Painterly Rendering with Curved Brush Strokes of Multiple Sizes

Aaron Hertzmann
Media Research Laboratory
Department of Computer Science
New York University

ABSTRACT

We present a new method for creating an image with a hand-painted appearance from a photograph, and a new approach to designing styles of illustration. We "paint" an image with a series of spline brush strokes. Brush strokes are chosen to match colors in a source image. A painting is built up in a series of layers, starting with a rough sketch drawn with a large brush. The sketch is painted over with progressively smaller brushes, but only in areas where the sketch differs from the blurred source image. Thus, visual emphasis in the painting corresponds roughly to the spatial energy present in the source image. We demonstrate a technique for painting with long, curved brush strokes, aligned to normals of image gradients. Thus we begin to explore the expressive quality of complex brush strokes.

Rather than process images with a single manner of painting, we present a framework for describing a wide range of visual styles. A style is described as an intuitive set of parameters to the painting algorithm that a designer can adjust to vary the style of painting. We show examples of images rendered with different styles, and discuss long-term goals for expressive rendering styles as a general-purpose design tool for artists and animators.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation — Display algorithms

Additional Keywords: Non-photorealistic rendering

1. INTRODUCTION

Art and illustration have historically been the sole domain of artists — skilled and creative individuals willing to devote considerable time and resources to the creation of images. Computer technology now allows the quick and easy creation of highly realistic images of natural and imaginary scenes. This technology automates the tedious details of photorealistic rendering, although the process is still driven by the human user, who selects the scene and rendering parameters. The technology for producing non-photorealistic works such as paintings and drawings is less advanced — the user must either "paint" the entire image interactively with a paint program, or else must process an image or 3D model through a narrowly-defined set of non-photorealistic filters. Ideally, a human user should be able to choose from a wide range of visual styles, while leaving the mechanical details of image creation to a computer. It is now possible to envision animating a feature-length movie in a watercolor or oil painting style, a feat that would be prohibitively labor-intensive with traditional media. Non-photorealistic rendering can also be used to inexpensively create attractive and concise images for graphic design and illustration.

Most current computer painterly rendering algorithms use very simple brush strokes that are all of equal size and shape. Thus, the resulting images tend to appear mechanical in comparison to hand-made work. In this paper, we present techniques for painting an image with multiple brush sizes, and for painting with long, curved brush strokes. We find the resulting images to be more visually pleasing and "natural" than those produced with previous algorithms.

Artists have long exploited the richness of natural media in a variety of unique styles. Naturally, we would like our computer algorithms to be capable of similar variety. Here we do not attempt to imbue "creativity" into the algorithms, but prefer a more cooperative relationship. Rather, the user selects a composition and a rendering style, and the computer produces an image from these choices. In this paper we show how to create rendering styles suitable for use by a human designer.

1.1 Related work

Two principal challenges face the production of satisfying non-photorealistic images. The first of these, physical simulation, attempts to closely mimic the physical appearance of real-world artistic media. Impressive systems have been demonstrated for watercolor [5] and a variety of other media [7], in which the user places brush strokes interactively or semi-interactively. Wet media such as watercolor and oil paint are the most challenging media to simulate, because of the complex and rich set of effects produced by fluid flow and transparency. In this paper, we are not concerned with a convincing physical simulation; Haeberli [8] and others have shown that striking compositions can be produced even with very simple painting models. A related area of research is multiresolution painting [2,15], a set of techniques for interactive painting at all scales.

This paper extends the complementary line of research: automatic painting and drawing without human intervention. Cohen [4,13] casts the problem in terms of artificial intelligence; his system, named Aaron, follows a set of randomized rules to create original compositions in a specific style. Aaron even has a robotic painting device. Unlike Cohen's work, we assume that the composition is provided to the system in the form of an input image to be painted. Hence, we can focus on creating a painterly style, and need not deal with the problem of creativity in designing a composition. Winkenbach and Salesin [20,21] describe a system for automatically creating pen-and-ink illustrations from 3D models, and Salisbury et al. [16] describe a technique for

email: *hertzman@mrl.nyu.edu*; URL: *http://www.mrl.nyu.edu/~hertzman*

producing pen-and-ink illustrations from images. Curtis et al. [5] produce watercolor paintings by a semi-automatic algorithm. However, their algorithm does not necessarily produce visible brush strokes, and thus lacks a painterly quality. A common method for processing an image for a painterly effect [1,7,11,14,17,22] is to place a jittered grid of short brush strokes over an image. These brush strokes may be aligned to the normals of image gradients, and all have the same size and shape. (Litwinowicz [11] uses clipped strokes; Treavett and Chen [17] use statistical analysis of the source image to guide stroke size, orientation and placement.) [14] and [17] do vary brush stroke size with respect to local detail levels. They appear to paint each image in a single pass, and thus lack the ability to refine the painting with multiple passes. [14] and [22] allow rendering parameters to be encapsulated and saved as "styles."

1.2 Overview

In the next section, we present a method for painting with different brush sizes to express various levels of detail in an image, and a technique for painting long, curved brush strokes to express continuous color regions in an image. In Section 3, we show how to abstract the rendering process to provide many painting styles. Finally, we discuss some future directions for non-photorealistic rendering.

2. PAINTING TECHNIQUES

2.1 Varying the brush size

Often, an artist will begin a painting as a rough sketch, and go back over the painting with a smaller brush to add detail. While much of the motivation for this technique does not apply to computer algorithms,[1] it also yields desirable visual effects. In Figure 1, note the different character of painting used to depict the blouse, sand, children, boat and sky. Each has been painted with different brush sizes as well as different stroke styles. This variation in stroke quality helps to draw the most attention to the woman and figures, and very little to the ground; in other words, the artist has used fine strokes to draw our attention to fine detail. (Other compositional devices such as shape, contrast and color are also used. These are not addressed in this paper.) To use strokes of the same size for each region would "flatten" the painting; here the artist has chosen to emphasize the figure over the background. In our image processing algorithm, we use fine brush strokes only where necessary to refine the painting, and leave the rest of the painting alone. Our algorithm is similar to a pyramid algorithm [3], in that we start with a coarse approximation to the source image, and add progressive refinements with smaller and smaller brushes.[2]

Our painting algorithm (Figure 2) takes as input a source image and a list of brush sizes. The brush sizes are expressed in radii $\boldsymbol{R}_1 \ldots \boldsymbol{R}_n$. The algorithm then proceeds by painting a series of *layers*, one for each radius, from largest to smallest. The initial canvas is a constant color image.

***Figure 1**: Detail of* At The Seashore, Young Woman having her Hair Combed by her Maid, *Edgar Degas, 1876-7. Note that the small brush strokes are only used in regions of fine detail (such as the children in the background), and draw attention to these regions.*

For each layer $\boldsymbol{R}_i$, we first create a *reference image* by blurring the source image. Blurring is performed by convolution with a Gaussian kernel of standard deviation $f_\sigma \boldsymbol{R}_i$, where f_σ is some constant factor.[3] The reference image represents the image we want to approximate by painting with the current brush size. The idea is to use each brush to capture only details which are at least as large as the brush size. We use a *layer* subroutine to paint a layer with brush $\boldsymbol{R}_i$, based on the reference image. This procedure locates areas of the image that differ from the reference image and covers them with new brush strokes. Areas that match the source image color to within a threshold (T) are left unchanged. The threshold parameter can be increased to produce rougher paintings, or decreased to produce paintings that closely match the source image.

This entire procedure is repeated for each brush stroke size. A pseudocode summary of the painting algorithm follows.

```
function paint(sourceImage, R1 ... Rn)
{
   canvas := a new constant color image

   // paint the canvas
   for each brush radius Ri,
         from largest to smallest do
   {
      // apply Gaussian blur
      referenceImage = sourceImage * G(fσ Ri)
      // paint a layer
      paintLayer(canvas, referenceImage, Ri)
   }

   return canvas
}
```

[1] One motivation is to establish the composition before committing fine details, so that the artist may experiment and adjust the composition.

[2] In fact, our original painting algorithm was based on the Laplacian pyramid: difference images ($\boldsymbol{L}_i$) guided brush stroke placement. However, the difference images assume a perfect reconstruction of the lower levels of the pyramid, and our reconstruction is deliberately imperfect. Thus, refinements at later levels of the pyramid caused unwanted artifacts. Our present algorithm avoids this problem by creating the difference images after every step of the painting.

[3] Non-linear diffusion [19] may be used instead of a Gaussian blur to produce slightly better results near edges. (Figure 5(a)).

Each layer is painted using a simple loop over the image canvas. The approach is adapted from the algorithm described in [11], which placed strokes on a jittered grid. That approach may miss sharp details such as lines and points that pass between grid points. Instead, we search each grid point's neighborhood to find the nearby point with the greatest error, and paint at this location. All strokes for the layer are planned at once before rendering. Then the strokes are rendered in random order to prevent an undesirable appearance of regularity in the brush strokes.

```
procedure paintLayer(canvas,referenceImage, R)
{
   S := a new set of strokes, initially empty

   // create a pointwise difference image
   D := difference(canvas,referenceImage)

   grid := f_g R

   for x=0 to imageWidth stepsize grid do
      for y=0 to imageHeight stepsize grid do
      {
        // sum the error near (x,y)
        M := the region (x-grid/2..x+grid/2,
                         y-grid/2..y+grid/2)

        areaError := ∑_{i,j M} D_{i,j} / grid^2
        if (areaError > T) then
        {
           // find the largest error point
           (x_1,y_1) := arg max_{i,j M} D_{i,j}
           s :=makeStroke(R,x_1,y_1,referenceImage)
           add s to S
        }
      }

   paint all strokes in S on the canvas,
      in random order
}
```

The following formula for color difference is used to create the difference image:[4] $|(r_1,g_1,b_1) - (r_2,g_2,b_2)| = ((r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2)^{1/2}$. In order to cover the canvas with paint, the canvas is initially painted a special "color" *C* such that the difference between *C* and any color is `MAXINT`.

In practice, we avoid the overhead of storing and randomizing a large list of brush strokes by using a Z-buffer. Each stroke is rendered with a random Z value as soon as it is created. The Z-buffer is cleared before each layer.

"`makeStroke()`" in the above code listing is a generic procedure that places a stroke on the canvas beginning at (x_1,y_1), given a reference image and a brush radius. f_g is a constant grid size factor. Following [9], Figure 3(a) shows an image illustrated using a "`makeStroke()`" procedure which simply places a circle of the given radius at (x,y), using the color of the source image at location (x,y). Following [11], Figure 3(b) shows an image illustrated with short brush strokes, aligned to the normals of image gradients.[5] Note the regular stroke appearance. In the next section, we will present an algorithm for placing long, curved brush strokes, closer to what one would find in a typical painting.

Our technique focuses attention on areas of the image containing the most detail (high-frequency information) by placing many small brush strokes in these regions. Areas with little detail are painted only with very large brush strokes. Thus, strokes are appropriate to the level of detail in the source image.

This choice of emphasis assumes that detail areas contain the most "important" visual information. Other choices of emphasis are also possible — for example, emphasizing foreground elements or human figures — but these would require semantic interpretation of the input images, which is known to be an extremely difficult problem in computer vision. The choice of emphasis could also be provided by a human user [16], or as output from a 3D renderer.

2.2 Creating curved brush strokes

Individual brush strokes in a painting can convey shape, texture, overlap, and a variety of other image features. There is often something quite beautiful about a long, curved brush stroke that succinctly expresses a gesture, the curve of an object or the play of light on a surface. To our knowledge, all previous automatic painting systems use a series of small brush strokes, identical aside from color and orientation, or else apply pigment simultaneously to large regions of an image. In contrast, we present a method for painting long, continuous curves. In particular, we focus on painting solid strokes of constant thickness to approximate the coloration of the reference image; exploiting the full expressivity of brush strokes is far beyond the scope of this paper. We model brush strokes as anti-aliased cubic B-splines, each with a given color and thickness. Each stroke is rendered by dragging a circular brush mask along the sweep of the spline.

In our system, we limit brush strokes to constant color, and use image gradients to guide stroke placement. Other authors have also used this concept [11,8,18] for placing strokes. The idea is that the strokes will represent contours of the image with roughly constant color. Our method is to place control points for the curve by following the normal of the gradient. When the color of the stroke deviates from the color under a control point of the curve by more than a specified threshold, the stroke ends at that control point. One can think of this as placing splines to roughly match the isocontours of the reference image.

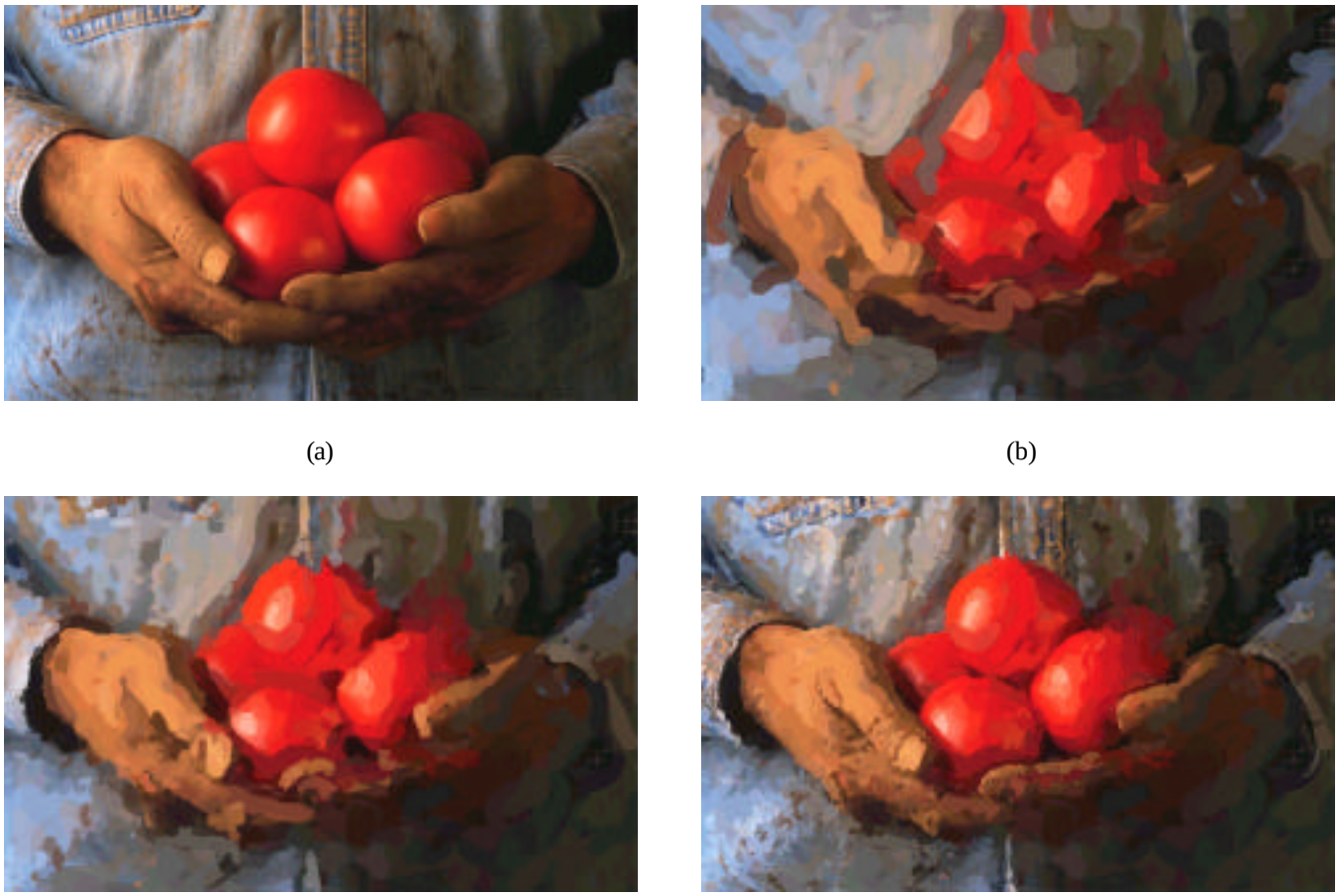
A more detailed explanation of the algorithm follows. The spline placement algorithm begins at a given point in the image (x_0,y_0), with a given a brush radius R. The stroke is represented as a list of control points, a color, and a brush radius. The control point (x_0,y_0) is added to the spline, and the color of the reference image at (x_0,y_0) is used as the color of the spline.

We then need to compute the next point along the curve. The gradient (θ_0) for this point is computed from the Sobel-filtered luminance[6] of the reference image. The next point (x_1,y_1) is placed in the direction $(\theta_0 + \pi/2)$ at a distance *R* from (x_0,y_0) (Figure 4(a)). We use the brush radius *R* as the distance between control points because *R* represents the level of detail we will capture with this brush size. This means that very large brushes create broad sketches of the image, to be later refined with smaller brushes.

[4] We have also experimented with more perceptually correct metrics, such as distance in CIE LUV [6] space. Surprisingly, we found that these often gave worse results.

[5] Note that no stroke clipping is used. Instead, small scale refinements of later layers automatically "fix" the edges of earlier layers.

[6] The luminance of a pixel is computed with $L(r,g,b) = 0.30*r + 0.59*g + 0.11*b$ [6].



(a) (b)

(c) (d)

Figure 2: Painting with three brushes. *(a) A source image. (b) The first layer of a painting, after painting with a circular brush of radius 8. (c) The image after painting with a brush of radius 4. (d) The final image, after painting with a brush of size 2. Note that brush strokes from earlier layers are still visible in the painting.*

(a) (b)

Figure 3: Applying the multiscale algorithm to other types of brush strokes. *Each of these paintings was created with brush strokes of radius 8, 4, and 2. (a) Brush strokes are circles, following [9]. (b) Brush strokes are short, anti-aliased lines placed normal to image gradients, following [11]. The line length is 4 times the brush radius.*

The remaining control points are computed by repeating this process of moving along the image normal to the image gradients and placing control points. The stroke is terminated when (a) the predetermined maximum stroke length is reached, or (b) the color of the stroke differs from the color under the last control point more than it differs from the current painting at that point. The maximum stroke length prevents an infinite loop from occurring. For a point (x_i, y_i), we compute a gradient direction θ_i at that point. Note, however, that there are actually two normal directions, and so two candidates for the next direction: $\theta_i + \pi/2$, and $\theta_i - \pi/2$. We choose the next direction so as to minimize the stroke curvature: we pick the direction $\boldsymbol{D}_i$ so that the angle between $\boldsymbol{D}_i$ and $\boldsymbol{D}_{i-1}$ is less than or equal to $\pi/2$. (Figure 4(b)).

We can also exaggerate or reduce the brush stroke curvature by applying an infinite impulse response filter to the stroke directions. The filter is controlled by a single predetermined filter constant, f_c. Given the previous stroke direction $\boldsymbol{D}'_{i-1} = (dx'_{i-1}, dy'_{i-1})$, and a current stroke direction $\boldsymbol{D}_i = (dx_i, dy_i)$, the filtered stroke direction is $\boldsymbol{D}'_i = f_c \boldsymbol{D}_i + (1-f_c)\boldsymbol{D}'_{i-1} = (f_c\, dx_i + (1-f_c)\, dx'_{i-1},\ f_c\, dy_i + (1-f_c)\, dy'_{i-1})$.

A pseudocode summary of the entire stroke placement procedure follows.

```
function makeSplineStroke(x0,y0,R,refImage)
{
   strokeColor = refImage.color(x0,y0)
   K = a new stroke with radius R
          and color strokeColor
   add point (x0,y0) to K
   (x,y) := (x0,y0)
   (lastDx,lastDy) := (0,0)

   for i=1 to maxStrokeLength do
   {
      if (i > minStrokeLength and
        |refImage.color(x,y)-canvas.color(x,y)|<
        |refImage.color(x,y)-strokeColor|) then
          return K

      // detect vanishing gradient
      if (refImage.gradientMag(x,y) == 0) then
        return K

      // get unit vector of gradient
      (gx,gy) := refImage.gradientDirection(x,y)
      // compute a normal direction
      (dx,dy) := (-gy, gx)

      // if necessary, reverse direction
      if (lastDx * dx + lastDy * dy < 0) then
        (dx,dy) := (-dx, -dy)

      // filter the stroke direction
      (dx,dy) :=fc*(dx,dy)+(1-fc)*(lastDx,lastDy)
      (dx,dy) := (dx,dy)/(dx^2 + dy^2)^1/2
      (x,y) := (x+R*dx, y+R*dy)
      (lastDx,lastDy) := (dx,dy)

      add the point (x,y) to K
   }
   return K
}
```

The minimum stroke length prevents the speckled appearance of very short strokes. To render a curved stroke, the spline is first computed by subdivision. An anti-aliased, circular mask is then drawn along the path of the curve.

We have shown how to draw a long, curved brush stroke, to represent continuous color regions in an image. This method works best in combination with the layering method of Section 2.1; see Figure 2(b) for an example of curved brush strokes without layering. In the future, we would like to enhance this technique to depict other features, such as contours and texture, and to use a richer stroke model, including pressure, bristles, wetness, and tapering.

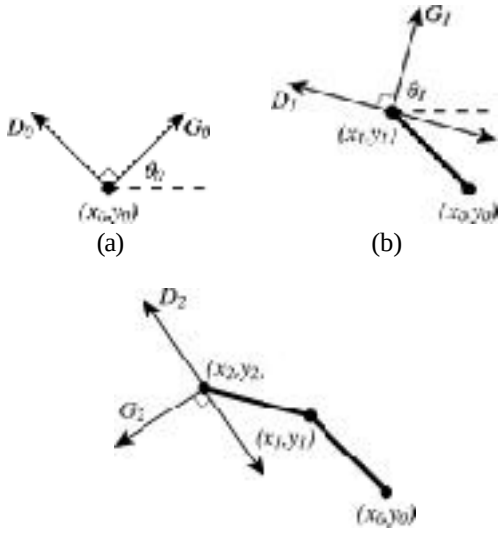


Figure 4: Painting a brush stroke. *(a) A brush stroke begins at a control point (x_0,y_0) and continues in direction $\boldsymbol{D}_0$, normal to the gradient $\boldsymbol{G}_0$. (b) From the second point (x_1,y_1), there are two normal directions to choose from: $\theta_1 + \pi/2$, and $\theta_1 - \pi/2$. We choose $\boldsymbol{D}_1$, in order to reduce the stroke curvature. (c) This procedure is repeated to draw the rest of the stroke. The stroke will be rendered as a cubic B-spline, with the (x_i,y_i) as control points. The distance between control points is equal to the brush radius.*

3. RENDERING STYLES

There is no one "right" algorithm for non-photorealistic rendering, just as there is no "right" approach to painting. We believe that the graphic designer or artist using a rendering system should be allowed to vary the computer's "artistic approach," rather than being forced to employ a single style of painting for every picture. In order to quantify the notion of painterly styles, we propose the use of *style parameters* to control the rendering process. These parameters should provide an intuitive way to vary visual qualities of the painting. Some possible style parameters include stroke curvature and how closely the painting should approximate the original. To be useful to a designer, style parameters should exhibit, as much as possible, the following four properties:

- **Intuitiveness** — Each style parameter should correspond to a visual quality of the painting. These qualities should be intuitive to an artist without any technical computer knowledge.
- **Consistency —** Styles should produce the same "visual character" for different images. For example, we should be able to choose a style based on a single frame of a video sequence, and then render the rest of the sequence in the same style.
- **Robustness** — Each parameter should produce reasonable results over a predetermined range, without "breaking" for some values. A default value should be available, so that extra parameters provide the user with

more options without adding any extra burden. Increasing a parameter should always monotonically increase or decrease some quality of the painting, rather than cause it to fluctuate.

· **Independence** — Style parameters should be independent of one another. Changing line thicknesses, for example, should not affect the saturation of an image.

A group of style parameters describes a space of *styles*; a set of specific values can be encapsulated in a style. Styles may be designed to imitate the styles of famous artists, or may represent other approaches to painting. Styles can be collected into libraries, for later use by designers. Although there may conceivably be hundreds of rendering parameters, the designer need only adjust the parameters appropriate to an application. Some commercial painterly rendering products [14,22] provide the ability to vary rendering parameters and to save sets of parameters as distinct styles.

3.1 Some style parameters

In the experiments that follow, we have used the following style parameters.

· **Approximation threshold** (T) — How closely the painting must approximate the source image. Higher values of this threshold produce "rougher" paintings. (See Section 2.1)

· **Brush sizes** — Rather than requiring the user to provide a list of brush sizes ($\boldsymbol{R}_1 \ldots \boldsymbol{R}_n$), we have found it more useful to use three parameters to specify brush sizes: Smallest brush radius ($\boldsymbol{R}_1$), Number of Brushes (n), and Size Ratio ($\boldsymbol{R}_{i+1}/\boldsymbol{R}_i$). We have found that a limited range of brush sizes often works best. (See Section 2.1)

· **Curvature Filter** (f_c) — Used to limit or exaggerate stroke curvature. (See Section 2.2)

· **Blur Factor** (f_σ) — Controls the size of the blurring kernel. A small blur factor allows more noise in the image, and thus produces a more "impressionistic" image. (See Section 2.1)

· **Minimum and maximum stroke lengths** (minLength, maxLength) — Used to restrict the possible stroke lengths. Very short strokes would be used in a "pointillist" image; long strokes would be used in a more "expressionistic" image. (See Section 2.2)

· **Opacity** (α) — Specifies the paint opacity, between 0 and 1. Lower opacity produces a wash-like effect.

· **Grid size** (f_g) — Controls the spacing of brush strokes. The grid size times the brush radius ($f_g \boldsymbol{R}_i$) produces the step size in the "`paintLayer()`" procedure. (See Section 2.1)

· **Color Jitter** — Factors to randomly add jitter to hue (j_h), saturation (j_s), value (j_v), red (j_r), green (j_g) or blue (j_b) color components. 0 means no random jitter; larger values increase the factor.

The threshold (T) is defined in units of distance in color space. Brush sizes are defined in units of distance; we specify sizes in pixel units, although resolution-independent measures (such as inches or millimeters) would work equally well. Brush length is measured in the number of control points. The remaining parameters are dimensionless.

3.2 Experiments

In this section, we demonstrate four painting styles: "Impressionist," "Expressionist," "Colorist Wash," and "Pointillist." Figure 6 shows the application of the first three of these styles to two different images. The distinct character of each style demonstrates the consistency of the painting algorithm. (Figures 3(f) and 5 are also rendered in the "Impressionist" style.)

Figure 7 shows a continuous transition between the "Pointillist" style and the "Colorist Wash" style. By interpolating style parameter values, we can "interpolate" the visual character of rendering styles. This demonstrates the robustness of the parameters.

The styles are defined as follows.

· **"Impressionist"** — A normal painting style, with no curvature filter, and no random color. T = 100, $\boldsymbol{R}$=(8,4,2), f_c=1, f_s=.5, a=1, f_g=1, minLength=4, maxLength=16

· **"Expressionist"** — Elongated brush strokes. Jitter is added to color value. T = 50, $\boldsymbol{R}$=(8,4,2), f_c=.25, f_s=.5, a=.7, f_g=1, minLength=10, maxLength=16, j_v=.5

· **"Colorist Wash"** — Loose, semi-transparent brush strokes. Random jitter is added to R, G, and B color components. T = 200, $\boldsymbol{R}$=(8,4,2), f_c=1, f_s=.5, a=.5, f_g=1, minLength=4, maxLength=16, j_r=j_g=j_b=.3

· **"Pointillist"** — Densely-placed circles with random hue and saturation. T = 100, $\boldsymbol{R}$=(4,2), f_c=1, f_s=.5, a=1, f_g=.5, minLength=0, maxLength=0, j_v=1, j_h=.3. (This is similar to the Pointillist style provided by [22].)

4. DISCUSSION AND FUTURE WORK

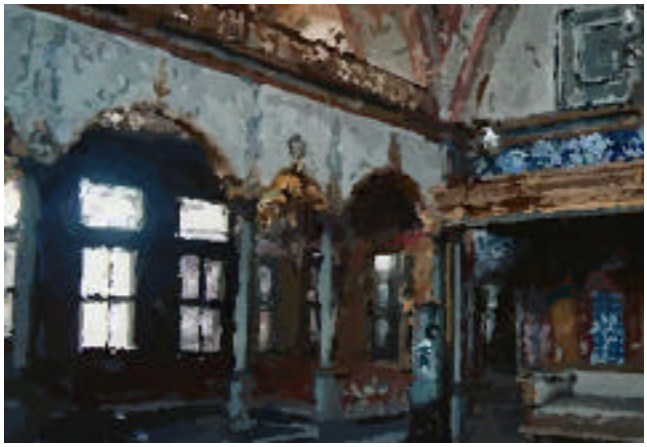
We have presented a new algorithm for producing paintings from images. Brush stroke sizes are selected to convey the level of detail present in the source image using a multiscale algorithm. Long, curved brush strokes are created by moving in a direction normal to image gradients. The painting may be made sketchier or more precise by changing a threshold parameter. Stroke curvature may be limited or exaggerated by filtering stroke direction. These and other parameters describe a space of rendering styles that can be created and modified by artists and graphic designers.

Painting is a complex and rich pursuit, involving many approaches and many ways to interpret a scene. Our goal in developing painting algorithms is similar to a goal pursued by artists: to develop expressive visual languages. Future work in this area should extend the strategies available to non-photorealistic rendering algorithms, both image-based and model-based. We should be able to draw inspiration from various artistic approaches, as well as from computer vision, cognitive science, and artificial intelligence.

Brush strokes may convey many physical properties such as color, texture, lighting, 3D shape, gesture, and overlap, as well as semantic elements such as emphasis, mood, and emotion. One long-term goal is to develop an approach to painting that will convey the "important" features of an image with carefully chosen brush strokes.

A relaxation-based approach [8,18] may also be useful for computer painting. Although relaxation algorithms are usually more compute-intensive than are direct algorithms, they do allow many visual constraints to be embedded into a single energy function, some of which may be difficult to achieve by a direct method.

Another interesting line of work is real-time processing of video [11] and models [12] with different styles. New techniques will be required to maintain temporal coherence for complex brush strokes with various size and shape attributes, while maintaining or changing rendering styles. One can envision a real-time interactive system in which the rendering style varies with the mood or the action.



(a)

(b)

Figure 5: Two "impressionist" paintings.

Acknowledgements

Many thanks to Ken Perlin for useful discussions and support throughout the course of this work. Thanks to Rich Radke, Jon Meyer, and Henning Biermann for discussions. The source images for Figures 5(b) and 7 were provided by Jon Meyer. The source image for Figure 6(b) was used by kind permission of CND, Inc. The author is supported by NSF grant DGE-9454173.

5. REFERENCES

[1] ADOBE SYSTEMS. Adobe Photoshop 4.0

[2] DEBORAH F. BERMAN, JASON T. BARTELL, DAVID H. SALESIN. Multiresolution Painting and Compositing. *SIGGRAPH 94 Conference Proceedings*, pp. 85-90. July 1994.

[3] PETER J. BURT AND EDWARD H. ADELSON. The Laplacian Pyramid as a Compact Image Code. *IEEE Transactions on Communications*. 31:532-540, April 1983.

[4] HAROLD COHEN. The Further Exploits of Aaron, Painter. *Stanford Humanities Review*. Vol. 4, No. 2. pp. 141-158. 1995

[5] CASSIDY J. CURTIS, SEAN E. ANDERSON, JOSHUA E. SEIMS, KURT W. FLEISCHER, DAVID H. SALESIN. Computer-Generated Watercolor. *SIGGRAPH 97 Conference Proceedings*, pp. 421-430. August 1997.

[6] JAMES FOLEY, ANDRIES VAN DAM, STEPHEN FEINER, JOHN HUGHES. *Computer Graphics: Principles and Practice*, Addison-Wesley, 1995.

[7] FRACTAL DESIGN CORPORATION. Fractal Design Painter.

[8] PAUL HAEBERLI. Paint by numbers: Abstract image representations. *Computer Graphics (SIGGRAPH 90 Conference Proceedings)*, 24(4):207-214, August 1990

[9] PAUL HAEBERLI. The Impressionist. http://www.sgi.com/graphica/impression

[10] JOHN LANSDOWN AND SIMON SCHOFIELD. Expressive rendering: A review of nonphotorealistic techniques. *IEEE Computer Graphics and Applications*, 15(3):29-37, May 1995.

[11] PETER LITWINOWICZ. Processing Images and Video for An Impressionist Effect. *SIGGRAPH 97 Conference Proceedings*, pp. 407-414. August 1997.

[12] LEE MARKOSIAN, MICHAEL A. KOWALSKI, SAMUEL J. TRYCHIN, LUBOMIR D. BOURDEV, DANIEL GOLDSTEIN, JOHN F. HUGHES. Real-Time Nonphotorealistic Rendering. *SIGGRAPH 97 Conference Proceedings*, pp. 415-420. August 1997.

[13] PAMELA MCCORDUCK. AARON's CODE: Meta-Art, Artificial Intelligence, and the Work of Harold Cohen. New York: W. H. Freeman & Co. 225 pages. 1991.

[14] MICROSOFT CORPORATION. Microsoft Image Composer 1.5

[15] KEN PERLIN AND LUIZ VELHO. LivePaint: Painting with Procedural Multiscale Textures, *SIGGRAPH 95 Conference Proceedings*, pp. 153-160. 1995.

[16] MICHAEL P. SALISBURY, MICHAEL T. WONG, JOHN F. HUGHES, DAVID H. SALESIN. Orientable Textures for Image-Based Pen-and-Ink Illustration. *SIGGRAPH 97 Conference Proceedings*, pp. 401-406. August 1997.

[17] S. M. F. TREAVETT AND M. CHEN. Statistical Techniques for the Automated Synthesis of Non-Photorealistic Images. *Proc. 15th Eurographics UK Conference*, March 1997.

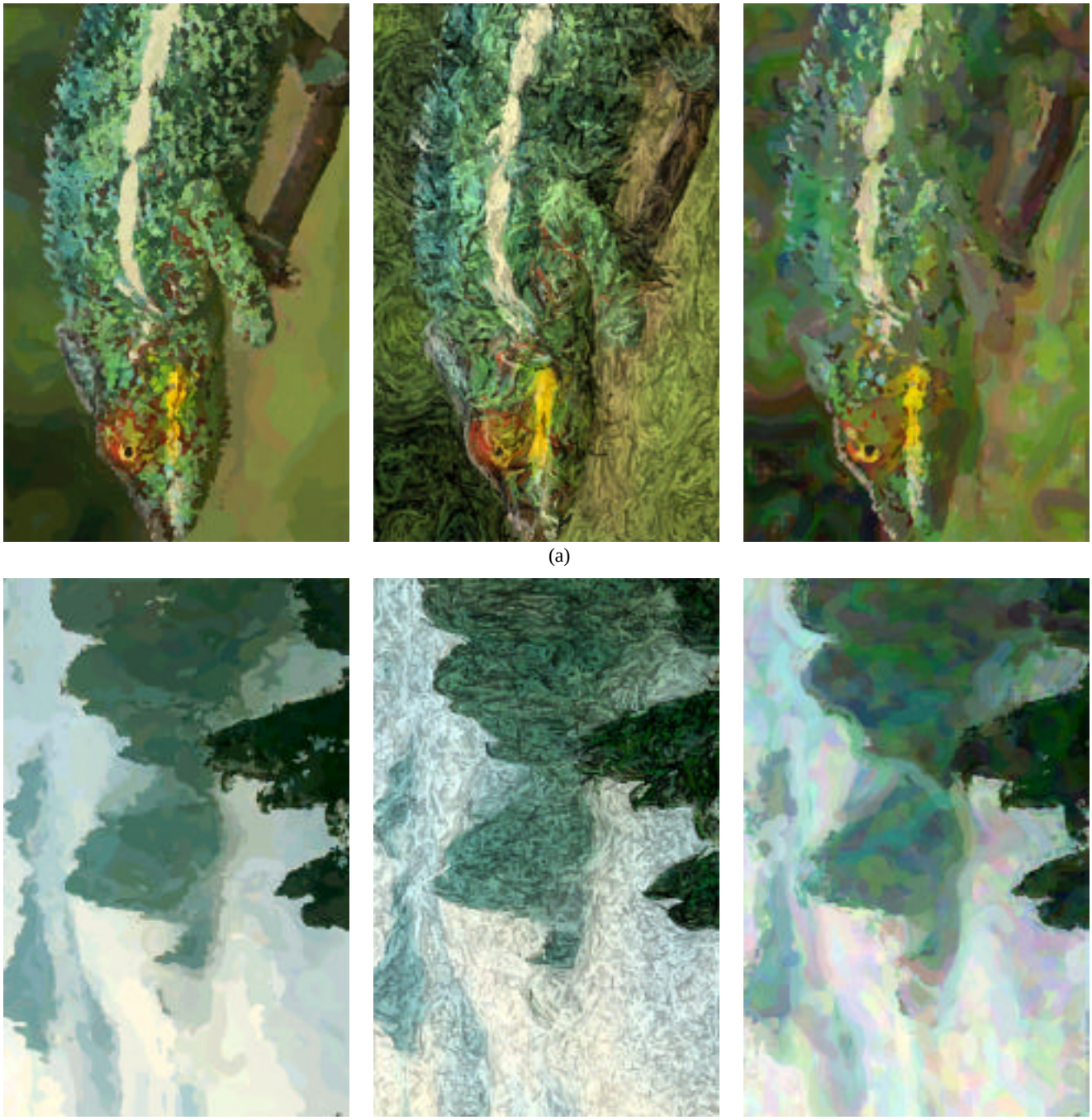
[18] GREG TURK AND DAVID BANKS. Image-Guided Streamline Placement. *SIGGRAPH 96 Conference Proceedings*, pp. 453-460. August 1996.

[19] JOACHIM WEICKERT, BART M. TER HAAR ROMNEY, MAX A. VIERGEVER. Efficient and reliable schemes for nonlinear diffusion filtering. *IEEE Transactions on Image Processing*. March 1998.

[20] GEORGES WINKENBACH AND DAVID H. SALESIN. Computer-Generated Pen-and-Ink Illustration. *SIGGRAPH 94 Conference Proceedings*, pp. 91-100. July 1994.

[21] GEORGES WINKENBACH AND DAVID H. SALESIN. Rendering Parametric Surfaces in Pen and Ink. *SIGGRAPH 96 Conference Proceedings*, pp. 469-476. August 1996.

[22] XAOS TOOLS. Paint Alchemy 2.0



(a)

(b)

Figure 6: Applying different painterly styles*. Left column: "Impressionist." Middle column: "Expressionist." Right column: "Colorist Wash." Note that the styles have a consistent visual appearance when applied to different images.*

(a) (b) (c)

Figure 7: Interpolating rendering styles*. Images (a) and (c) are rendered in the "Colorist Wash" and "Pointillist" styles, respectively. The average of their parameters was used to produce the style for (b). (The number of layers (n) was rounded up to 3.)*