# Paint by Relaxation

**Aaron Hertzmann**
Media Research Laboratory
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
719 Broadway, 12th Floor
New York, NY 10003
hertzman@mrl.nyu.edu
http://www.mrl.nyu.edu/hertzmann
NYU Computer Science Technical Report

May 1, 2000

**Abstract**

We use relaxation to produce painted imagery from images and video. An energy function is first specified; a painting is then generated by performing a 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 various 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 painting that effectively conveys an image with few strokes. This economical style produces moderate temporal coherence when processing video, without losing the essential 2D quality of the painting. 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. Procedural stroke textures may be used to enhance visual appeal.

## 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. In some sense, this paper presents the first stroke-based painterly image filter that can produce non-“impressionistic” paintings[1]. 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 [HP00] requires many small strokes, and still produces a degree of flickering that may be objectionable for some applications.

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, skill levels, and time constraints, from desktop publishing to feature film production.

### 1.1 Related Work

We now review previous work on painterly rendering. (A more general survey of NPR can be found in [CGG$^+$99].) One thread of work are painterly filters, which do not require an explicit 3D model of the world being painted. Haeberli [Hae90] demonstrated an interactive painting system for quickly producing painted representations of still image. Several commercial packages (e.g. [Ado, Rig99]) provide automatic painterly image filters based on these ideas, using fixed size and shape strokes. Litwinowicz [Lit97] 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 [Her98]. 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 [HP00]. 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 [Mei96] 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* [Lit99] and *Tarzan* [Dan99]. Such systems naturally provide excellent temporal coherence and an economical painting style, if desired. However, exact 3D information is often unavailable. 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 [Hae90] introduced the use of relaxation for painterly rendering, and Turk and Banks [TB96] used relaxation to illustrate vector fields with streamlines. This paper extends this work. Optimization meth-

[1] By “impressionistic,” we refer to a painting composed of a scattershot collection of brush strokes rather than to a specific historical painting style. We have attempted to recreate a somewhat “generic,” modern painting style: realistic, but with visible brush strokes, inspired by artists as Richard Diebenkorn, Wayne Thiebaud, and Lucien Freud.

ods have been used in other areas of computer graphics that entail both aesthetic and mathematical concerns, such as animation [GSG+99], surface modeling (e.g. [WW92]), and lighting design (e.g. [KPC93]).

Two basic approaches have been used to create realistic-looking stroke textures. One general approach is to use some approximate simulation of physical media, by simulation of bristles [Str86] and/or the fluid dynamics of paint [Sma90, CAS+97]. Such methods provide impressively-realistic appearance at the cost of much longer computation time. Alternatively, procedural textures may be used [Cur99]; procedural textures are more ad hoc, but faster to evaluate and easier to control. However, the choice of texturing strategy is usually independent of the stroke placement algorithm.

### 1.2 Overview

The rest of this paper is organized as follows: We first describe the energy of a painting in Section 2. The relaxation procedure for this energy is described in Section 3, and the sub-procedure for applying relaxation to a single stroke is described in Section 4. We then describe stroke texturing in Section 5, painterly video processing in Section 6, and our user interface in Section 7. Some important implementation details are given in Section 8, followed by discussion in Section 10, and future work in Section 11.

## 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:

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

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.[2]

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.

We use the following energy function for a painting:

$$\begin{aligned} E(P) &= E_{app}(P) + E_{area}(P) + E_{nstr}(P) + E_{cov}(P) \\ E_{app}(P) &= \sum_{(x,y) \in \mathcal{I}} w_{app}(x,y) ||P(x,y) - G(x,y)|| \end{aligned}$$

[2]We also experimented with ordering by brush radius, and found that, due to the nature of the relaxation, large strokes were never placed after small strokes in any ordering. This occurs because the smaller strokes are almost always a better approximation to the source image than is a large stroke; hence, adding a large stroke on top of many small strokes almost always increases the painting energy in the short term.

$$\begin{aligned} E_{area}(P) &= w_{area} \sum_{S \in P} \text{Area}(S) \\ E_{nstr}(P) &= w_{nstr} \cdot (\text{the number of strokes in } P) \\ E_{cov}(P) &= w_{cov} \cdot (\text{the number of empty pixels in } P) \end{aligned}$$

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 will be created with respect to a source image; this is the image that we are painting a picture of. 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 $w_{app}$ 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 quality, 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 8). 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).

Some important visual constraints are difficult to write mathematically. For example, the desire that the painting appear to be composed of brush strokes would be difficult to write as an energy function over the space of bitmaps; hence, we parameterize a painting in terms of strokes. Hence, we define some aspects of a painting style as parameters to the relaxation procedure, rather than as weights in the energy function.

## 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 (number of brush strokes * number of brush stroke parameters), and there does not appear to be an analytic solution.

Following Turk and Banks [TB96], 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 interesting question 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 a variation on snakes [KWT87], adapted to the painting energy function. This method refines an approximate stroke placement choice to find a local maximum for a given stroke. This is done by an exhaustive local search to find the best control points for the stroke, while holding fixed the stroke's radius and color, and the appearance of the rest of the painting. We also modify the snake algorithm to include the number of stroke control points as a search variable.

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

Here we omit many implementation details. 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 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 [HP00]. 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. a positive weight on the area or number of strokes terms).

**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 over the brush strokes 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. The accompanying 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 [Her98]), with transparency, and/or with procedural textures. Texturing operations are omitted from the main relaxation procedure primarily 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. Procedural stroke texturing is described in the Appendix.

All of our stroke scan-conversion is performed in software, using the triangle-strip method described in the Appendix. We have not used hardware scan-conversion because we need to be able to reliably traverse every pixel in a brush stroke without rendering (for adding/removing strokes and summing image statistics), which is not easily afforded by hardware scan-conversion. Scan-converting in software allows us to guarantee that a loop over a stroke visits every pixel of the stroke exactly once.

## 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 [Lit97]. 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 [SAH91]. 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 more "impressionistic" 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 [SWHS97].)

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.

Our user interface allows several different methods for visualizing the painting and editing painting styles, including output painting, difference image, source image, weight image, pseudocolor (each stroke assigned a random color), etc. The user may paint directly into the output image, and may paint weights into the weight image. Painting styles and parameters are controlled by various widgets. Specific strokes may be reactivated by clicking on them. During relaxation, changes to the painting are displayed as they are computed, to give the user a view of the work in progress and to allow modification to the style during the computation.

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.

## 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

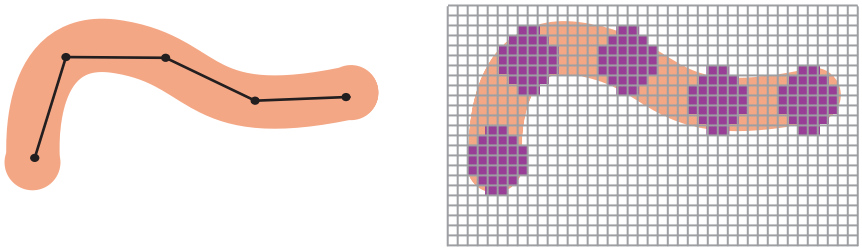


Figure 1: Approximating the improvement of a stroke. *Left:* A stroke is defined as a smooth path approximating a given list of control points. *Right:* The improvement of a stroke is approximated by the sum of the improvements of discs centered at each control point. Each disc has radius $R$. The improvement of placing a disc of radius $R$ at location $(x, y)$ is stored as an image $I_R(x, y)$. (Although not illustrated in the figure, control points are roughly $R$ pixels apart from each other.) The image $I_p$ stores improvement to the painting due to each pixel. $I_R(x, y)$ is computed by summing $I_p$ over a disc of radius $R$. The images $I_R$ and $I_p$ are computed lazily, in order to avoid redundant computation.

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 $E(P-S+T)-E(P-S)$, which can be evaluated faster.[3] 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 roughly as the union of discs centered at each control point. 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: $E_{mem}(P) = w_{mem} \sum(||v_i - v_{i+1}|| - R)^2$, for a stroke of radius $R$. This is a variant of membrane energy [KWT87].

All of the terms of the true energy function $E(P)$ have corresponding terms in the approximation $I(T)$. 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 $I_p(x, y)$ (Figure 1). This formulation allows us to use lazy evaluation to avoid unnecessary extra 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:

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

[3] 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.

$C$ 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:

$$I_{app} + I_{empty} = \sum_{(x,y) \in \text{control points}} I_R(x, y) \tag{1}$$

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** $\mathit{lastImp} \leftarrow I(S')$
        $S' \leftarrow \arg\min_{T \in neighborhood(S')} I(T)$
    **while** $I(S') < \min\{\mathit{lastImp}, H\}$
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 in the above text represents the actual stroke relaxation, described in the next section.

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.

### 8.1.1 Locally Optimal Stroke Search

In this section, we describe a dynamic programming (DP) algorithm for searching for a locally optimal stroke, with a fixed stroke color and brush radius. This is a slightly modified version of an algorithm developed by Amini et al. [AWJ90] for use in finding image contours, based on snakes [KWT87].

We search for the locally optimal control polygon, and fix all other stroke and painting parameters. We first define an energy function $e(T)$ of a stroke $T$ consisting of 2D control points $v_0, v_1, ..., v_{n-1}$:

$$e(T) = \sum_{i=0}^{n-1} e_0(v_i) + \sum_{i=0}^{n-2} e_1(v_i, v_{i+1}) \tag{2}$$

For our purposes, the energy $e(T)$ is the improvement $I(P, T, S)$ defined in the previous section. Note that $n$ is also variable; we are searching for the optimal control polygon of any length. For now we will assume that the stroke length is kept constant, and later describe how to modify it. Higher order terms, such as bending energy, may also be added to optimization (e.g. $\sum_{i=0}^{n-3} e_1(v_i, v_{i+1}, v_{i+2})$) as described by Amini et al. [AWJ90]; however, the complexity is exponential in the order of the energy, so we have focused only on the lower-order terms.

The main loop of the search is as follows: Given a current stroke, we want to find new control points in the neighborhood of the current control points (Figure 2). Each control point has a window of $w \times w$ candidates in which we will search for the best nearby control point. We normally use $w = 5$. The intuition for the algorithm comes from the observation that the energy is purely local: the optimal location for the first control point is connected only to the optimal location for the second control point; the rest of the curve has no interaction with the first control point. Therefore, we can create a DP table that will take the position of the second control point and return the corresponding optimal location for the first point.

We start by builing the DP tables as:

$$\begin{aligned}
s_0(v_1) &= \min_{v_0} e_0(v_0) + e_0(v_1) + e_1(v_0, v_1) \\
s_1(v_2) &= \min_{v_1} s_0(v_1) + e_0(v_2) + e_1(v_1, v_2) \\
&\vdots \\
s_{i-1}(v_i) &= \min_{v_{i-1}} s_{i-2}(v_{i-1}) + e_0(v_i) + e_1(v_{i-1}, v_i)
\end{aligned}$$

These tables have the important property that:

$$s_{i-1}(v_i) \quad = \quad min_{v_0 \ldots v_i} e(v_0, \ldots v_i)$$

In other words, given a fixed value for control point $v_i$, the optimal location for the prior control point $v_{i-1}$ can be looked up as $s_{i-1}(v_i)$.

In addition to the energy tables, we also create a corresponding table that can be backtracked to find the optimal stroke with the corresponding optimal energy:

$$P_{i-1}(v_i) = \arg\min_{v_{i-1}} s_{i-2}(v_{i-1}) + e_0(v_{i-1}) + e_1(v_{i-1}, v_i) \tag{3}$$

Using these tables, we can see that the optimal choice for the last control point, $v_n$, is given by

$$v_n = \arg\min_{v_n} s_{n-1}(v_n) \tag{4}$$

The rest of the stroke is found recursively:

$$v_{i-1} = P_{i-1}(v_i) \tag{5}$$

Furthermore, we can find the length $m$ of the locally optimal curve of any length (up to $n$) simply by searching the tables for the smallest value:

$$m = \arg \min_{minLength \leq m \leq maxLength} \min_{v} s_{m-1}(v) \tag{6}$$

This operation requires minimal overhead, because we can keep track of the minimum entry while building the tables.

We also add a control point $v_{n+1} = 2v_n - v_{n-1}$ to the stroke before each relaxation step, and search near this longer stroke. This allows the stroke length to increase as well as decrease during the search.

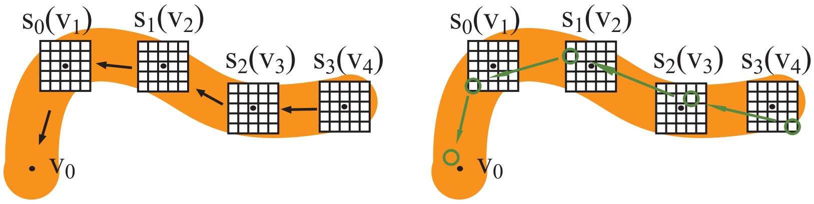



Figure 2: *Left:* Dynamic programming for computing a locally optimal stroke shape. A $w^2$ table is constructed around every control point, except the first. $s_{i-1}(v_i)$ is a table that contains the energy of the optimal stroke for a given value for $v_i$. *Right:* The energy of the optimal stroke of length $n$ can be found by searching the $n$-th table: $e_{min} = \min_{v_n} s_{n-1}(v_n)$.

The computational complexity of a single search step is linear in the length of the stroke, but exponential in the order of the energy: $\Theta(n(w^2)^k)$ energy evaluations are required for a stroke of order $k$ and length $n$ For $k = 2$ and $w = 5$ this means 625 computations per control point per iteration, where each computation may be quite expensive.

For stroke evolution, we chose dynamic programming for the advantage of generality. Gradient-based methods such as used by Kass et al. [KWT87] require the energy function to be differentiable, and it appears that a graph-theory formulation [MB95] of this problem would be NP-complete.

### 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 [Car84], 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 based from a mouse click, and rerendering strokes with pseudocolors.

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 Texture Buffer

Our system allows the user to experiment with modifying the stroke rendering styles. Because stroke positions are not modified during this process, it would be wasteful to repeatedly scan-convert strokes while changing colors. We use a texture buffer to avoid this effort. The texture buffer is a fragment buffer with additional texture coordinates provided with each fragment. The fragment also contains partial derivatives for use in the procedural texture; the complete texture fragment data structure contains:

- $(u, v)$ texture coordinates
- Jacobian matrix of partials: $\begin{bmatrix} \frac{\partial s}{\partial x} & \frac{\partial t}{\partial x} \\ \frac{\partial s}{\partial y} & \frac{\partial t}{\partial y} \end{bmatrix}$, where $s(x, y)$ and $t(x, y)$ define the mapping from screen space to stroke texture space. The partials are computed as a product of stroke scan-conversion.
- Stroke index, used to look up the stroke color and texturing function.

Before experimenting, we first scan convert all brush strokes into the texture buffer. Rendering with new textures and colors is then a matter of traversing the texture buffer, calling the appropriate procedure for each texture coordinate, and compositing. The texture buffer could also be used to incorporate texture into relaxation; we currently do not do this for efficiency concerns.

### 8.4 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. One connection is used between each client and the server; in our implementation, this takes the form of a single UNIX socket on 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. Although there are many ways of generating suggestions, all suggestions can be written as one of two types of messages: **Delete** stroke number $n$, or **Create/modify** stroke number $n$ to have radius $R$, color $C$, and control polygon $P$. (There is no distinction between “create” and “modify” commands.) The same protocol is used for sending suggestions to the server as for announcing changes to the clients. Two other message types are used: one message tells clients to advance to the next frame in a video sequence, and another announces changes to the style parameters made by the user.

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. In this setup, a main copy of the painting resides on shared storage. Whenever a processor generates a suggestion, it is placed on a global suggestion queue. Before computing a suggestion, a processor must first test every suggestion in the queue and clear it out if the queue is not already in use, to keep the painting up-to-date. The image is broken into blocks, and before testing a suggestion, the processor acquires a write lock on the affected image blocks.

## 9 Improved Search Methods

In this section, we speculate on how standard methods can be applied to improve the solutions detected by the relaxation algorithm.

The methods in this section are primarily designed to find better local minima, not to speed up the search. A pass of relaxation at full resolution seems to be required after any other computations, to clean up any errors resulting from simplifying the problem.

### 9.1 Coarse-to-Fine

We have experimented with one standard image processing technique, coarse-to-fine processing. The idea is to perform relaxation on a subsampled version of the problem, and use the result as a starting point for relaxation on the complete image. Reducing the image size is like increasing the search window around each control point during stroke relaxation. The complexity of the problem is reduced because the image is smaller, but fine details are lost.

Resampling the painting requires producing a new relaxation problem that is equivalent to the old one, at a different scale. The source and weight images are resampled, and the stroke control points and radii are adjusted as well. The energy function is modified that the energy for the new painting is equivalent to the old energy, up to a scale factor. This is done by observing the effect of rescaling the painting on the energy function: for example, if the image is doubled in size, the values of all terms of the energy function quadruple except for the number of strokes weight ($w_{nstr}$). Therefore, an equivalent energy function is achieved by multiplying $w_{nstr}$ by the scale factor squared. For example, if the image width and height are each being divided in half, then $w_{nstr}$ should be divided by 4.

In our experiments thus far, coarse-to-fine processing has given worse results than processing without coarse-to-fine. One possible explanation for this is that the details lost to subsampling are so significant that the minima of the subsampled problem are not a good approximation to the full problem. If so, then this is evidence that the energy function is very poorly-behaved.

### 9.2 Monte Carlo

A standard relaxation technique are the Monte Carlo methods. In these methods, we replace the relaxation loop with the following:

$P \leftarrow$ empty painting

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

The variable $p$ is a threshold that decreases from 1 to 0 by some predefined or automatic schedule during the relaxation, and RANDOM is a random-number generator in the range $[0, 1]$. Accepting suggestions that appear to be unprofitable allows the relaxation to escape from local minima. As relaxation proceeds, $p$ goes to 0, the relaxation settles into a local minimum.

### 9.3 Other Annealing Methods

The general strategy of annealing is to reduce the complexity of the energy function, for example, by blurring the energy function. Other forms of annealing may be possible, such as by removing terms from the energy function, or increasing the weights. For example, we could reduce the number of strokes weight to get a lot of brush strokes, then increase the cost so that extra strokes will be deleted.

## 10 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.

## 11 Future Work

In this section, we outline some of the many avenues available for extending this 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. Preliminary experiments with a pairwise temporal coherence energy have been unsuccessful because the brush strokes generated for a single frame typically do not correspond sufficiently to scene geometry. 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.

# A Brush Stroke Rendering

In this chapter, we describe methods we have used for scan-converting and texturing brush strokes. However, methods more sophisticated than these are available (e.g. [HLW93, Met, NM00]).

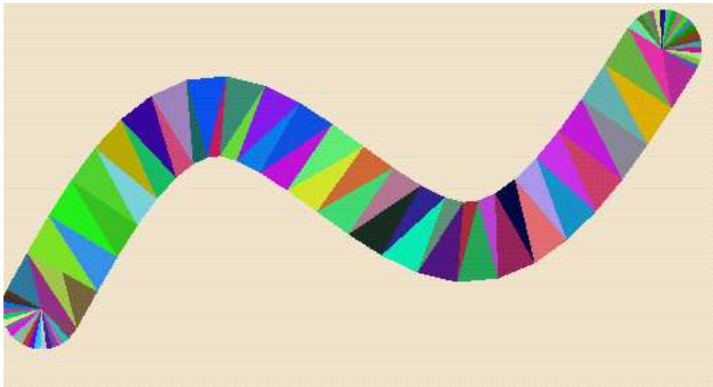
## A.1 Brush Stroke model

In our system, a basic brush stroke is defined by a curve, a brush thickness $R$, a stroke color $C$. The stroke is rendered by placing the stroke color at every image point that is within $R$ pixels of the curve. The curve is an endpoint-interpolating cubic B-spline defined by a set of control points. A dense set of curve points can be computed by recursive subdivision.

Our original implementation, used for the figures and video in [Her98], was chosen for ease of implementation, and simply swept an arbitrary brush profile along a curve, with some fancy bookkeeping to correctly handle stroke opacity. In the remaining implementation and images, we used the faster method described in the next section.

## A.2 Rendering with Triangle Strips

Our basic technique for scan-converting a brush stroke is to tessellate the stroke into a triangle strip:



Given a moderately dense list of control points $\mathbf{p}_i$ and a brush thickness $R$ we can tesselate the stroke by the following steps:

1. Compute curve tangents at each control point. An adequate approximation to the tangent for an interior stroke point $\mathbf{p}_i$ is given by $\mathbf{v}_i = \mathbf{p}_{i+1} - \mathbf{p}_{i-1}$. The first and last tangents are $\mathbf{v}_0 = \mathbf{p}_1 - \mathbf{p}_0$ and $\mathbf{v}_{n-1} = \mathbf{p}_{n-1} - \mathbf{p}_{n-2}$.

2. Compute curve normal directions as perpendicular to the tangents: $\mathbf{n} = (\mathbf{n}_{xi}, \mathbf{n}_{yi}) = (\mathbf{v}_{yi}, -\mathbf{v}_{xi})/||\mathbf{v}_i||$

3. Compute points on the boundary of the stroke as points offset by a distance $R$ along the curve normal direction. The offsets for a control point are $\mathbf{a}_i = \mathbf{p}_i + R\mathbf{n}_i$ and $\mathbf{b}_i = \mathbf{p}_i - R\mathbf{n}_i$.

4. Tesselate the stroke as shown above.

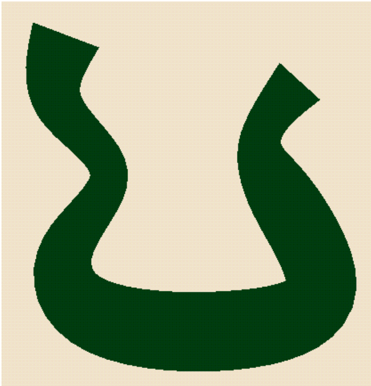


Figure 3: A variable-thickness stroke. Thicknesses were randomly generated for each control point.

5. If desired, add circular “caps” as triangle fans.

This algorithm can also be used with varying brush thickness (Figure 3), by specifying a profile curve for the thickness. We do this by assigning a thickness for each control point, and subdividing the thicknesses at the same time as subdiving the control point positions.

This method fails when the stroke has high curvature relative to brush thickness and control point spacing. Such situations can be handled, for example, by repeated subdivision near high curvature points. Generally, we have not found these errors to be of much concern, although they may be problematic for high-quality renderings.

### A.3 Procedural Brush Textures

Paintings with the texture and appearance of real media can add substantial appeal to a painting. The first step in creating stroke textures is to define a parameterization for a stroke [HLW93]. A stroke defines a mapping from $(x, y)$ screen coordinates to $(s, t)$ texture coordinates. The parameter $s$ varies along the spine of the stroke in proportion to the curve’s arc length, and the parameter $t$ varies normal to the curve from $-1$ to $1$, as illustrated in Figure 4. Multiplying $t$ by the brush radius gives a parameterization proportional to distance in image space.

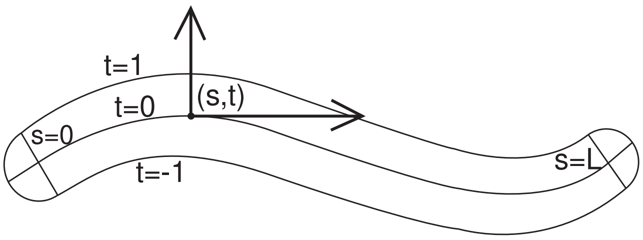



Figure 4: Parameterization of a brush stroke

We have explored the use of procedural textures for providing brush stroke textures. Procedural textures are typically faster than paint simulation approaches based on cellular automata [Sma90, CAS+97] and provide finer control over appearance; on the other hand, using procedural textures make it more difficult to achieve the complex, naturalistic appearance produced by simulation. Paper texture and canvas can also be synthesized procedurally [Wor96, Cur99]. Scanning images of real paint strokes is an excellent way to get good stroke textures. We currently employ a simple bristle texture given by $bristle(s,t) = noise(c_1 s, c_2 t)$, where $noise(x,y) \in [-1,1]$ is a noise function [Per85], and $c \approx 1000$. This function can be used for the stroke opacity, or as a height field for paint lighting; we compute lighting as a function of the dot product between the view direction and the slope of the height field: $I(x,y) = f(L^T \nabla H(x,y))$, where $H(x,y)$ is the height field at $(x,y)$. We have used $f(\rho) = |\rho| + .8$. The stroke texture itself provides the height field parameterized in texture space: $H(s,t)$. The stroke parameterization also defines a mapping from screen space to texture space as $(s(x,y), t(x,y)$; the derivative of this mapping can be computed during stroke scan-conversion and summarized by the Jacobian matrix

$$J(x,y) = \begin{bmatrix} \frac{\partial s}{\partial x} & \frac{\partial t}{\partial x} \\ \frac{\partial s}{\partial y} & \frac{\partial t}{\partial y} \end{bmatrix} \tag{7}$$

The intensity can be computed as:

$$\begin{aligned}
H(x,y) &= h(s(x,y), t(x,y)) \\
\nabla H(x,y) &= \begin{bmatrix} \frac{\partial h(s(x,y),t(x,y))}{\partial x} \\ \frac{\partial h(s(x,y),t(x,y))}{\partial y} \end{bmatrix} \\
&= \begin{bmatrix} \frac{\partial h}{\partial s}\frac{\partial s}{\partial x} + \frac{\partial h}{\partial t}\frac{\partial t}{\partial x} \\ \frac{\partial h}{\partial s}\frac{\partial s}{\partial y} + \frac{\partial h}{\partial t}\frac{\partial t}{\partial y} \end{bmatrix} \\
&= J(x,y) \begin{bmatrix} \frac{\partial h}{\partial s} \\ \frac{\partial h}{\partial t} \end{bmatrix} \\
&= J(x,y) \nabla h(s(x,y), t(x,y)) \\
I(x,y) &= f(L^T \nabla H(x,y)) = f(L^T J(x,y) \nabla h(s(x,y), t(x,y)))
\end{aligned}$$

Additionally, a soft edge can be added to the stroke by multiplying the stroke opacity by a falloff curve parameterized in $t$.

## Acknowledgments

We are grateful to Ken Perlin and Denis Zorin for fruitful discussions and feedback, to Robert Buff, Will Kenton and Celine Wei for help with capturing input images and video, and to Philip Greenspun (http://photo.net/philg) for input images for Figures 8, 10, 11, and 12. The author is supported by NSF Grant DGE-9454173.

## References

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

[AWJ90] Amir A. Amini, Terry E. Weymouth, and Ramesh C. Jain. Using Dynamic Programming for Solving Variational Problems in Vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):855–867, September 1990.

[Car84] Loren Carpenter. The A-buffer, an Antialiased Hidden Surface Method. *Computer Graphics (SIGGRAPH '84 Proceedings)*, 18(3):103–108, July 1984.

[CAS+97] Cassidy J. Curtis, Sean E. Anderson, Joshua E. Seims, Kurt W. Fleischer, and David H. Salesin. Computer-Generated Watercolor. In *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 421–430, August 1997.

[CGG+99] 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.

[Cur99] Cassidy Curtis. Non-photorealistic animation. In Stuart Green, editor, *Non-Photorealistic Rendering*, SIGGRAPH Course Notes, chapter 6. 1999.

[Dan99] Eric Daniels. Deep Canvas in Disney's Tarzan. In *SIGGRAPH 99: Conference Abstracts and Applications*, page 200, 1999.

[GSG+99] Barton Gawboy, Mark Schafe, Michael Gleicher, Zoran Popović, and Jeffrey Thingvold. *Motion Editing: Principles, Practice, and Promise*. SIGGRAPH Course Notes. 1999.

[Hae90] 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.

[Her98] Aaron Hertzmann. Painterly Rendering with Curved Brush Strokes of Multiple Sizes. In *SIGGRAPH 98 Conference Proceedings*, pages 453–460, July 1998.

[HLW93] S. C. Hsu, I. H. H. Lee, and N. E. Wiseman. Skeletal strokes. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, Video, Graphics, and Speech, pages 197–206, 1993.

[HP00] Aaron Hertzmann and Ken Perlin. Painterly Rendering for Video and Interaction. In *Proceedings of the First Annual Symposium on Non-Photorealistic Animation and Rendering*, June 2000. To appear.

[KPC93] John K. Kawai, James S. Painter, and Michael F. Cohen. Radioptimization — Goal Based Rendering. *Proceedings of SIGGRAPH 93*, pages 147–154, August 1993.

[KWT87] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active Contour Models. *International Journal of Computer Vision*, 1(4), 1987.

[Lit97] Peter Litwinowicz. Processing Images and Video for an Impressionist Effect. In *SIGGRAPH 97 Conference Proceedings*, pages 407–414, August 1997.

[Lit99] Peter Litwinowicz. Image-Based Rendering and Non-Photorealistic Rendering. In Stuart Green, editor, *Non-Photorealistic Rendering*, SIGGRAPH Course Notes. 1999.

[MB95] Eric N. Mortensen and William A. Barrett. Intelligent Scissors for Image Composition. *Proceedings of SIGGRAPH 95*, pages 191–198, August 1995.

[Mei96] Barbara J. Meier. Painterly Rendering for Animation. In *SIGGRAPH 96 Conference Proceedings*, pages 477–484, August 1996.

[Met] MetaCreations. Painter.

[NM00] J.D. Northrup and Lee Markosian. Artistic Strokes. In *Proceedings of the First Annual Symposium on Non-Photorealistic Animation and Rendering*, June 2000. To appear.

[Per85] Ken Perlin. An Image Synthesizer. *Computer Graphics (Proceedings of SIGGRAPH 85)*, 19(3):287–296, July 1985.

[Rig99] Right Hemisphere Ltd. Deep Paint, 1999. Software package.

[SAH91] 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.

[Sma90] David Small. Modeling Watercolor by Simulating Diffusion, Pigment, and Paper fibers. In *SPIE Proceedings*, volume 1460, 1990.

[Str86] Steve Strassmann. Hairy Brushes. In David C. Evans and Russell J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 225–232, August 1986.

[SWHS97] Michael P. Salisbury, Michael T. Wong, John F. Hughes, and David H. Salesin. Orientable Textures for Image-Based Pen-and-Ink Illustration. In *SIGGRAPH 97 Conference Proceedings*, pages 401–406, August 1997.

[TB96] Greg Turk and David Banks. Image-Guided Streamline Placement. In *SIGGRAPH 96 Conference Proceedings*, pages 453–460, August 1996.

[Wor96] Steven P. Worley. A Cellular Texture Basis Function. *Proceedings of SIGGRAPH 96*, pages 291–294, August 1996.

[WW92] William Welch and Andrew Witkin. Variational surface modeling. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):157–166, July 1992.

(a)

(b)

(c)

(d)

Figure 5: A source image, and painterly renderings with the method of [Her98] with 1, 2, and 3 layers, respectively. The algorithm has difficulty capturing detail below the stroke size.

(a) (b)

(c)

Figure 6: Paint by relaxation, with (a) one layer (for comparison with Figure A.3(b)) and (b) two layers (for comparison with A.3(c)). Strokes are precisely aligned to image features, especially near sharp contours, such as the lower-right corner of the jacket. (c) A looser painting with two layers.

(a)

(b)

Figure 7: Procedural brush textures applied to Figure 6(a) and Figure 6(c).


Figure 8: Spatially-varying style provides a useful compositional tool for directing attention. (a) Source image. (b) Interactively painted weight image ($w_{app}$). (c) Painting generated with the given weights. More detail appears near faces and hands, as specified in the weight image. (d) A more extreme choice of weights and focus. Detail is concentrated on the rightmost figures.

Figure 9: A looser painting style, with and without texture

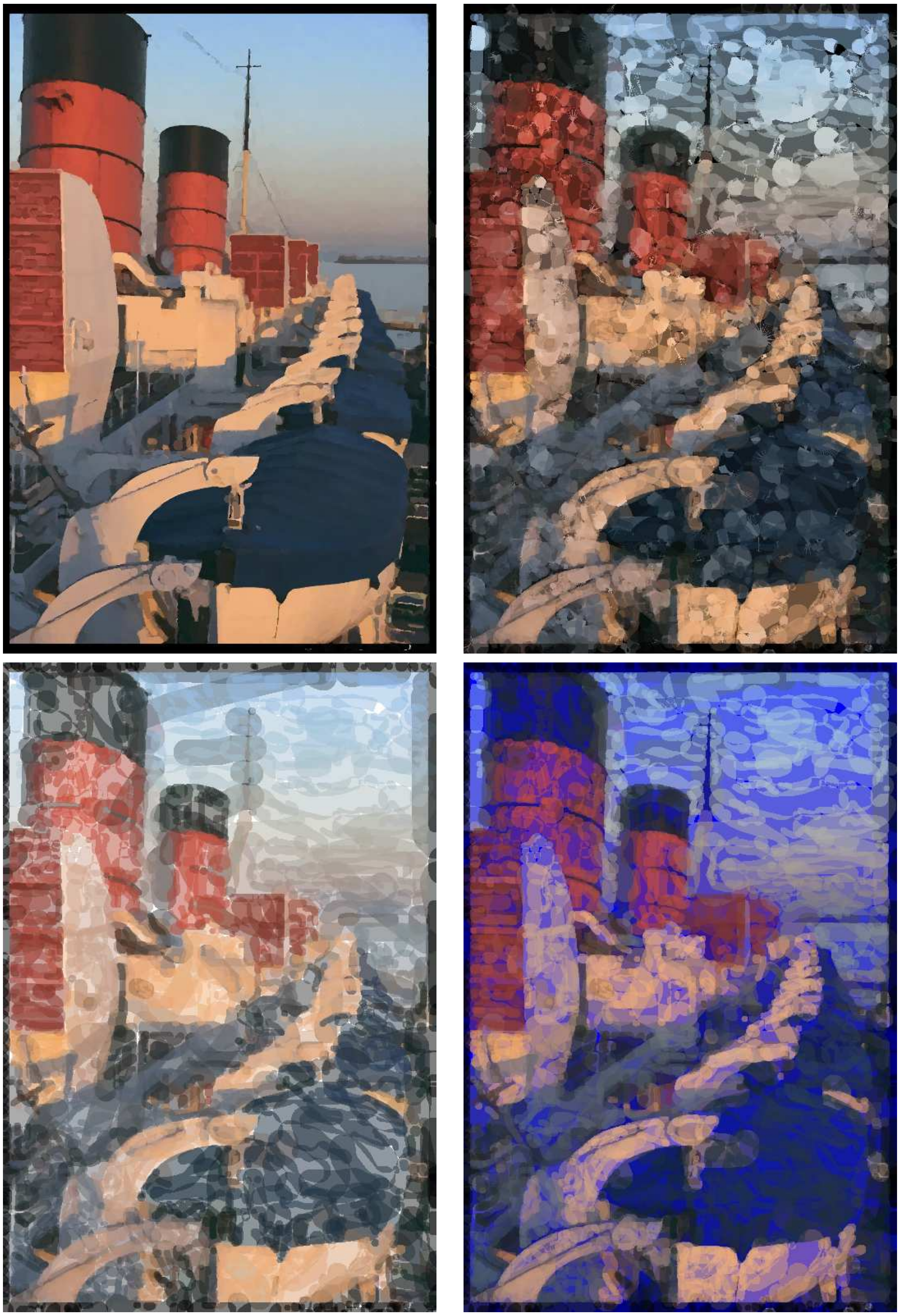




Figure 10: Queen Mary paintings, illustrating various uses of texture and transparency

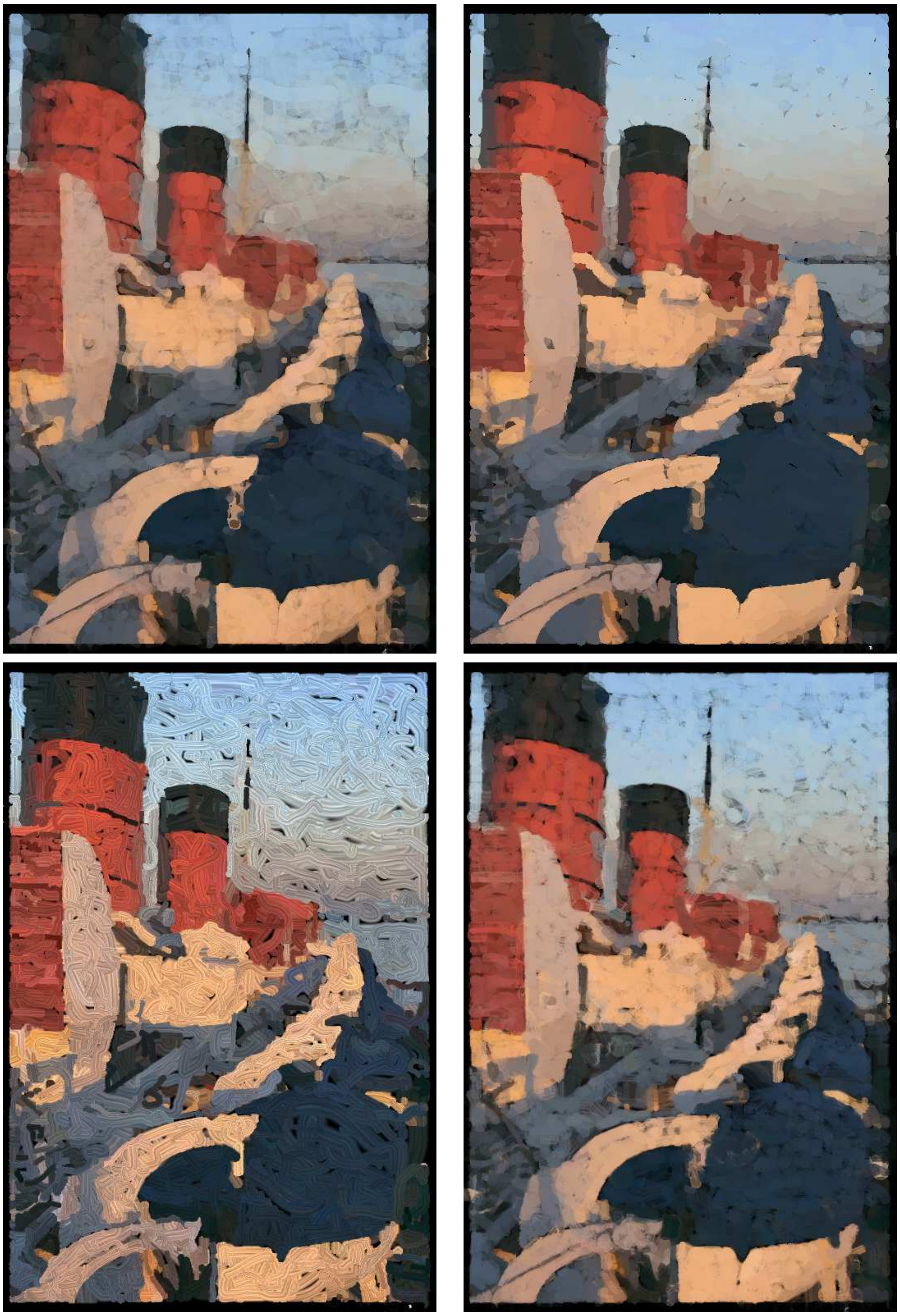


Figure 11: Queen Mary paintings, illustrating various uses of texture and transparency

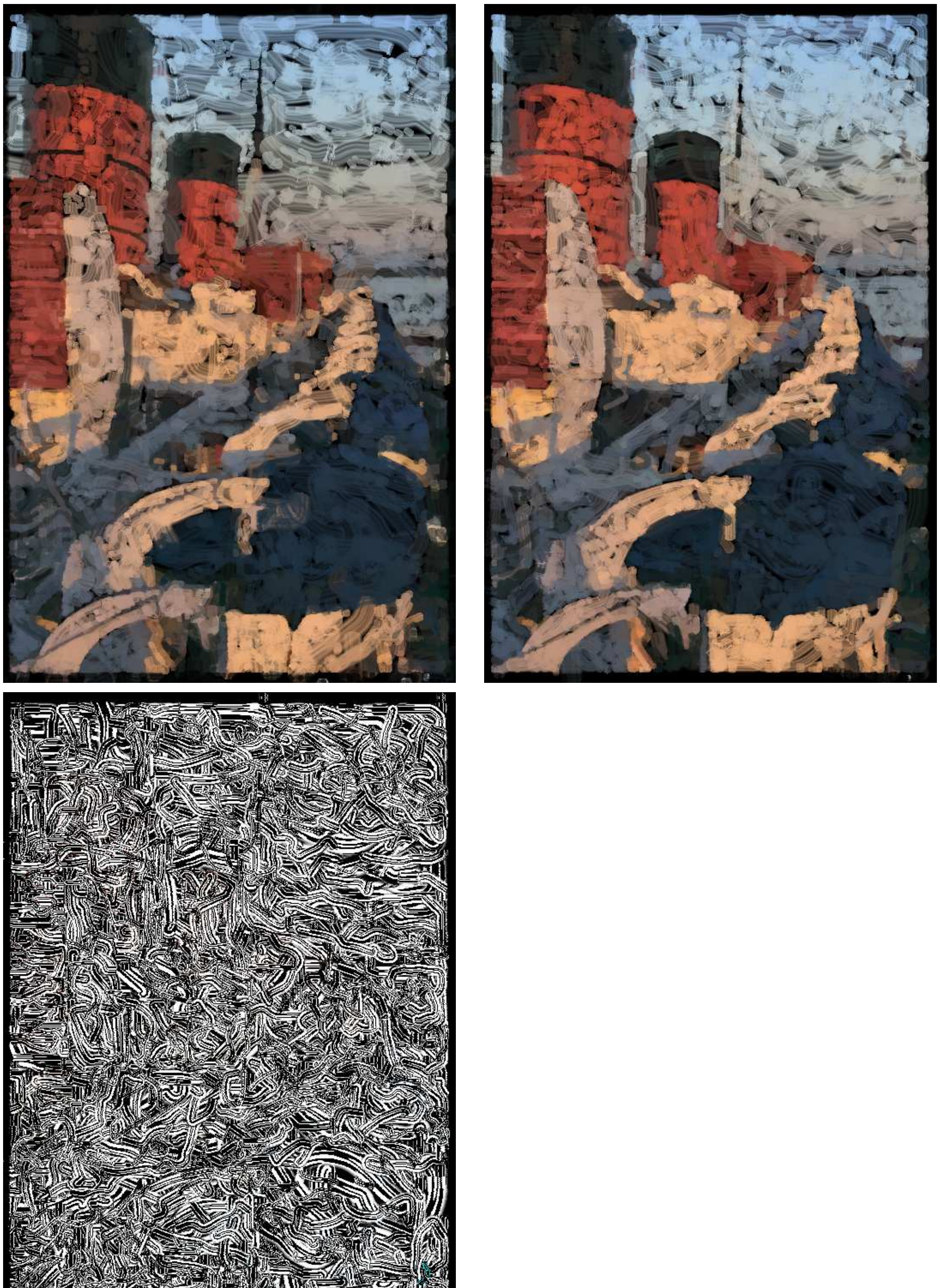




Figure 12: Queen Mary paintings, illustrating various uses of texture and transparency

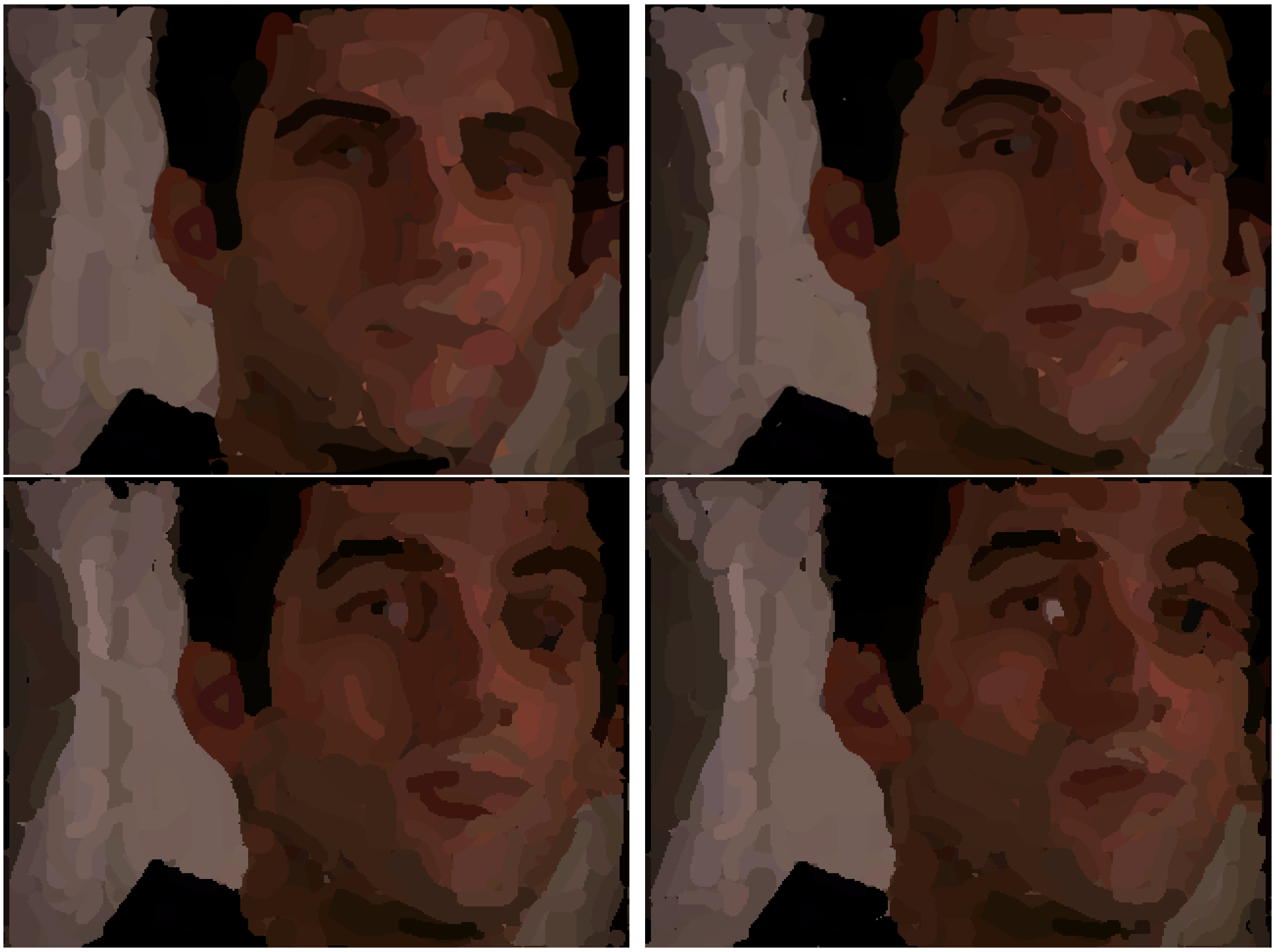


Figure 13: Consecutive frames from a painterly animation.