Artistic Vision: Painterly rendering using computer vision techniques.

Bruce Gooch Greg Coombe Peter Shirley

University of Utah Technical Report Number UUCS-00-017

Category: research

Format: print

Contact:	Bruce Gooch Department of Computer Science University of Utah 50 S Central Campus Dr RM 3190 Salt Lake City, UT 84112–9205
phone:	(801) 585–0010
fax:	(801) 581–5843
email:	bgooch@cs.utah.edu

Estimated # of pages: 8

Keywords: painting, skeleton

We present a method that takes a raster image as input and produces a painting-like image composed of strokes rather than pixels. Unlike previous automatic painting methods, we attempt to keep the number of brush-strokes small. This is accomplished by first segmenting the image into features, finding the medial axes points of these features, converting the medial axes points into ordered lists of image tokens, and finally rendering these lists as brush strokes. Our process creates images reminiscent of modern realist painters who often want an abstract or sketchy quality in their work.

Artistic Vision: Painterly rendering using computer vision techniques.

Category: research

Abstract

We present a method that takes a raster image as input and produces a painting-like image composed of strokes rather than pixels. Unlike previous automatic painting methods, we attempt to keep the number of brush-strokes small. This is accomplished by first segmenting the image into features, finding the medial axes points of these features, converting the medial axes points into ordered lists of image tokens, and finally rendering these lists as brush strokes. Our process creates images reminiscent of modern realist painters who often want an abstract or sketchy quality in their work.

CR Categories: I.3.7 [Computing Methodologies]: Computer Graphics—2D Graphics

Keywords: painting, skeleton

1 Introduction

The art of painting relies on representation and abstraction. In "realist" painting, the abstraction occurs when the detail of real images is approximated with limited spatial resolution (brush strokes) and limited chromatic resolution (palette). Economy is a quality of many good paintings, and refers to the use of only those brush strokes and colors needed to convey the essence of a scene. This notion of economy has been elusive for computer-painting algorithms. We explore an automated painting algorithm that attempts to achieve economy, particularly in its use of brush strokes. We also allow the user to iterate with the system to improve the default results. Paintings with economy may be useful for creating real paintings using robots, creating physical painting "replicas".

There are two main tasks involved in the creation of a digital painting. First is the creation of brush stroke positions. The second is the "rendering" of brush strokes into pixel values. If the brush stroke positions are manually created by a user, then this is a classic "paint" program. If the brush stroke positions are computed algorithmically, then this is an "automatic" painting system. In either case, once the brush stroke geometry is known, the brush strokes must then be rendered, usually simulating the physical nature of paint and canvas [4, 16, 22].

The economy of painting is determined when brush stroke paths and widths are created. We present an algorithm that carefully chooses brush stroke parameters in a way that we believe achieves economy. This method is summarized in Figure 1. The digital image is first converted into a set of "tokens" which are mini brush strokes with position, orientation, width, and color. These tokens are then collected into longer stroke-sets. Finally these stroke-sets are each converted into a single brush stroke. We use a variation of standard algorithms to render these brush strokes.

We review previous digital painting strategies in Section 2. We give an overview of our algorithm in Section 3. The conversion from a single segment of an image to a set of planned brush strokes, which is the core of our contribution, is covered in Section 4. We then show some resulting paintings in Section 5, and discuss possible improvements to our method in Section 6.

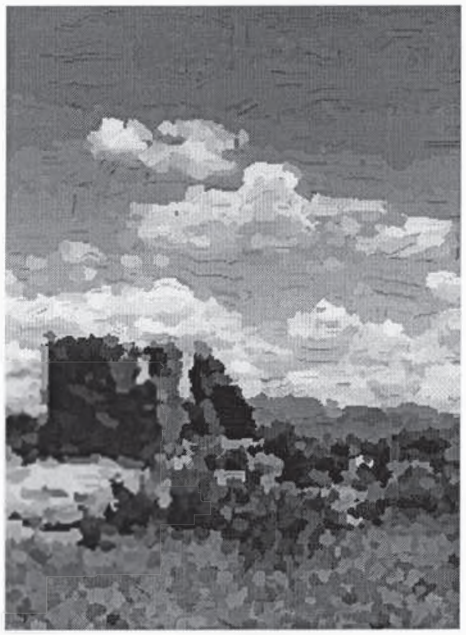

Figure 1: *A landscape painting of Hovenweep National Monument. This painting was made automatically using the system described in this paper and a scanned vacation photograph.*

2 Background

Two basic approaches to digital painting and drawing are used in computer graphics. The first simulates the characteristics of an artistic medium such as canvas and paint. The second attempts to automatically create drawings or paintings by simulating the artistic process. These approaches can be combined as they are dealing with different aspects, one low-level and one high-level, of painting/drawing.

Work intended to simulate artistic mediums can be further divided into those which simulate the physics of making a work of art, and those which simulate the "look and feel" of a particular medium. Strassmann simulated the look of traditional sumi-e painting with polylines and a raster algorithm [22]. Pham augmented this algorithm using b-splines and offset curves instead of a polyline to achieve a smoother brush path [16]. Williams provides a method of merging painting and sculpting by using the raster image as a height field [24].

Smith points out that by using a scale-invariant primitive for a brush stroke, multi-resolution paintings can be made [20]. Berman et al. showed that multi-resolution painting methods are efficient in both speed and storage [1]. Perlin and Velho used multi-resolution procedural textures to create realistic detail at any scale or dimension [15]. Their work emphasizes that digital paintings stored as strokes may be useful for transmitting stylized images across a network.

Several authors have simulated the interaction of paper/canvas and a drawing/painting instrument. Cockshott simulated the substrate, diffusion, and gravity in a physically-based paint system [4]. Curtis et al. modeled fluid flow, absorption, and particle distribution to simulate watercolor [6]. Sousa and Buchanan simulated pencil drawing by modeling the physics and interaction of pencil lead, paper, and blending tools [21].

While the works discussed above are concerned with the low-level interaction of pigment with paper or canvas, other authors aid a user in the creation of an art work, or automate the process altogether. Haeberli built a paint system that re-samples a digital image based on a brush, and then automated this system using a second control image [8]. Wong built a system for charcoal drawing that prompts the user for input at critical stages of the artistic process [25]. Meier produced painterly animations using a particle system [14]. Litwinowicz produced impressionist-style video by re-sampling a digital image and tracking optical flow [13]. Hertzmann refined Haeberli's technique by using progressively smaller brushes to create a hand-painted effect from a photograph automatically [10]. Shira et al [19] use image moments and a subdivision scheme to build digital paintings automatically. Gooch et al. automatically generated technical illustrations from polygonal models of CAD parts [7].

The algorithm described in this paper should be grouped with the latter set of works that simulate the high-level results of the artistic process rather than the physics of the painting process. Our work uses computer vision algorithms to paint an image that is reminiscent of the way an artist might paint it. Our technique results in a resolution-independent list of brush strokes which can be rendered into raster images using any brush stroke rendering method.

3 Algorithm

The steps of the algorithm are as follows (Figure 2):

1. Decompose the images into segments.
2. Decompose each segment into brush strokes.
3. Render brush strokes in some order.

Recent work in computer graphics has shown that a first order approximation to the tone mapping operator should probably be achromatic [18]. Hence source images are segmented based on pixel luminance, and color brush strokes using the color ratios suggested by Schlick [18]. By allowing a user to set the number of intensity thresholds the image will be segmented into. A set of approximately perceptually uniform grey levels is created, and the image is segmented using a flood filling algorithm. A similar technique is sometimes used by human artists when they first produce tonal sketches of the scene they are painting [20].

Each segment is independent; two segments of the same luminance are treated independently (i.e. the letters *T* and *H* in Figure 2). A more sophisticated segmentation strategy could be used without changing the rest of our pipeline.

Brush-stroke path generation is the most complex part of the system. First computer vision algorithms are used to smooth the segmented regions. The system next finds a discrete approximation to the central axis of each segment, called the ridge set, which determines a brush path. Elements of the ridge set are pieced together into tokens. These tokens can, at the users discretion, be spatially sorted into ordered lists. In the final image this second sorting has the effect of painting a region with a single large stroke instead of many small strokes. The system also estimates the "thickness" along the central axis to control brush width. The details of the brush stroke generation are the main contribution of this paper and are covered in the next three sections.

Modified versions of Strassman [22] and Pham's [16] algorithms are used to render brush strokes to the screen. Strassman and Pham modeled sumi-e brushes which taper on and taper off to a point during a brush stroke. I instead choose to model a Filbert brush

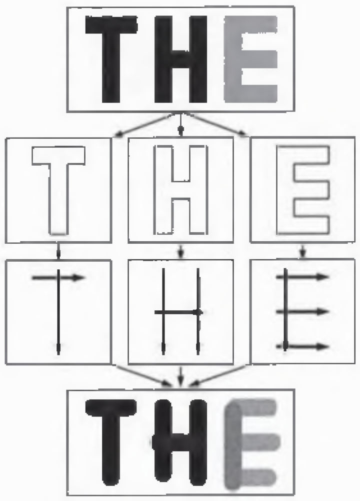

Figure 2: *The basic steps in the algorithm. First the image is segmented using flood filling. Next each segment is independently decomposed into brush strokes using vision algorithms and an approximate medial axis transform. Each brush stroke is then "rendered" into a raster image.*

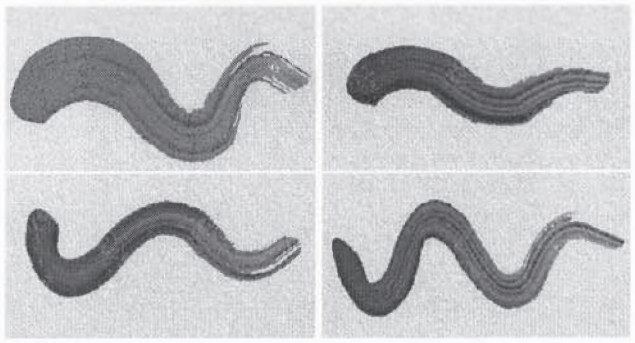

Figure 3: *Digitally simulated brush strokes modeled on a Filbert brush.*

used in oil painting. Filbert brushes are made as round brushes with round tips, and then flattened. They are good all-purpose brushes combining some of the best features of flat and round brushes [20]. To model a Filbert brush, The "taper-on" is constrained to a circular curve and the "taper-off" to a parabolic curve. Examples of this type of simulated brush stroke are shown in Figure 3.

4 Segmentation and smoothing

The image is first segmented based on intensity. Then the segmented regions are filtered using computer vision algorithms to smooth the region edges, and to any holes inthe regions. This process aids in the medial axis computation phase of the process by reducing noise.

4.1 Segmentation

The software user is allowed to set the number of intensity thresholds the image will be segmented into. An array of floats representing these thresholds is created by recursively evaluating the expression:

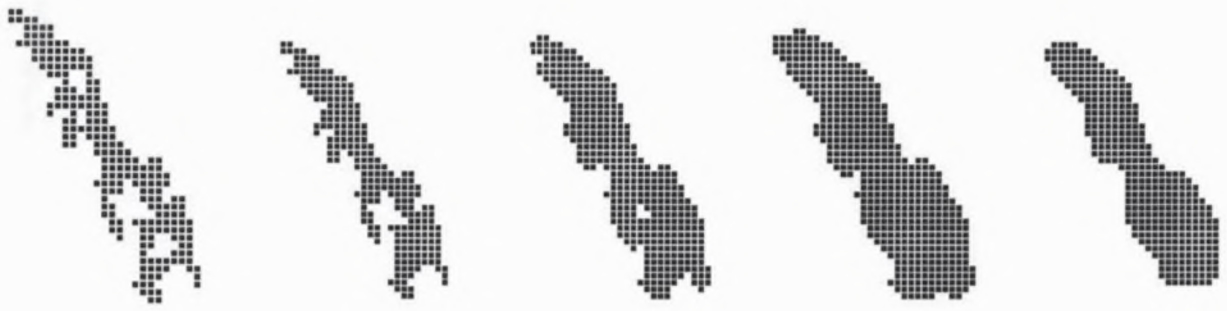

Figure 4: An example of a segmented region. The region after the hole filling algorithm has been run. The region after being grown, the region after having been shrunk.

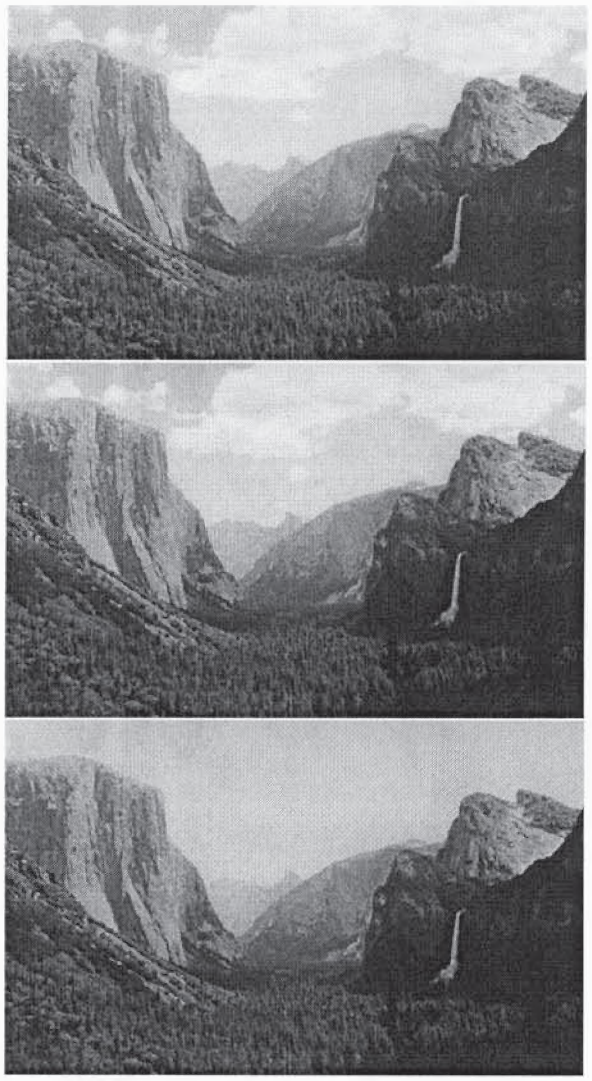

Figure 5: An example of varying the number of gray levels in the segmentation and the resulting images. From top to bottom; the source image, image segmented using 72 gray levels, image segmented using 48 gray levels. Note that the coulds have faded into the sky in the 48 gray level image. The effects of this type of segmentation artifact is reflected in the final painted image.

$$r = \left(\frac{1}{I_0}\right)^{\frac{1}{n}}$$

were r is the current threshold, I_0 is the initial intensity, and n is the number of intensity levels [9].

Pixel values are sorted by intensity into an array. This allows the use of the brightest pixel as a seed to flood fill regions of the image. The flood filling algorithm proceeds by adding the seed pixel to a region list, and setting a head pointer to the beginning of the region list. Next a gray level pointer is set to the first entry of the intensity threshold array. Flood filling begins by checking the neighbors of the pixel pointed to by the head pointer. If any neighbor has a value greater than the value pointed to by the gray level pointer it is added to the region array. When the current neighbors have been checked the head pointer is advanced, if no new data members exist in the region list the process is complete and the region list is returned.

To fill a new region values are popped from the brightest pixel queue, tested to see if they belong to a previously segmented region, and if not used to seed the region. If this new seed pixel is below the current gray level pointer the gray level pointer is adjusted downward.

In practice regions with fewer than twenty five pixels tended to evaporate during processing by the vision algorithms. Therefore if a region has fewer than twenty five pixels the region is recycled by not forming a region with the pixels, then choosing a brightest pixel as if a region had been formed. This allows pixels to be added to another region.

4.2 Hole Filling

The first vision algorithm run on a segmented region is a hole filling algorithm. Segmented regions are stored as boolean arrays with *true* indicating membership of that pixel in the region. Each *false* pixel is queried to find how many true neighbors it has, pixels with more than five *true* neighbors are changed to *true*. Next each *false* pixel is tested for having *true* values above, below, to the right, and to the left of it. If true values are found for each of these the algorithm next checks the pixels form the above value to the below value to make sure that they all have true left and right values. Then from the right value to the left value all the pixels are checked to find if they have *true* above and below values. If all of these values are *true* the pixel is is set to *true*.

4.3 Expanding and Shrinking

Next expanding and shrinking algorithms are used on the region to smooth the boundary of the region and removing isolated noise pixels in the region. First expand the region using the following rule: change a pixel from *false* to *true* if any of its neighbors are *true*. Next shrink the region using the rule: Change a pixel from *true* to *false* if any neighbor is *false*. In practive it was found that two applications of expanding followed by shrinking produces good results. Expanding and shrinking operations may also be applied to the approximate medial axis, described in the next section with good results.

5 Ridge Set Extraction

Once a region is smoothed and in-filled the medial axis (skeleton) is computed and used as a basis for the brush stroke rendering algorithm. The medial axis of a region is essentially the "spine" of an object. For example, the medial axis of a person would roughly be the stick figure associated with that person. The medial axis was first presented by Blum [2], and has been shown to be useful in coarsely approximating 2D [12] and 3D objects [11]. The medial axis has also been shown to be a good approximation of the way the human visual system perceives shape [3].

Previous automatic painting methods use a hierarchy of image grids to segment the source image into strokes. We first attempted to use a similar system but found that the results were highly sensitive to the underlying grid structure. The problems with grid artifacts led us to the medial axis transform. The medial axis transform yields scale and rotation invariant measures for a segment, and is independent of a grid structure. In addition the transform yields width information along the medial axis.

While the medial axis is a continuous representation, there are several types of algorithms for computing the medial axis in image space, including thinning algorithms and distance transforms. The distance transform algorithms are not as sensitive to boundary noise and produce width information, but they tend to produce double lines and often don't preserve the connectedness of the medial axis lines. Thinning algorithms, like Rosenfeld's parallel algorithm [17], preserve the connectedness of components and produce smooth medial axis lines, but are sensitive to noise along the boundary, which produces undesirable spikes (spurs). Although it is possible to filter some of the spurs, filtering often results in a loss important information. Another drawback to thinning algorithms is that they do not produce information about the distance to the boundary, which is needed for brush stroke width estimation in the application.

The positive aspects of both techniques are combined in this thesis to form a hybrid method. First apply the distance transform to extract a discrete approximation of the medial axis called the ridge set. I then thin the ridge set to remove spurs, caused by boundary noise, and double lines, caused when the medial axis falls between pixels. The combination of techniques results in a ridge set with distance information, and reduced sensitivity to noise along the boundary.

5.1 Distance Transform

For each pixel in the image, compute the shortest distance to the boundary using a distance transform [12]. On its first pass the distance transform assigns a value of one to each pixel in the region. Subsequent passes of the distance transform over the region approximate a Euclidean distance transform. By finding a single diagonal zero valued pixel if such a neighbor exists, or the smallest non zero neighbor of the current pixel. If this value is larger than the current value of this pixel one is added to the current value of the pixel in the case of a vertical or horizontal smallest neighbor, and the square root of two is added to the current pixel value in the case of a vertical smallest neighbor. The algorithm proceeds until no values in the region are changed in a pass over the region. Notice in Figure 4.3 that any pixel affects only the next level of pixel values (analogous to the next layer in an onion skin). Since the value at a pixel represents the distance to the boundary, the number of passes over the image is proportional to the radius of the largest circle that touches both boundaries.

5.2 Extract Approximate Medial Axis

This distance transformed region can be tested in a manner that yields a set of ridge points, which are points that are further away from the boundaries than the surrounding points. These ridge points form a discrete approximation to the medial axis. The distance transform also gives us an approximate width at each ridge point. These widths are used as offsets when rendering brush strokes.

A very conservative test is used in the search for a ridge set. Pixels are part of the ridge set only if they are greater than or equal to the value of all of the pixels in their eight neighborhood. This strict test necessitates some of the grouping algorithms performed later by leaving gaps in the ridge set. However, in practice using a less conservative test results in noisy line segments which tend to produce nervous uncontrolled brush strokes. In addition the thinning algorithms tend to work faster, and to produce smoother line segments when presented with sparse ridge sets.

5.3 Thin Axis

To address the problem of double lines in the distance transform, we treat the ridge set as a binary image and run a thinning algorithm over the set. We use Rosenfeld's parallel thinning algorithm [17], which runs over each point in an image and removes the point if it is not 8-simple. A pixel is called 8-simple if it cannot be removed without destroying the 8-connectivity of the set. Rosenfeld's algorithm and the connectivity problems associated with the 8-simple test, as well as the details of our implementation of the algorithm are discussed in the appendix. This thinning algorithm eliminates double-lines and most other noise from the ridge set. The algorithm typically requires 2-3 passes over the binary ridge set data. Another advantage of the thinning algorithm is that points in the ridge set are gauranteed to be at most 3-connected, that is given any point in the ridge set that point has at most three neighbors in its 8-neighborhood.

At the core of Rosenfeld's parallel thinning algorithm is a test for the 8-simpleness of a pixel. We present a fast method for determining whether a pixel neighborhood is 8-simple.

input for each pixel c in image, $N = \{i \mid i \in 8\text{nbd of c},\ i \in S\}$

output boolean simple, not_simple

More completely, let S = set of pixels in the current segment. Adjacency refers to 8-connectedness (pixels on sides or diagonals).

- An *8-neighborhood* is a collection of all pixels that are adjacent to a center pixel.
- $p \in S,\ q \in S$ are *8-connected* if they are adjacent.
- An 8-neighborhood is *8-simple* if $\forall p \in S$, the removal of the center pixel does not change the 8-connectedness of p (i.e. the center pixel is a redundant path).

Construct a graph $G\,(V,\ E)$ as such: let $V = \{v \mid v \in 8\text{-nbd}\}$ and let $E = \{e_{ij} \mid \|i-j\| \leq \sqrt{2}\}$. This is illustrated in Figure 7.

The graph G represents the 8-connectedness paths of the neighborhood. The intersection of our input set N with the graph G results in a new graph, $G'\,(V,\ E)$. Now our test for 8-simpleness just becomes a test for the connectedness of the planar graph G'.

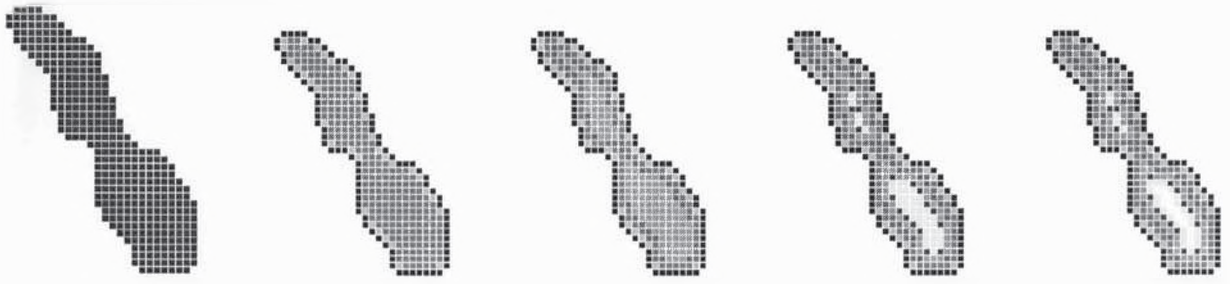
Figure 6: An example of a distance transform on a region. First all pixel values inthte region are initilized to one if they are in the segmented regionn, zero if they are not. Next multiple passes are made over the region. At each pass if a pixels neighborhood is nonzero, and contains values that are all less than or equal to the current value of that pixel a value is adde to the contents of the pixel. This example shows the increasing value of the pixels by changing their color values to warmer values.

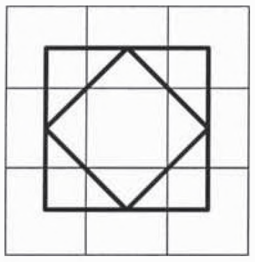
Figure 7: The graph representation of an 8-neighborhood.

This means that we can use Euler's Theorem for connected planar graphs, which states that $v + r - 2 = e$, where v denotes vertices, r denotes regions of plane, e denotes edges. Rearranging the terms yields two conditions for 8-simpleness in a pixel 8-neighborhood: 1) there can be no isolated pixels; 2) $v - 1 \leq e$.

The method for this is to represent $G(V, E)$ as $E_i = \{j \mid \|i - j\| \leq \sqrt{2}\}$. Then, $\forall i \in N$ {

```
    if ( $E_i \cap N = \phi$ ) return not_simple; // isolated pixel
    else $edges$ += degree( $E_i \cap N$ );
}
if ( $edges \geq v - 1$ ) return simple;
else return not_simple
```

The E_i can be encoded in binary, resulting in a fast test. The only storage requirements are the eight sets E_i, which can be stored as eight integers. Most previous thinning algorithms in the computer graphics literature enumerate and store every case, resulting in a large overhead.

6 Ridge Set Tokenizing and Grouping

The combination of the distance transform and thinning algorithms yields a set of approximate medial axis points and a set of width values associated with each point. Next group spatially coherent points into tokens, as shown in Figure 5.3. These tokens can next grouped into strokes, and finally strokes from different segmentation regions may be grouped together.

Tokens are formed by classifying the points in the ridge set based on 8-connectedness of the points with other members of the ridge set. Local searches for connected points are then used to group the points into tokens.

We explored two methods for grouping tokens. The first involves taking the moments of the tokens using the width values as weights at each point. All tokens are then compared, and merged together using a variant of Prim's minimum spanning tree algorithm [5]. The second involves creating a cone of acceptable values at the beginning and end of each token. If two tokens cones intersect the tokens are merged into a single larger token. The methods differ in complexity, speed, and in the manner in which the grouped tokens can be rendered as brush strokes. Both methods are discussed in detail and their merits are compared.

Strokes from different segmentation regions can be grouped using the same methods used for merging tokens. Merging strokes however, comes with a large memory overhead because all of the data structures involved in processing a region must be saved off for later comparison and merging instead of being reused in processing the next segmented region.

6.1 Forming Tokens

The first step in tokenizing the ridge set is to classify the members of the set based on their 8-connectedness with other members of the ridge set. Points are classified as follows; points with no neighbors in their 16-neighborhood (two pixel rings out from the given point) are classified as orphans, points with one neighbor in their 16-neighborhood are classified as seed points, points with two neighbors in their 16-neighborhood are classified as line points, and points with 3 neighbors are classified as branch points. During classification queues for each type of point in the region are instantiated and filled with the given points. In addition a binary array the same size as the ridge set is instantiated and its members set to true. The binary array will be used to find if any member of the ridge set has been added to a token. Orphan points are considered to be tokens with a single point.

Grouping of tokens proceeds as follows. Points are popped from the seed queue and added to token objects if the entry in the binary array corresponding to the popped seed is true, else a new seed is popped. The entry in the binary array corresponding to the seed is set to false. Seeds have only one point in their 16-neighborhood, find this point. If the point is a seed; add the point to the current token, set the binary array position to false, make a new token, pop a new seed from the seed array. If the point is a line point; add the point to the current token, set the position in the binary array to false, find the 16-neighbor of the line point. Note that finding the 16-neighbor is assumed to be dependent on the state of the binary array. If the point is a branch point; add the point to the token object, reclassify the point as a line point, create a new token object and pop a new seed from the seed array.

Now that the ridge set is grouped into tokens as in Figure 5.3 This token set can be grouped into strokes.

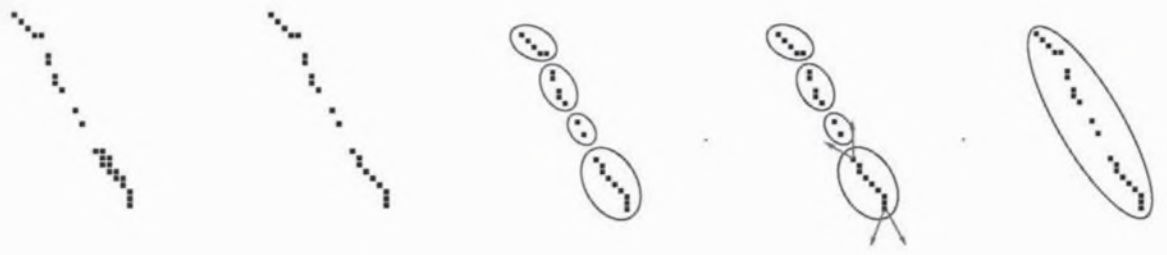

Figure 8: An example of ridge set extraction, thinning and grouping. Frist the ridge set is extracted from the distance transformed image. Next our thinning algorithim is applied. The resulting ridge set is next grouped into tokens, and these tokens are merged into a stroke.

6.2 Grouping Tokens via Moments

The first method we explored for grouping tokens into strokes was to take the moment of the token using the width values associated with the token points as weight values. The first moments yield the center of mass for the token which is used as the position of the token. While the second moments allow us to construct a major and minor axis for the token which is used for the length and width of the token, and an angle of orientation. At this time we also sample the color of the token points from the original image.

The original image is sampled in a rectangular box corresponding to the width value at the given ridge point. Color values are added together and averaged. The color value of orphan tokens are sampled in a similar manner. The color value for the token is then computed by taking a weighted average of all of the points in the token. The average for the token is weighted by the width values of the token points.

Once all of the points are grouped into tokens. The information in the current set of tokens is used to create a set of strokes. This process is a variant of Prim's minimum spanning tree algorithm [5]. A list of edges connecting every pair of tokens that are within a distance tolerance is created. Next an edge cost is computed for each edge by summing the differences of the token properties position, orientation and color. For every token other than the orphan tokens a priority queue is used to select the lowest cost edge e_{ij}. The algorithm then attempts to merge these tokens into a stroke. Strokes are represented as an ordered set of tokens $S = \{t_0, \ldots, t_n\}$. A merge is successful if vectors from the position of both tokens along the major axis of the tokens point toward each others positions and the angle formed by the vectors is less than forty five degrees. This process is repeated by continuing to prioritize and attempt to merge tokens and strokes until all the tokens are merged into a single stroke, or until all possible combinations of token merges have been attempted with negative results.

Next attempt to merge the orphan tokens into the existing strokes. Edge costs are computed in a similar manner as for regular tokens. Merges are successful if the orphan token lies within a forty five degree cone formed by the position of the token in the direction of the major axis, and two time the length of the hypotenuse of the major and minor axis of the token.

The merging process generates a set of strokes which cover the original set of tokens. Each token in the stroke has a width given by the minor axis of the moment of the token. By using a spatial b-spline to connect token positions and a scalar b-spline to interpolate widths, This process creates a smoothly varying form which approximates the shape of the segment and forms the basis for a brush stroke.

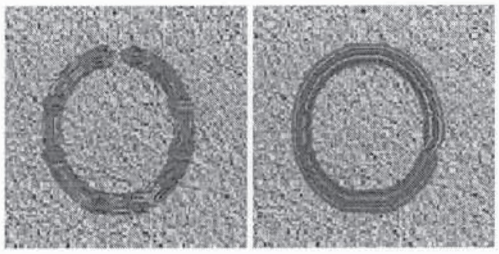

Figure 9: The medial transform algorithm is very susceptible to noise, and therefore can generate a large number of strokes in a single segmented region. The connect strokes algorithm attempts to link these small strokes into larger smooth strokes. This example shows a region painted with, left image 62 strokes, and without, right image 1 stroke, the use of the connection algorithim.

6.3 Grouping Tokens via Search Cones

We also explored a second method for grouping tokens that involves searching a cone of possible values for the beginning of another token. For each token search cones are created at the end points of the token in the direction of a vector formed by the first two points of the token and the last two points of the token and out to a distance of seven pixels. Then the cones of every token are tested to see if they intersect the cone of any other token. Tokens with intersecting cones are merged into strokes and the process is repeated until no further merges are possible. Unmerged tokens are made into single token strokes. Next the algorithim attempts to merge the orphan tokens into the existing strokes by testing their positions verses the search cones.

The major advantage of this method over the moment method is speed, the cone intersections can be hard coded, and merging generally takes less than log n passes over the data were n is the number of tokens. In addition in order to render this type of stroke the ridge set points are used directly without the over head of computing b-spline curves. The disadvantage is that this stroke representation may lack smoothness, and may self intersect causing artifacts when rendered with alpha blending.

6.4 Grouping Strokes

In addition to grouping tokens and strokes inside a single segmented region strokes and tokens from different regions can be merged. An example of why this may be beneficial is shown in Figure 6.4. The improvement to the system is purely esthetic and may not be suitable in every case. Stroke merging however incurs a huge cost in memory for the system.

Stroke merging can be accomplished using either the moment method or the search cone method. The memory overhead come form the fact that all of the tokens and strokes form all of the regions need to be saved and checked after each region is segmented for a given gray level. In practice this slows the system down by between one and two orders of magnitude depending on the memory requirements of the images used and the available memory of the machine on which the system is being run.

7 Rendering Images

7.1 Stroke Representation

The final phase of the algorithm is rendering an image from the brush stroke representations. Brush stroke rendering depends on the method of token grouping used in the previous phase of the algorithm. Next modified versions of Strassmann [22] or Pham's [16] algorithms are used to render brush strokes.

Strokes made up of tokens that are grouped using the moment method are rendered by using the positions of the moments as the control points of a b-spline curve. This list of control points is known as the control polynomial. As seen in Figure 7 The stroke rendering process begins by adding additional control points to the beginning and end of the control polynomial in order to ensure adequate length for the brush stoke. These points are found by applying and offset to the first and last control points of the control polygon. The offset used is the same offset as between the last and next to last points, and the first and second points respectively.

A spacial B-spline curve, described by this control polynomial, which smoothly interpolate a curve along the edge of a brush stroke are computed. As well as a b-spline curve that blends the width values at each of the control points to give the strokes a smoothly varying width. Finally offset curves in both directions normal to the spacial curve are computed using the width spline values.

Two integer arrays are filled with points calculated along both offset curves in a pairwise fashion. These arrays of points yeild list of quads which are filled using standard line drawing. The color of each line may be perturbed to yield an effect similar to a brush artifact.

Strokes that are grouped using the search cone method are rendered using a simpler method. The token points are entered in order into integer arrays representing the (X, Y) coordinates of the points. Next for each X, Y point a local normal direction is calculated by assuming that the given point is a point in a line segment composed of itself and its closest neighbor in the token. Lastly for each X, Y point two edge points are calculated in either direction normal to the point at a distance equal to the stored width at that point.

This method has the advantages of speed, and a great deal less computational and coding complexity than the B-spline method. However, this method can create visible artifacts when used with alpha blending. The normal directions calculated for each of the stroke points can intersect, causing what Strassmann called the "Bow Tie" effect. This effect is demonstrated in Figure 7. When a low Alpha value is use, cause the brush stroke to appear opaque, the brush stroke may contain these self intersections causing areas of the stroke to appear darker. In practice this effect is generally not noticeable with alpha values higher than 0.75.

8 Brush Effects

Painting effects such as brush artifacts, paint mixing between layers, and stroke connection are common to both rendering methods. Strassmann's algorithm is modified by adding rounded end caps to the brush strokes. Alpha blending is used to simulate paint mixing on a per stroke basis.

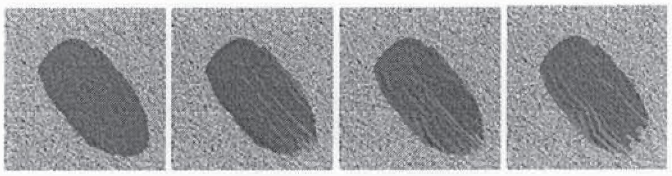

Figure 12: Depending on the viscosity of the paint a ridge artifact of varying height can be created when the paint is allowed to dry. This effect is simulated by adding light and dark bands to the stroke. A dry brush effect is simulated by halting the drawing of the brush stroke in a random manner. In our system these effects are adjustable by the user.

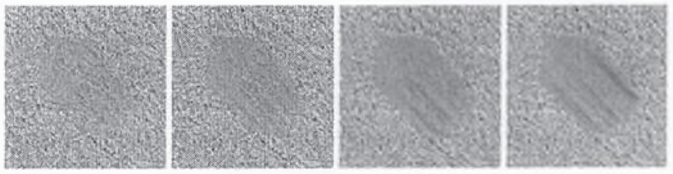

Figure 13: Alpha blending is used to simulate paints of various opacity. Paints with a high opacity will show the underlying substrate while paints with a low opacity will block the view of the substrate. The user adjustable Alpha parameter controls the percentage of blending between the under painting and the brush strokes.

Visual effects of the grouping algorithms discussed in the previous section are demonstrated in Figure 8.2. The medial transform algorithm is very susceptible to noise, and therefore can generate a large number of strokes in a single segmented region. The connect strokes algorithm attempts to link these small strokes into larger smooth strokes.

Alpha blending is used to simulate paints of various opacity. Paints with a high opacity will show the underlying substrate while paints with a low opacity will block the view of the substrate. The Alpha parameter controls the percentage of blending between the under painting and the brush strokes.

8.1 Under Painting

Under painting is simulated by allowing the user to render strokes on top of another image. Meyer [14], Hertzmann [10] and Shira et al [19] discuss under painting in their work. Meyer rendered brush strokes on an a background image. Hertzmann and Shira et al. blur the source image and render strokes on top of the blurred image.

The system allows the user to import separate source images and under painting images. In this way strokes can be rendered onto a background image like Meyer, or onto blurred images like Hertzmann and Shira. In addition the under painting can be used for artistic effect as in Figure 8.1. One can also use the output of the system as the input to another image allowing a painting to be built up in layers.

8.2 Effects of Segmentation

As discussed in Section 4 varying the number of gray levels in the segmentation can greatly vary the way in which the segmented image is perceived. This difference in the segmentation of the source image also has an effect on the painted image as seen in Figure 8.2.

8.3 User Directed Enhancement

One weakness in the system is the fact that human artists will attempt to make brush strokes in a manner that communicates the un-

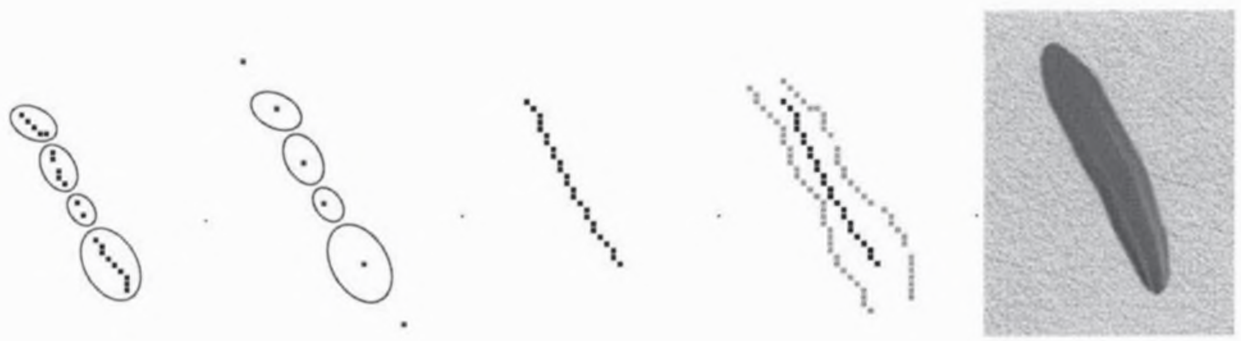
Figure 10: An example of our method for drawing strokes based on moment tokens. First, points are grouped into tokens and the moment of the group is taken Second, points are replaced by the moment centroid and additional points are added to the beginning and end of the token list Third, the points are used as the control polygon of a B-spline curve Forth, offset curves are computed to find the with of the stroke. Last, the stroke is rendered. Note that this stroke is smoother thatn the stroke that results form the method demonstrated below.

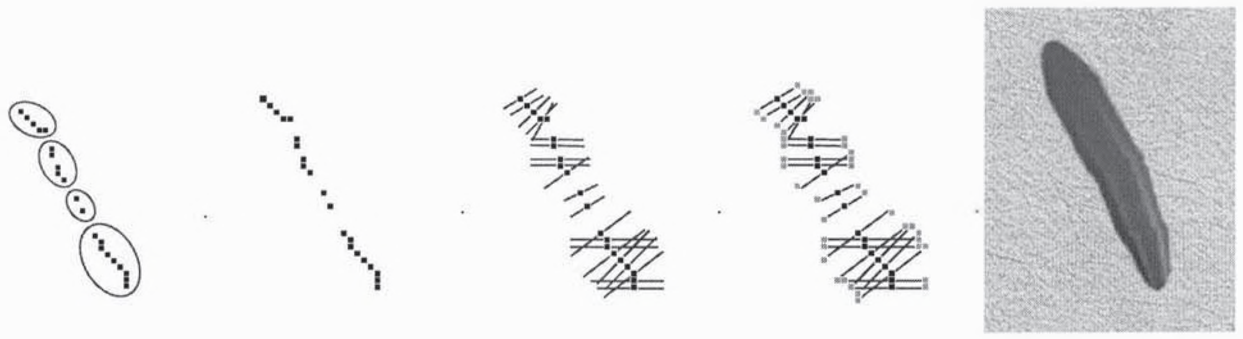
Figure 11: An example of our method for drawing strokes based on line lists. First, points are grouped into tokens. Second, tokens are grouped into a list of points. Third, for every point a normal line is found. Forth, based on the normal directions and the width at each point edge points for the stroke are computed. Last, a brush stroke is rendered. Note that while this stroke contains some artifacts the rendering time is significantly lower than for the stroke rendered using the above method.

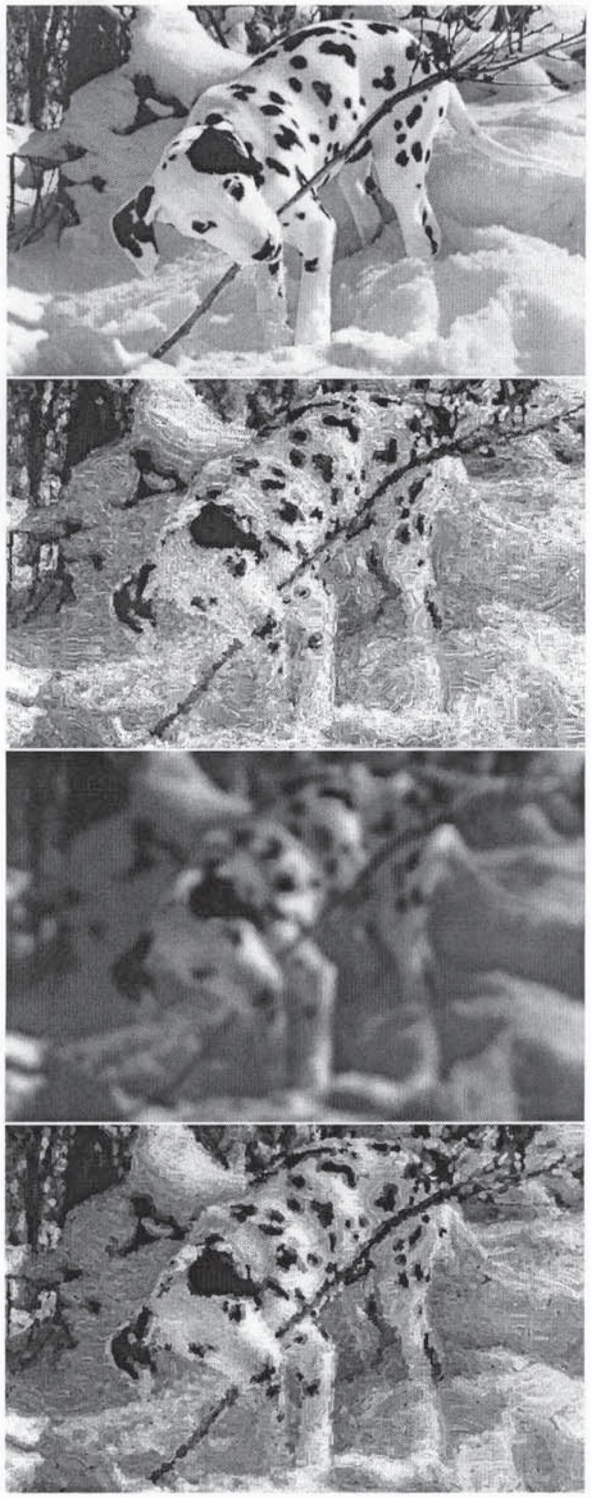

Figure 14: Under painting is a method used by artists to block in colored regions of a painting. We simulate under painting by allowing the user to render strokes on top of another image. This example shows a source image. Next we see a painting made from this image and suing the source image as an under painting. The third image is an under painting made by changing the color gamut of the original image and then blurring the image. The final painting was made by painting strokes, using the first image as a source, onto the modified underpainting. This tecnique can be expanded upon to create images with multiple layers.

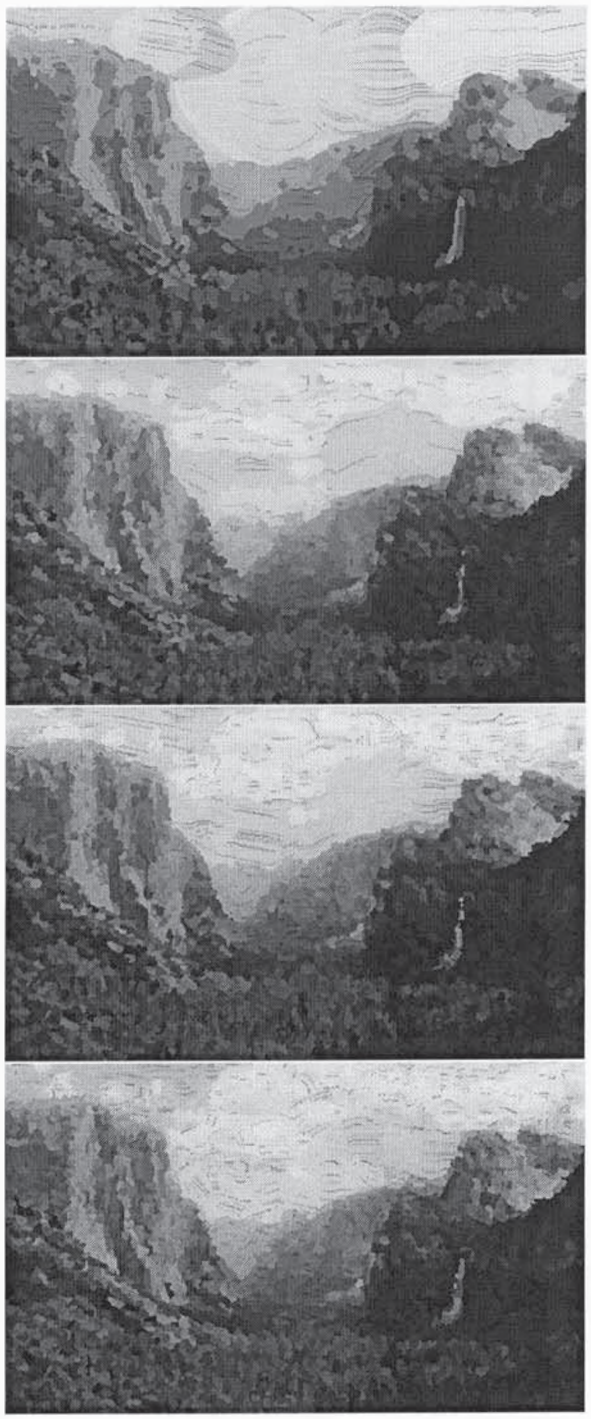

Figure 15: An example of varying the number of gray levels in the segmentation and the resulting paintings. All paintings used the Yosemite Vacation Image as a source image. From top to bottom the images were segmented using; 12, 48, 72, and 150 gray levels respectively.

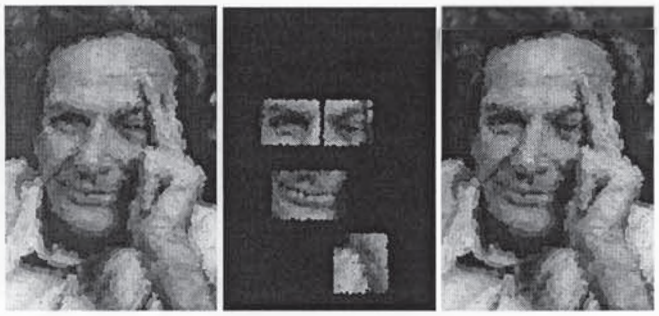

Figure 16: An example of user directed enhancement. The Feynman portrait is deemed by the user to lack detail in regions surrounding the eyes, mouth, and hand. The user selects these regions, and raise the segmentation level. New higher frequency strokes are drawn over the original strokes, hopefully improving the result.

derlying three dimensional structure of the subject. Because there is no additional information about the source image two semi successful work arounds have been implemented. The first, and simplest work around is to start brush strokes at the widest end of the stroke. This way any taper or dry brush effect proceeds in a manner consistent with physics.

The second method allows the user to select areas of interest in the image and re segment these areas with a higher number of gray levels in the segmentation. This allows the user to direct the placement of high frequency information in the image, and in many cases improves the visual appearance of the painting. An example of this process is shown in Figurereffig:enhancement.

9 Conclusion and Future Work

Our method achieves the basic goal of keeping the number of brushstrokes small compared to previous methods. The method is suitable for a variety of image types as shown in the previous section. However, there are a variety of image types where the method is poorly suited. These include images that require sophisticated segmentation, and images where viewers are highly sensitive to specific features of the image, such as detailed portraits.

We think productive future work would include improvements to every stage of the algorithm. Better segmentation, such as that given by anisotropic diffusion [23], would give immediate improvements in linking brushstrokes to salient features of the image. The computation of medial axes could be made less sensitive to noise if a continuous medial axis algorithm based on Voronoi partitions were used. This would simplify the job of the token-merging step in our algorithm which currently must account for noisy input. A simple improvement would be more sophisticated ordering of brushstrokes, such as optimizing order based on edge correlation with the original image. Once brushstroke order is known, more physically-based paint-mixing would give a look more reminiscent of oil painting. Our system could benefit from a user-assisted stage at the end to improve brushstroke-ordering. An estimate of foveal attractors in the image could allow brushstroke size to be varied with probable viewer interest. Most challenging, our method could probably be extended to animated sequences, using time-continuous brushstroke maps to ensure continuity. However, it is not clear what such animated sequences would look like, or to what extent they are useful. The most exciting potential future effort is to create actual stroke-based hardcopy using robotic or other technology.

References

[1] BERMAN, D. F., BARTELL, J. T., AND SALESIN, D. H. Multiresolution painting and compositing. *Proceedings of SIGGRAPH 94* (1994), 85–90.

[2] BLUM, H. A transformation for extracting new descriptions of shape. *Models for the Perception of Speech and Visual Form* (1967), 362–380.

[3] BURBECK, C. A., AND PIZER, S. M. Object representation by cores: Identifying and representing primitive spatial regions. *Vision Research 35*, 13 (1995), 1917–1930.

[4] COCKSHOTT, T. *Wet and Sticky: A Novel Model for Computer-Based Painting*. PhD thesis, University of Glasgow, 1991.

[5] CORMEN, T., LEISERSON, C., AND RIVEST, R. *Introduction to Algorithms*. MIT Press, 1990.

[6] CURTIS, C. J., ANDERSON, S. E., SEIMS, J. E., FLEISCHER, K. W., AND SALESIN, D. H. Computer-generated watercolor. *Proceedings of SIGGRAPH 97* (August 1997), pages 421–430.

[7] GOOCH, B., SLOAN, P.-P. J., GOOCH, A., SHIRLEY, P., AND RIESENFELD, R. Interactive technical illustration. *1999 ACM Symposium on Interactive 3D Graphics* (April 1999), 31–38.

[8] HAEBERLI, P. E. Paint by numbers: Abstract image representations. *Proceedings of SIGGRAPH 90 24*, 4 (August 1990), 207–214.

[9] HEARN, D., AND BAKER, M. P. *Computer Graphics*. Prentice-Hall, 1986.

[10] HERTZMANN, A. Painterly rendering with curved brush strokes of multiple sizes. *Proceedings of SIGGRAPH 98* (July 1998), 453–460.

[11] HUBBARD, P. M. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics 15*, 3 (July 1996), 179–210.

[12] JAIN, R., KASTURI, R., AND SCHUNCK, B. *Machine Vision*. McGraw-Hill, 1995.

[13] LITWINOWICZ, P. Processing images and video for an impressionist effect. *Proceedings of SIGGRAPH 97* (August 1997), 407–414.

[14] MEIER, B. J. Painterly rendering for animation. *Proceedings of SIGGRAPH 96* (August 1996), 477–484.

[15] PERLIN, K., AND VELHO, L. Live paint: Painting with procedural multiscale textures. *Proceedings of SIGGRAPH 95* (August 1995), 153–160.

[16] PHAM, B. Expressive brush strokes. *Computer Vision, Graphics, and Image Processing. Graphical Models and Image Processing 53*, 1 (Jan. 1991), 1–6.

[17] ROSENFELD, A. A characterization of parallel thinning algorithms. *InfoControl 29* (November 1975), 286–291.

[18] SCHLICK, C. Quantization techniques for visualization of high dynamic range pictures. *Proceedings of the 5th Eurographics Workshop* (June 1994), 7–20.

[19] SHIRAISHI, M., AND YAMAGUCHI, Y. Image moment-based stroke placement. Tech. Rep. skapps3794, University of Tokyo, Tokyo Japan, May 1999.

[20] SMITH, A. R. Varieties of digital painting. Tech. rep., Microsoft Research, August 1995.

[21] SOUSA, M. C., AND BUCHANAN, J. W. Computer-generated graphite pencil rendering of 3d polygonal models. *Computer Graphics Forum 18*, 3 (September 1999), 195–208.

[22] STRASSMANN, S. Hairy brushes. *Siggraph 20*, 4 (Aug. 1986), 225–232.

[23] TUMBLIN, J., AND TURK, G. Lcis: A boundary hierarchy for detail-preserving contrast reduction. *Proceedings of SIGGRAPH 99* (August 1999), 83–90. ISBN 0-20148-560-5. Held in Los Angeles, California.

[24] WILLIAMS, L. 3D paint. *1990 Symposium on Interactive 3D Graphics* (1990), 225–233.

[25] WONG, E. Artistic rendering of portrait photographs. Master's thesis, Cornell University, 1999.