Stroke Based Painterly Rendering

David Vanderhaeghe, John Collomosse

Chapter 1
Stroke Based Painterly Rendering

David Vanderhaeghe and John Collomosse

Abstract Many traditional art forms are produced by an artist sequentially placing a set of marks, such as brush strokes, on a canvas. Stroke based Rendering (SBR) is inspired by this process, and underpins many early and contemporary Artistic Stylization algorithms. This Chapter outlines the origins of SBR, and describes key algorithms for placement of brush strokes to create *painterly renderings* from source images. The chapter explores both local greedy, and global optimization based approaches to stroke placement. The issue of creative control in SBR is also briefly discussed.

1.1 Introduction

Stroke based Rendering (SBR) is the process of synthesizing artwork by compositing rendering marks (such as lines, brush strokes, or even larger primitives such as tiles) upon a digital canvas. SBR under-pins many Artistic Rendering (AR) algorithms, especially those algorithms seeking to emulate traditional brush-based artistic styles such as oil painting.

The SBR paradigm was proposed in the early nineties by Paul Haeberli [8], in the context of his semi-automated 'Paint By Numbers' painting environment that sought to rendering impressionist paintings from photographs. Although this work is now regarded as having catalyzed the field of AR, Haeberli's stated intention was to improve the richness of manually created digital paintings. Such paintings

David Vanderhaeghe
IRIT, CNRS, Université Paul Sabatier, 118 route de Narbonne, Toulouse, FRANCE.
e-mail: vdh@irit.fr

John Collomosse
Centre for Vision Speech and Signal Processing – University of Surrey, Guildford, Surrey, GU2 7XH, UK.
e-mail: j.collomosse@surrey.ac.uk

Fig. 1.1 Stroke based Rendering. Left: Impressionist rendering created interactively using Haeberli's Paint by Numbers algorithm [8]. The source image in the top-right is rendered as the user clicks upon the canvas with pre-selected stroke sizes. Right: Driving the interactive elements of this process using a pseudo-random number generator will cause loss of salient detaill [9].

frequently lacked color depth, which Haeberli attributed to a prohibitively paintings lacked lengthy 'time to palette'; the time taken by the user to select a new color.

Haeberli's concept was simple but effective. The user interacts with a canvas of identical size to the source photograph they wish to render in a painterly style. Each time the user clicks to place a brush stroke on the digital canvas, the color of that stroke is sampled from the corresponding position in the source photograph. The strokes are much larger than the pixels from which the color is sampled, and this leads to an abstraction of detail reminiscent of that seen in real paintings. Furthermore, the noise inherent in the sampling of individual pixel color leads to a color variation reminiscent of the impressionist style (this can be further exaggerated by addition of Gaussian noise to the RGB values of the color).

Haeberli's system not only automates the selection of color, but also drives other stroke attributes such as orientation. In impressionist artwork, strokes are often painting tangential to edges in the scene. This can be emulated by running an edge detector (e.g. the Sobel operator) over the greyscale source photograph $I(x,y)$:

$$\Theta(x,y) = atan\left(\frac{\delta I}{\delta y}/\frac{\delta I}{\delta x}\right) \tag{1.1}$$

i.e. strokes are painted so that their longest axis is orthogonal to $\Theta(x,y)$. Fig. 1.1 illustrates the effect achieved using an open implementation of Haeberli's system [1].

Note that in Haeberli's system the user selects the size of stroke, the position on the canvas, and the order in which strokes are laid down. The set of stroke attributes are summarized in Table 1.1.

Under Haeberli's framework, strokes behave rather like 'rubber stamps' — an image of a stroke is painted centered at $\mathbf{P}$ and oriented by θ. The texture and visual properties of the stroke (beyond the color) are decoupled from the representation

[1] Available at `http://kahlan.eps.surrey.ac.uk/EG2011`

Attribute	Name	Derived...
$\mathbf{P} = (x, y)$	stroke seed position	Manually
s	stroke scale	Manually
$\theta = \Theta(x, y)$	stroke orientation	Automatically
$\mathbf{c}$	RGB color	Automatically
o	stroke order	Manually

Table 1.1 Haeberli's representation of a painting as an ordered list of strokes underpins many later SBR algorithms.

of the ordered stroke list. This representation is therefore highly versatile and can be used to represent pastel, crayon, oil paint etc. Later, more sophisticated AR algorithms sought to automate beyond these paint daubs, to produce elegant curved brush strokes [12]. We discuss these approaches later in section 1.2.3.

In this Chapter we restrict ourselves solely to the matter of semi-automatic or automatic stroke placement. We briefly consider the simple texturing of strokes, but refer the reader to Chapter 2 for a more detailed discussion of brush and media simulation. In line with most AR research papers, we consider the matters of stroke rendering and stroke placement as decoupled.

1.2 Iterative approaches to Automatic Painting

Haeberli's interactive systems catalyzed the development of fully automated painterly rendering algorithms, raising research questions such as "can the user be left out of the rendering work-flow?" and if so, "to what degree is is desirable to do so?". As we describe later in Chapter 13, increased automation has also opened up the possibility of video stylization. The issue of user control in discussed further section 1.4.

A trivial adaptation to fully automate Haeberli's pipeline is to drive the values of manually set attributes with a pseudo-random number generator [9]. However, randomizing the order of strokes and their sizes can cause a loss of important ('salient') detail in the image as Fig. 1.1 illustrates.

A human artist will typically over-paint fine strokes, on top of coarser strokes, to depict fine important details in the rendering. Therefore we can link rendering order and stroke size to a simple automated measure of detail derived from the image. Since we already compute first derivative edge information to derive $\Theta(x, y)$ we can also derive a measure of edge magnitude $|\nabla I(x, y)|$:

$$|\nabla I(x,y)| = \left(\frac{\delta I}{\delta x}^2 + \frac{\delta I}{\delta y}^2 \right)^{\frac{1}{2}} \tag{1.2}$$

Strokes should be scaled size in inverse proportion to $|\nabla I(x, y)|$. The stroke ordering should be modified so as to paint smaller strokes later, over-painting coarser details.

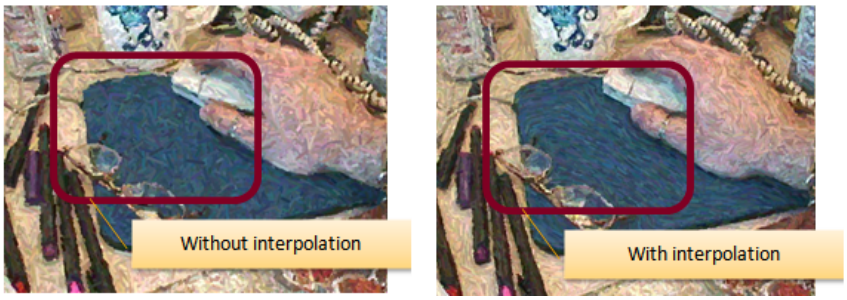

Fig. 1.2 Stroke detail in impressionist rendering. Comparison of painting via Litwinowicz' method [17] with and without interpolation of orientation gradient. ©1997 ACM, images used by permission.

1.2.1 Automated Impressionist Painting

Litwinowicz published the first automated painterly rendering algorithm in 1997 [17]. Litwinowicz' technique not only automated the rendering process, reducing user interaction to parameter setting, but also extended painterly rendering to video. We describe the latter aspects of the algorithm in further detail within Chapter 13.

Litwinowicz addresses the stroke placement and ordering problem in a straightforward manner. Pixel locations are sub-sampled in a regular grid, and a stroke generated at each sampled location. In practice, strokes are generated densely typically at every other pixel location. These strokes are rectangular in shape, and rendered in a random order. As performed by Haeberli's system, the color and orientation of each stroke is determined automatically. The latter using the Sobel operator (first derivative of intensity) as per eq.1.1.

To prevent loss of detail, each stroke is clipped against strong edges in the image. These are detected via thresholding field $|\nabla I(x,y)|$ at a constant value (eq. 1.2). The stroke may be thought of as a rectangle centered upon the initial stroke position, and oriented to align with the local edges in the image. The rectangle extends from the center position outward until a strong edge is met or the stroke exceeds a maximum length. This process prohibits strokes to cross strong image edges, preventing loss of detail through "coloring outside the lines".

Unfortunately the gradient field $\Theta(x,y)$ (eq. 1.1) does not offer reliable values over the whole image, and provides a noisy estimate when the source image gradient varies smoothly, or not at all. To provide smooth direction flow and mitigate noise, a more robust direction flow computation is usually required.

Gradient direction is reliable when the magnitude of edge gradient $|\nabla I(x,y)|$ is high, i.e. close to image edges and other high frequency artifacts. A practical approach to obtain a cleaner direction flow is to estimate the gradient direction local to such artifacts and interpolate elsewhere. Litwinowicz proposed the use of thin-plate spline interpolation [17], resulting in an improved aesthetic (Fig.1.2). The mathematics of this interpolation are covered in more detail within Chapter 8.

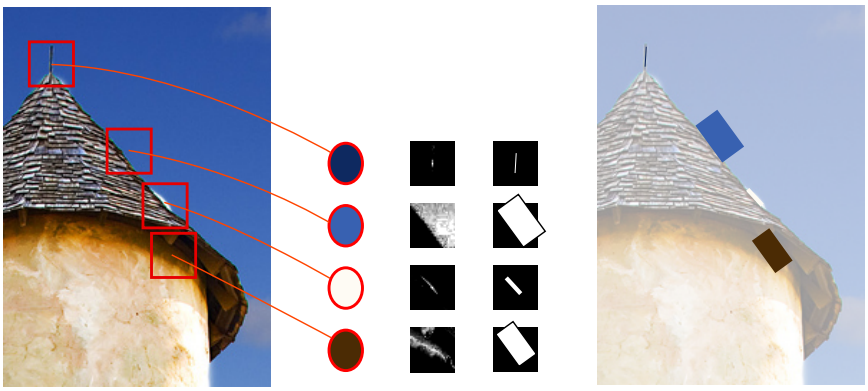

Fig. 1.3 The iterative approach of Shiraishi and Yamaguchi [20]. a) source image, b) pixel color for four sample pixels, c) difference between the pixel color and surrounding pixel colors is stored in an array, d) moment analysis of the array gives a rectangle with the same moments, e) for each anchor point, the attributes of the rectangle computed for this pixel are read to render the stroke.

Radial basis function are also well adapted to perform directional interpolation, and were explored by Hays and Essa [11] for video painting. But interpolation of vector fields $\frac{\delta I}{\delta x}$ or $\frac{\delta I}{\delta y}$ do not provide results consistent with the fact we only care about orientation angle; considering the direction flow as a tensor field provides better results as described by Kagaya et al. [16]. This extends earlier work exploring the interactive editing of the directional field by Zhang et al. [26].

1.2.2 Painterly Rendering using Image Moments

Shiraishi and Yamaguchi [20] proposed an alternative mechanism to orient and scale strokes, that does not rely upon edge gradient. Rather, they rely upon 2D statistical moments computed local to each stroke; an approach suggested earlier by Treavett and Chen [23]. Shiraishi and Yamaguchi compute, for each pixel p of the source image, the difference in color between p and pixels around p in a window of user specified scale. The result of each difference is stored in an array A of the same size of the window as indicated in Fig. 1.3.

The system computes the 2D central moments of A to define a rectangular stroke that approximates the region of similar color in the neighborhood of p. Strokes are aligned so that their principal axis is aligned with the principal eigenvector of the difference array. The output image may be over-painted in several passes from coarse scales, to finer scales, reminiscent of Hertzmann's multi-resolution paint process (described shortly in subsection 1.2.3). As with most painterly rendering algorithms, the stroke color is sampled locally from the source image.

1.2.3 Multi-resolution Painting using Curved Strokes

Early painterly rendering algorithms [8, 17, 23] were limited to placing simple daubs of paint. Essentially these were 2D sprites (textured rectangles) resembling a brush stroke, that were scaled and oriented by the stroke placement process.

Hertzmann proposed an alternative paradigm for painting in 1998 [12], that progressed painterly rendering with two key innovations. First, the painting process uses long, curved β-spline strokes rather than sprites for painting. This greatly enhances the realism and aesthetics of the resulting renderings. Second, the painting process was performed iteratively over multiple "layers" from coarse to fine. The painting process for each layer was driven by a source image down-sampled to a particular spatial resolution. For this reason this innovation is sometime referred to as multi-resolution painterly rendering.

1.2.3.1 Curved Stroke Formation

As with prior work, Hertzmann's algorithm makes independent decisions for each stroke's placement based on information in the source image local to the stroke position. Execution proceeds in a local, greedy manner with each stroke being rendered in in a pre-determined randomized order (after [17]). We first describe how a single layer is painted; this process is summarized in Fig. 1.1.

Each stroke is generated independently via a process of "hopping" between pixels, reminiscent of Line Integral Convolution (LIC). Given a starting point or "seed" pixel, the stroke by jumping from the current (seed) pixel to another pixel a predetermined distance away. The direction of jump is derived from the gradient field $\Theta(x,y)$ of eq. 1.1. Specifically the jump is made orthogonal to this angle in order that the stroke is formed tangential to edge structure in the source image. At each hope there is 180° ambiguity in the hopping direction, the direction of hop that minimize the curvature of the stroke being formed is taken. Stroke growth is terminated when the color of the image pixels departs significantly from the color of the stroke being formed. A minimum and maximum size for strokes is also enforced.

Each pixel visited forms a control point for a spline curve. Thus the eventual shape of the stroke is defined by a path which is built by following the direction flow. In Hertzmann's method these control points are approximated by a $\beta-$spline, which helps to smooth noise. However other closely related painting techniques that perform a similar hopping (e.g. the genetic paint system of subsection 1.3.2 use an interpolating Catmull-Rom spline for greater accuracy.

1.2.3.2 Coarse to fine painting

Hertzmann's algorithm generates layers of paint strokes, from coarse to fine. Prior to painting, a low-pass pyramid is generated by blurring and sub-sampling the source image at a range of decreasing scales. Typically octave intervals are used for each

Algorithm 1.1 Hertzmann's algorithm for build the list of control point that form a brush stroke, used in Alg. 1.2. Here the functions $I_{src}(.)$ and $I_{paint}(.)$ denote the RGB pixels values in the source image and painting respectively.

```
1: function MAKESTROKE(x, y) return stroke
2:     d ← Θ(x, y)
3:     strokeColor ← I_ref(x, y)
4:     stroke.pushControlPoint(x, y)
5:     for i: 1 → maxSize-1 do
6:         (x, y) ← (x, y) + d
7:         d′ ← Θ(x, y)
8:         d ← f_dir d + (1 − f_dir) d′
9:         srcColor = I_src(x, y)
10:        paintColor = I_paint(x, y)
11:        if i >=minimumSize AND |srcColor−strokeColor| > |srcColor−paintColor| then
12:            break
13:        end if
14:        stroke.pushControlPoint( x, y)
15:    end for
16: end function
```

level of the pyramid. For example, a low-pass pyramid of 4 levels would comprise images $\{1,2,4,8\}$ times smaller than the original. The coaresest rendering layer is generated from the most heavily sub-sampled image.

The coarsest painting layer is first generated by creating strokes for all pixels, using the algorithm described in subsection 1.2.3.1. Stroke size (radius) is proportional to the degree of sub-sampling used at the corresponding layer of the low-pass pyramid, and reflects the scale of visual features expected to occur at that level.

Subsequent layers are then painted by compositing over strokes already laid down in previous layers. However, unlike the first (coarsest) layer, the algorithm does not paint the entire layer (as this would entirely occlude the previously painted layer). Rather, the algorithm selects which areas must be repainted by monitoring differences between the current and previous layers in the low-pass pyramid. This is equivalent to detecting which visual details have been revealed by moving to a higher resolution layer in the low-pass pyramid. Specifically, to trigger the start of a new stroke, Hertzmann's approach computes the sum of squares differences over a cell of size equivalent to the stroke's radius. If this difference is over a user defined threshold, then the stroke is painted. The algorithm is summarized in Fig. 1.2.

Figure 1.4 illustrates the results of the painting process across successive layers via this multi-resolution algorithm. The main drawback is the repeated over-painting of edges and discontinuities in the scene. Strokes painted at such discontinuities cause large differences between successive layers, so triggering the edge for re-painting at each level of detail. To mitigate against this Huang et al. [15] propose to use multiple brush sizes per layer, rather than a constant size. Huang et al. define a grid with varying cell size (Fig. 1.5) and draw strokes starting in each cell with a radius function of grid size. Cell size is adapted to reflect a pre-supplied importance map. A trivial importance map might be simply the edge magnitude field $|\nabla I(x,y)|$, however as discussed later, such maps are better derived from automated salience

Algorithm 1.2 Hertzmann's coarse to fine painting algorithm [12]

```
1: function PAINTIMAGE(I_in)
2:     for all R in Radius from largest to smallest do
3:         I_ref ← F(I_in, R)
4:         GridSize ← R
5:         for all cell C of the grid spacing GridSize do
6:             A ← Σ_{(i,j)∈C} |P(i,j) − I_ref(i,j)|
7:             if A > t then                          ▷ t is a user defined threshold
8:                 (x,y) ← argmax_{(i,j)∈C} |P(i,j) − I_ref(i,j)|
9:                 S ← makeStroke(x,y)
10:                paintStroke( P, S)                 ▷ paint S onto the virtual canvas P
11:            end if
12:        end for
13:    end for
14: end function
```

measures. Each cell is divided along the longest axis until the sum of the importance map it covers is below a user defined threshold, or a minimum size is reached. The axis aligned cutting line position i with:

$$\arg\min_i \left(\frac{M_i^s + \delta}{M_i^l + \delta}\right)\left(\frac{A_i^s}{A_i^l}\right)$$

M_i^s and M_i^l denote respectively the average importance over respectively the smaller and the larger divisions of the cell, and δ is a small constant to prevent from divide by zero. A_i^s and A_i^l are the respective areas of the smaller and larger divisions of the cell. Cells that are larger than a maximum allowed size are also subdivided (Fig. 1.5). Having selected stroke scale and position using this process, painting proceeds as per subsection 1.2.3.1.

Fig. 1.4 Hertzmann's multi-resolution curved brush approach [12]. From top to bottom, left to right. The source image, three successive layers of the painterly rendering process demonstrating the over-painting, and the final rendering.

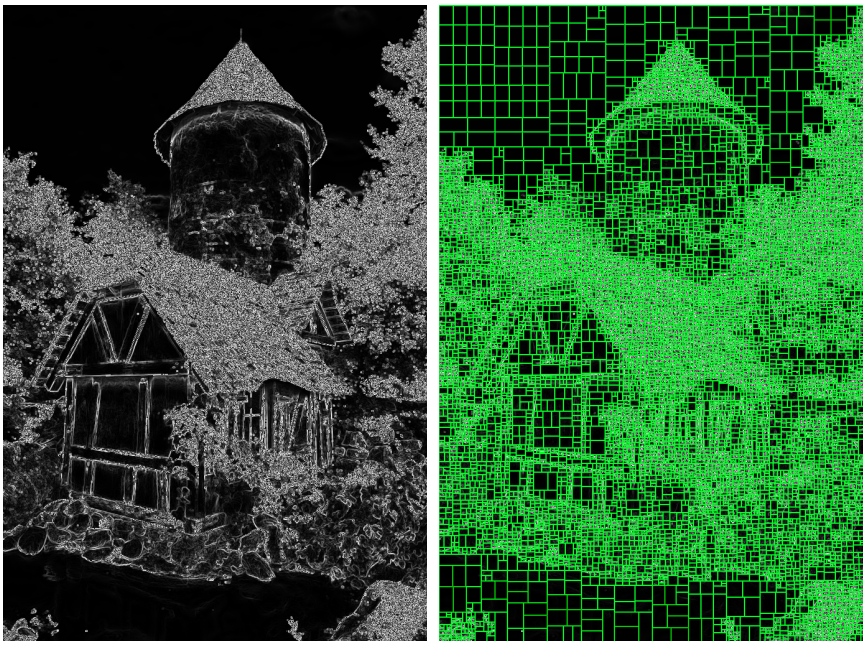

Fig. 1.5 Grids computed using Huang's approach [15]. Importance map (left) and the grid derived from the importance map (middle) and maximum cell size constraints. The input image as per Fig. 1.4.

1.2.4 Transformations on the Source Image

In the algorithms described so far, the raw source image is used to drive the rendering process. However pre-filtering the source image can improve the painterly result. Basic filters that remove noise and small color variations in the image are good candidates for such a pre-process. Such noise can trigger the generation of strokes with spurious color or inappropriate size. Chapter 5 describes a variety of filters such as Gaussian blur, the bilateral filter and morphological operators that are well suited to filter the source images of a SBR system.

Other transformations include color shift. Zhao and Zhu [27] propose to boost color saturation proportional to the underlying importance map computed for the source image. The SBR process of point sampling color, and generating marks of greater size than one pixel, may be regarding as an integration or low-pass filtering process. Such a process has a propensity to wash out colors, and so this process can help emphasize important details in the final rendering.

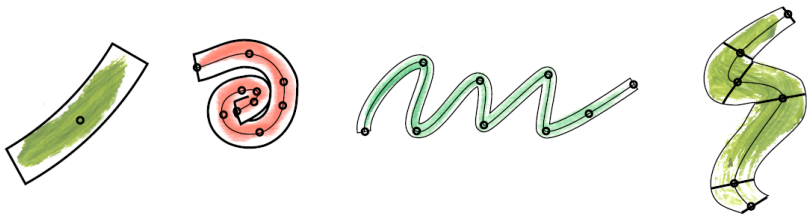

Fig. 1.6 Example spline strokes textured along a quad-strip defined from the curved stroke path.

1.2.5 Texturing Spline Strokes

The curved brush strokes formed by stroke growth algorithm, such as variants of Hertzmann's algorithm [12], are interpolated with smooth (C^1) continuity using a β-spline or Catmull Rom spline [7]. Chapter 2 describes a number of brush models, of increasing sophistication, that may be swept along this curved path to emulate a variety of media types. However in many SBR implementations a simple texture mapping suffices to produce a reasonable aesthetic.

As the piecewise cubic spline of the stroke is commonly expressed in a parametric form, it is trivial to compute a normalized local coordinate system (u,v) for the purpose of texture mapping. The u coordinate of the texture map spans the total path of the curve, and the v coordinate is expressed along the normal. Note that u should be an arc-length parameterization to ensure smooth texture mapping. The direction of the normal can be obtained via the second derivative of the curve at any point. The spatial extent of the v axis is constant, and relates directly to the stroke radius; i.e. the radius of a circle that would cover the stroke's footprint if swept along the curved path of the stroke. This radius is half the stroke's apparent width when rendered.

Sampling (u,v) at regular intervals yields a simple quad-strip sweeping along the path. A triangular strip may be similarly constructed. Either forms the basis for texture mapping. Usually the texture applied is a digitally scanned brush stroke, with an alpha mask to enable overlapping strokes to be seamlessly rendered. In addition to regular texture mapping, it is also possible to introduce a bump map texture onto the same coordinate system usually with minimal additional implementation complexity in the graphics library (e.g. OpenGL). Figure 1.6 provides some examples of textured strokes.

1.3 Global Optimization for SBR

The algorithms discussed so far place strokes in a local greedy manner according to a set of heuristics. These heuristics encourage aesthetically beneficial behavior, such as painting fine details last with small strokes, or aligning strokes tangential to edges. By designing such heuristics within the algorithm, we aim to guide the output emerging from the process towards a desired aesthetic. However, because rendering decisions are made independently on a per-stroke basis, it may be difficult to ensure the generation of that aesthetic.

An alternative approach, discussed in this section, is to consider rendering as an optimization (search) problem. In SBR our final output is a visualization of an ordered list of strokes (and associated attributes). Finding the rendering that is in some sense exhibits the "optimal" or intended aesthetic for a given image, is equivalent to finding the correct sequence of stroke configurations. Quantitatively assessing the optimality of an artistic rendering is neither a trivial nor even a well-defined task, as any art critic might attest! Nevertheless, given a source image, a number of low-level criteria may be expressed by which we may judge the quality of the resulting artistic stylization. For example, measuring the degree to which important details in the source image and depicted within the painting. Or ensuring that where possible, long expressive strokes are used with minimal over-painting.

Defining the optimality of a rendering (i.e. stroke configuration) is one hurdle to overcome when tackling SBR as an optimization problem. The other is to decide how to perform the global optimization over all strokes; a very high dimensional search space. We now discuss two early approaches to painterly rendering that propose solutions to these key problems.

1.3.1 Paint by Relaxation

Although optimization based painting was suggested as early as Haeberli's seminal paper [8], the first algorithmic solution did not appear in the literature until almost ten years later. Hertzmann proposed a *paint by relaxation* process [13] in which his curved stroke algorithm (Section 1.2.3) was used to create an initial painting. The strokes comprising that painting were then iteratively modified; either strokes were added or deleted, or strokes were moved to better positions, in order to maximize an "energy" function.

Hertzmann's innovation was to treat each stroke as an active contour or *snake*. Snakes are commonly used in Computer Vision to fit curves to edges in an image. Like brush strokes under Hertzmann's curved brush model, snakes are also typically represented by a piecewise cubic spline whose position is determined by a set of control points. When a snake is fitted to a edge, its control points are iteratively updated to minimize an energy function comprising internal and external terms. This process is known as *relaxation*. The internal term sums the magnitude of the first and second derivatives over the snake. With respect to these quantities, the

snake will have high energy if its control points are irregularly spaced or it exhibits high curvature. The external term measures the evidence for an edge based on the portion of the image the snake is currently positioned over. A high value reflects poor alignment with an edge.

Hertzmann adapted this relaxation idea, replacing the classical snake energy function with his own weighted sum of terms. Given a source image $S(x,y)$ and an output painting $O(x,y)$, the energy a particular painting (P) comprising a set of S strokes is:

$$E_{app}(P) = \omega_1 \sum_{x=1}^{\text{width}} \sum_{y=1}^{\text{height}} |P(x,y) - M(x,y)| \tag{1.3}$$

$$E_{area}(P) = \omega_2 \sum_{S \in P} \text{Area}(S) \tag{1.4}$$

$$E_{nstr}(P) = \omega_3 \cdot |S| (\text{in } P) \tag{1.5}$$

$$E_{cov}(P) = \omega_4 \cdot (\text{unpainted pixels in } P) \tag{1.6}$$

The weights $\omega_{1..4}$ control the influence of each quality attribute and are determined empirically. The first term E_{app} is a function of the appropriates of strokes, based on the difference of vector functions $P(x,y)$ and $G(x,y)$. These functions encode the RGB pixel color in the painting and source photograph, respectively. The other subscripted energy terms $E_{...}(P)$ refer to area of strokes (*app*), number of strokes (*nstr*) and coverage of the canvas (*cov*). Expression $S \in P$ refers to all strokes comprising painting P. Summing the areas of strokes in Equation 1.4 yields a value analogous to the quantity of paint used in the painting. During optimization, the *minimum* of this energy function is sought. A similar model of stroke redundancy was proposed contemporaneously in the global approach of Szirányi et al. [22], though using an alternative form of optimization (Monte Carlo Markov Chain).

The optimization adopted by Hertzmann is an adaptation of Amini et al.'s snake relaxation process [1]. This is a dynamic programming based algorithm that efficiently scans the pixel neighborhood of each control point on the curve, and identified the "move" for a particular control point that will best minimize the energy function. In addition, each stroke is visited in a randomized order to determine in the energy function would be better minimized if the stroke were deleted. A similarly stochastic stroke addition process is also incorporated; further details are given in [13].

Hertzmann's optimization technique dramatically improves stroke accuracy and retention of detail versus the local greedy approach described in Section 1.2.3. The technique has a secondary benefit in that a painting optimized for a given image, may be optimized over a slightly different image without incurring major changes to the constituent strokes. This stability to change enables coherent video stylization; the current frame is optimized using the previous frame as an initialization. However the drawback of the approach is the significant computational expense of the optimization.

Fig. 1.7 Global optimization using a Genetic Algorithm. Left: Source image, and a painterly rendering produced using Litwinowicz' method [17] based on intensity gradient; all fine details are emphasized in the painting. Right: A rendering using the salience adaptive scheme of Collomosse et al., with close-up on the sign (region A). The non-salient trees have been suppressed, but salient detail on the sign is emphasized. In region B the salience map has been artificially suppressed to illustrate the abstraction effect.

1.3.2 Perceptually based Painting

Paintings are abstractions of photorealistic scenes in which salient elements are emphasized. Artists commonly paint to capture the structure and elements of the scene that they consider to be important; remaining detail is abstracted away in some differential style or level of detail. For example, an artist would not depict every leaf on a tree, or brick in a wall, present in the background of a composition.

The painterly rendering algorithms described so far are guided by intensity gradient magnitude ($|\nabla I(x,y)|$), or similar statistical measures that exhibit responses to high spatial frequencies in the image. They therefore have a tendency to emphasize all high frequency content in the image, rather than the perceptually salient visual content that an artist might depict. To produce renderings that ostensibly represent artwork, it is often necessary to manually doctor this field to reduce fidelity in the painting and enhance the composition [13].

Collomosse et al. proposed a global optimization technique using a Genetic Algorithm (GA) to search the space of possible paintings, so locating the optimal painting for a given source image [6]. Their optimality criterion measures the strength of correlation between the level of detail in a painting and the *salience map* of its source image. A salience map (sometimes referred to as importance map) is an automatically computed 2D field that encodes the perceptual significance of regions within an image. Their optimization technique builds upon an earlier local greedy approach to painting using salience maps [3].

Like most SBR approaches, the technique builds upon Haeberli's abstraction of a painting; an ordered list of strokes [8] (comprising control points, thickness, etc. with color as a data dependent function of these). Under this representation the space of possible paintings for a given source image is very high dimensional, and the aforementioned optimality criterion makes this space extremely turbulent. Stochastic searches that model evolutionary processes, such as GAs, are reasonable search

strategies in such situations; large regions of problem space can be covered quickly, and local minima more likely to be avoided [14]. Furthermore the GA approach adopted allows different regions within a painting to be optimized independently, and later combined to produce improved solutions in later generations.

The optimization proceeds as follows. First, a population of several hundred paintings are initialized from the source image using a stochastic variant of Hertzmann's curve stroke algorithm [12] Section 1.2.3. The curved strokes are created by hopping between pixels, with direction sampled from a Gaussian normal variate center upon $\Theta(x,y)$. Brush stroke length and order are also stochastically. The random variates introduced into the stroke generation process generate a diverse population of possible paintings, some of which are better than others.

The entire population is rendered, and edge maps of each painting are produced by convolution with Gaussian derivatives, which serve as a quantitative measure of local fine detail. The generated maps are then compared to a precomputed salience map of the source image. The mean squared error (MSE) between maps is used as the basis for evaluating the fitness quality $F(.)$ of a particular painting; the lower the MSE, the better the painting:

$$F(I,\psi) = 1 - \frac{1}{N}\sum |S(I) - E(\Psi(I,\psi))|^2 \tag{1.7}$$

The summation is computed over all N pixels in source image I. $\Psi(.)$ is our painterly process, which produces a rendering from I and a particular ordered list of strokes ψ corresponding to an individual in the population. Function $S(.)$ signifies the salience mapping process described in subsection 2.2.1, and $E(.)$ the process of convolution with Gaussian derivatives.

The population is evaluated according to equation (1.7) and individuals are ranked according to fitness. The bottom 10% are culled, and the best 10% of the population pass to the next generation. The middle 80% are used to produce the remainder of the next generation — two individuals are selected stochastically using roulette wheel selection. These individuals are bred via genetic crossover, and subjected to genetic mutation, to produce a novel offspring for the successive generation. Figure 1.8 demonstrates the results of the optimization, showing salient detail being emphasized and non-salient detail being abstracted away over 70 iterations of the optimization. Figure 1.7 shows the abstraction effect of the salience map versus a process drive by intensity gradient.

A further contribution of Collomosse et al.'s system was a user-trainable measure of image salience [10], recognizing the inherently subjective nature of image important. This simultaneously measured salience and classified artifacts into categories such as corner, edge, ridge and so on. This information could also be harnessed to lay down different styles of stroke to depict different image artifacts.

Fig. 1.8 Three iterations of refinement in the Genetic Painting process of Collomosse et al.. From left to right, 1st iteration, 30th iteration and 70th iteration. Images courtesy of Collomosse et al. [6], Springer.

1.4 Creative Control

Early artistic stylization papers broached the topic of impersonating the human artist, or seeking to pass an "artistic Turing Test". However the major of contemporary techniques tend to now motivate their goal as offering a creative tool for use by artists, rather than removing the artist from the loop completely. An interesting question is then the degree of control a creative has over the artistic rendering process, and how this is expressed.

User control is a trade-off between efficiency, freedom of expression and ease of use, i.e. a system provide a way to obtain the user's desired aesthetic whilst lessening tedious, automatable tasks.

Currently most artistic rendering systems enable the user to influence the "style" of presentation; using the terminology of Willats and Durand [24]. Willats and Durand refer to the rendering process as a mapping between a scene – a reference image in our case and a set of marks with attributes (color, color gradient, relative depth, etc.). The link between the scene and the marks is the style. Other higher level aspects of the scene such as geometry and composition are typically not manipulated, though there has been some progress in this area [4, 27].

One may consider this control over style to be expressed at either a low, medium of high semantic level.

1.4.1 Low-level Control

The lowest level of SBR style control is the modification of individual parameters that govern the heuristics on the stroke placement algorithm itself. For example, the thresholds on maximum stroke length during Hertzmann's stroke growth [12], window sizes in Shiraishi and Yamaguchi's method [20], the levels of low-pass filtering in a multi-resolution method, or coefficients on a color transfer function. For an expert user these offer precise control. However the final rendering is an emergent property of a complex process incorporating many such parameters. This makes them counter-intuitive for non-expert users to set.

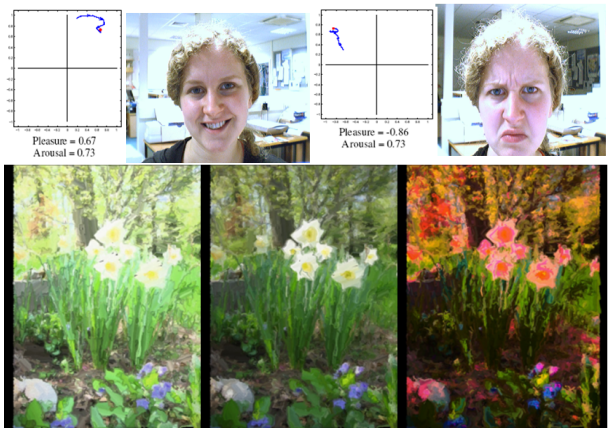

Fig. 1.9 High-level style control: a face picture is analyzed to define a mood. The mood is linked to middle level strokes attributes to produce the painting using the technique of Shugrina *et al* [21]. Image courtesy John Collomosse.

1.4.2 Mid-level Control

Mid-level control enables algorithm parameters to be set at the level of the desired style, for example specifying presets for algorithm parameters that are known to generate output resembling a particular artistic genre. Hertzmann documents four such presets for his curved brush algorithm [12], covering expressionist, impressionist, pointillism and colorist wash. It is also possible to define regions of an image containing specific parameter presets. This is a broader definition of the behavior seen in the "paint by optimization" approach, where differing levels of emphasis may defined for different regions in the image.

Zeng et al. [25] presented a region-based painterly rendering algorithm that splits the scene into a tree of regions, each region being classified by its semantic content. This allowed the users to select different styles for human skin, buildings, vegetation or sky. They also use the decomposition to compute the orientation of strokes. Region based control over stroke orientation is also a feature of interactive painting environments such as AniPaint [18].

1.4.3 High-level Control

High level control is goal directed, enabling intuitive control at a semantic level. A user might desire a "bright, cheerful" or "dark, gloomy" painting in a particular style, but not have the experience to select the individual parameters necessary to create the effect. This kind of control was first demonstrated by Shugrina et al. in the context of an interactive painting, where the user's facial expression was used

to estimate user state within a 2D emotional space. A mapping was established between this space, and a higher dimensional parameter space of an impasto oil painterly algorithm [21] (Figure 1.9). Another option is to let the painterly rendering system provides some potential paintings and let the user rate them [2]. Then new paintings are generated taking into account the score of each candidates, using an interactive evolutionary algorithm.

1.5 Discussion

We have documented the development of SBR from the semi-automated paint systems of the early nineties, to automated processes driven by low-level image measures (such as image gradient) and higher level measures such as salience maps.

Many forms of visual art are created through the manual composition of rendering marks, such as brush strokes. Stroke based rendering (SBR) has therefore become a foundation of many early and contemporary automated artistic rendering algorithms. Many of the subsequent chapters in the first part of this book describe more sophisticated algorithms that build upon the SBR concepts presented here.

SBR focuses only on the lowest level of stylization; that of placing individual marks (strokes) in a greedy or global manner. There are many other considerations in the creation artwork, for example geometry, layout and scene composition that require analysis at a higher conceptual level [19, 24].

Due to their reliance on structural information in the image, a higher level of spatial of temporal analysis is required to access these styles [5]. For example, in the second part of this book we see techniques analyzing the image at the level of the region or performing domain specialized structural analysis (e.g. for portrait painting). Nevertheless these algorithms often incorporate some form of SBR in the final presentation of the processed scene elements.

References

1. Amini, A., Weymouth, T., Jain, T.: Using dynamic programming for solving variational problems in computer vision. IEEE Transaction of Pattern Analysis and Machine Intelligence (PAMI) **9**(12) (1990)
2. Collomosse, J.: Supervised genetic search for parameter selection in painterly rendering. In: Proceedings EvoMUSART (LNCS), vol. 3907, pp. 599–610. Springer (2006)
3. Collomosse, J., Hall, P.M.: Painterly rendering using image salience. In: Proc. Eurographics UK, pp. 122–128 (2002)
4. Collomosse, J., Hall, P.M.: Cubist style rendering from photographs. IEEE Trans. Vis. Comput. Graphics **4**(9), 443–453 (2003)
5. Collomosse, J.P.: Higher level techniques for the artistic rendering of images and video. Ph.D. thesis, University of Bath, UK (2004)
6. Collomosse, J.P., Hall, P.M.: Genetic Paint: A Search for Salient Paintings. In: Proceedings of EvoWorkshops, *Lecture Notes in Computer Science*, vol. 3449, pp. 437–447. Springer (2005)

7. Farin, G.: Curves and surfaces for CAGD: a practical guide, 5th edn. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2002)
8. Haeberli, P.E.: Paint By Numbers: Abstract Image Representations. In: Computer Graphics (Proceedings of SIGGRAPH 90), pp. 207–214 (1990)
9. Haggerty, P.: Almost automatic computer painting. IEEE Comput. Graph. Appl. **11**(6), 11–12 (1991)
10. Hall, P.M., Owen, M.J., Collomosse, J.P.: A trainable low-level feature detector. In: Proceedings Intl. Conference on Pattern Recognition (ICPR), vol. 1, pp. 708–711. IEEE (2004)
11. Hays, J., Essa, I.: Image and video based painterly animation. In: Proc. NPAR, pp. 113–120 (2004)
12. Hertzmann, A.: Painterly Rendering with Curved Brush Strokes of Multiple Sizes. In: Proceedings of SIGGRAPH 98, Computer Graphics Proceedings, Annual Conference Series, pp. 453–460 (1998)
13. Hertzmann, A., Perlin, K.: Painterly Rendering for Video and Interaction. In: NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering, pp. 7–12 (2000)
14. Holland, J.: Adaptation in Natural and Artificical Systems. An introductory analysis with applications to biology, control and artificial intelligence. Univ. Michigan Press (1975)
15. Huang, H., Fu, T.N., Li, C.F.: Painterly rendering with content-dependent natural paint strokes. Vis. Comput. **27**(9), 861–871 (2011). DOI 10.1007/s00371-011-0596-5. URL `http://dx.doi.org/10.1007/s00371-011-0596-5`
16. Kagaya, M., Brendel, W., Deng, Q., Kesterson, T., Todorovic, S., Neill, P.J., Zhang, E.: Video painting with space-time-varying style parameters. IEEE Trans. Vis. Comput. Graphics **17**(1), 74–87 (2011)
17. Litwinowicz, P.: Processing Images and Video for an Impressionist Effect. In: Proceedings of SIGGRAPH 97, Computer Graphics Proceedings, Annual Conference Series, pp. 407–414 (1997)
18. O'Donovan, P., Hertzmann, A.: AniPaint: Interactive painterly animation from video. IEEE Trans. Vis. Comput. Graphics **18**(3), 475–487 (2012)
19. Santella, A., DeCarlo, D.: Visual interest and NPR: An evaluation and manifesto. In: Proc. NPAR, pp. 71–150 (2004)
20. Shiraishi, M., Yamaguchi, Y.: An Algorithm for Automatic Painterly Rendering based on Local Source Image Approximation. In: NPAR 2000 : First International Symposium on Non Photorealistic Animation and Rendering, pp. 53–58 (2000)
21. Shugrina, M., Betke, M., Collomosse, J.P.: Empathic Painting: Interactive stylization through observed emotional state. In: NPAR 06, pp. 87–96. ACM Press, New York, NY, USA (2006). DOI dx.doi.org/10.1145/1124728.1124744
22. Szirányi, T., Tóth, Z., Figueiredo, M., Zerubia, J., Jain, A.: Optimization of paintbrush rendering of images by dynamic MCMC methods. In: Proc. EMMCVPR, pp. 201–215 (2001)
23. Treavett, S.M.F., Chen, M.: Statistical techniques for the automated synthesis of non-photorealistic images. In: Proc. EGUK, pp. 201–210 (1997)
24. Willats, J., Durand, F.: Defining Pictorial Style: Lessons from Linguistics and Computer Graphics. Axiomathes **15**(3), 319–351 (2005)
25. Zeng, K., Zhao, M., Xiong, C., Zhu, S.C.: From Image Parsing to Painterly Rendering. ACM Transactions on Graphics **29**(1), 2:1–2:11 (2009)
26. Zhang, E., Hays, J., Turk, G.: Interactive Tensor Field Design and Visualization on Surfaces. IEEE Transactions on Visualization and Computer Graphics **13**(1), 94–107 (2007)
27. Zhao, M., Zhu, S.C.: Sisley the Abstract Painter. In: NPAR '10: Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering, pp. 99–107. ACM, New York, NY, USA (2010). DOI http://doi.acm.org/10.1145/1809939.1809951