follows:

Measure $E(P)$
Select an existing stroke $S \in P$
Remove the stroke, and measure $E(P - S)$
Optionally, modify some attribute of the stroke
Relax the stroke:
    $S' \leftarrow S$
    do $lastImp \leftarrow I(S')$
        $S' \leftarrow \arg\min_{T \in neighborhood(S')} I(T)$
    while $I(S') < \min\{lastImp, H\}$
Measure the new painting energy $E(P + S' - S)$.
$P \leftarrow \arg\min_{Q \in \{P, P-S, P+S'-S\}} E(Q)$

The **do** loop above represents the actual stroke relaxation. We use a modified version of the dynamic programming algorithm described by Amini et al. [2]. This method creates dynamic programming tables containing optimal control point configurations for subproblems of strokes of length 2 to n; the minimum entry in the n-th table corresponds to the locally optimal stroke of length n. We modify this procedure to find the optimal stroke length as well as control point locations, by observing that the dynamic programming tables contain the minimum stroke energies for each stroke length. Thus, we can find the optimal number of control points simply by locating the minimum element among all tables corresponding to the allowed stroke lengths (as specified by the painting style). In addition, we extend the length of the stroke before each relaxation step, by adding a point to the end of the stroke, thus allowing the stroke to lengthen as well as to shrink. The algorithm is described in detail in [8]. The constant H is used to prevent pursuing strokes that appear to be unprofitable. We normally use $H = 0$; H could also be determined experimentally.

8.2. Fragment Buffer

When we delete a brush stroke, we need to find out what the new painting looks like. A simple (but exceedingly slow) method would be to rerender the revised painting from scratch. Because brush stroke deletion is a very common operation in the relaxation process, we need a faster way to delete brush strokes.

We choose an approach similar to an A-buffer [3], called a *fragment buffer*. Each pixel in the image is associated with a list of the fragments that cover that pixel, as well as their associated stroke indices. A fragment is defined as a single RGBA value, and each list is sorted by depth. In our current implementation, this is equivalent to sorting by the order of stroke creation.

To compute the color of a pixel, we composite its fragment list from bottom to top, unless the top fragment is opaque. Creating or deleting a stroke requires generating or deleting one fragment for each pixel covered by the stroke. In our experiments, the length of a typical fragment list rarely exceeds 5 for automatically-generated images.

The fragment buffer is also useful for other operations, such as picking a stroke from a mouse click.

There are a variety of alternatives to this method, based on saving partial rendering results; we chose this as a good trade-off between simplicity and efficiency.

8.3. Parallel Implementation

We have implemented a parallel version of the relaxation algorithm for better performance. A server system contains the main copy of the painting, and client machines provide suggestions to the server. Whenever a client generates a suggestion, it is sent to the server over the socket. These suggestions are kept in a queue on the server, and are tested in the order they were received. Whenever the server commits a change to the painting, it is announced over the socket, and adopted by all clients in their local copies of the painting.

This system is limited by the main processor's speed in processing suggestions. We have also devised (but not implemented) a decentralized version in which any processor may commit changes to a painting after acquiring a lock on the appropriate image region.

9. Discussion

We have presented a novel method for generating painted imagery from images and video. This is done by searching for a painting that minimizes some energy function. The style of the painting is completely specified by the energy function, constraints on stroke sizes, the brush textures, and, to some extent, the relaxation steps. This allows us to produce an economical painting style, and to easily accommodate user-specified constraints or spatial variations in style. In the long run, it provides a general framework for designing painting styles in terms of energy functions.

The relaxation algorithm requires substantially more computation time than do previous algorithms, but can produce images in novel styles. The distinction is analogous to the situation in photorealistic graphics, with a continuum from fast, low-end methods to more expensive high-end methods. Painting can be done at different points along the continuum: relaxation can be combined with another painting algorithm, and computation time can be traded off for more polished results. As in photorealistic graphics, we expect that the high-end painterly rendering of today can be a part of the consumer-level graphics of tomorrow.

10. Future Work

The most pressing need is for speed; we were unable to test many of the styles that we wanted, because they would have been too slow to compute. We are hopeful that faster hardware in the next few years will go a long way to speeding up the computation. The suggestion mechanism could also be tuned.

Given faster computation, we can explore standard methods for finding better minima, including annealing, Monte Carlo methods, and genetic algorithms, any of which would be straightforward to incorporate into relaxation. Likewise, stroke relaxation could be improved by using a better approximation, and including stroke color, radius, and texture in the search.

The temporal coherence of painted video needs improvement. We experimented with an energy term that penalizes the difference between painted frame t_i and the painted frame t_{i-1} warped to t_i. This produced very poor results, because the brush strokes are placed in frame t_{i-1} without regard to the scene geometry. For example, a stroke that is placed across the silhouette of a white surface against a white background will not work if a change of viewpoint reveals a red surface at the silhouette. Global optimization over an entire sequence may give better results.

A more sophisticated user interface would be helpful, especially one that provides diagnostic tools for estimating energy functions from examples or from local information.

From an artistic standpoint, the most interesting future work is the exploration of the energy functions and painting styles. Thus far, we have only scratched the surface of painting and animation styles that can be created by energy minimization.

11. Acknowledgments

We are grateful to Ken Perlin and Denis Zorin for fruitful discussions and feedback, to Chris Bregler for pointing out the work of Amini et al., to Robert Buff and Will Kenton for help with capturing input images and video, and to Philip Greenspun (http://photo.net/philg) for Figure 3(a). This work was supported in part by NSF Grant DGE-9454173.

References

[1] Adobe Systems. Adobe Photoshop. Software package.

[2] A. A. Amini, T. E. Weymouth, and R. C. Jain. Using Dynamic Programming for Solving Variational Problems in Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):855–867, Sept. 1990.

[3] L. Carpenter. The A-buffer, an Antialiased Hidden Surface Method. *Computer Graphics (SIGGRAPH '84 Proceedings)*, 18(3):103–108, July 1984.

[4] C. Curtis, A. Gooch, B. Gooch, S. Green, A. Hertzmann, P. Litwinowicz, D. Salesin, and S. Schofield. *Non-Photorealistic Rendering*. SIGGRAPH 99 Course Notes, 1999.

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

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

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

[8] A. Hertzmann. Paint by Relaxation. Technical Report TR2000-801, NYU Computer Science, May 2000.

[9] A. Hertzmann and K. Perlin. Painterly Rendering for Video and Interaction. In *Proceedings of the First Annual Symposium on Non-Photorealistic Animation and Rendering*, June 2000.

[10] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active Contour Models. *International Journal of Computer Vision*, 1(4), 1987.

[11] A. W. Klein, W. W. Li, M. M. Kazhdan, W. T. Correa, A. Finkelstein, and T. A. Funkhouser. Non-photorealistic virtual environments. *Proceedings of SIGGRAPH 2000*, July 2000.

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

[13] P. Litwinowicz. Image-Based Rendering and Non-Photorealistic Rendering. In S. Green, editor, *Non-Photorealistic Rendering*, SIGGRAPH Course Notes. 1999.

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

[15] M. P. Salisbury, M. T. Wong, J. F. Hughes, and D. H. Salesin. Orientable Textures for Image-Based Pen-and-Ink Illustration. In *SIGGRAPH 97 Conference Proceedings*, pages 401–406, Aug. 1997.

[16] M. Shiraishi and Y. Yamaguchi. An algorithm for automatic painterly rendering based on local source image approximation. *NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering*, pages 53–58, June 2000.

[17] E. P. Simoncelli, E. H. Adelson, and D. J. Heeger. Probability Distributions of Optical Flow. In *Proc. IEEE Conference of Computer Vision and Pattern Recognition*, June 1991.

[18] G. Turk and D. Banks. Image-Guided Streamline Placement. In *SIGGRAPH 96 Conference Proceedings*, pages 453–460, Aug. 1996.