Automatic Painting with Economized Strokes

Bruce Gooch Greg Coombe Peter Shirley

University of Utah www.cs.utah.edu

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 use very few brush-strokes. 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.

Automatic Painting with Economized Strokes

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 use very few brush-strokes. 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

Most art relies on representation and abstraction. In "realist" painting, the abstraction occurs when the detail of real images is approximated with limited spatial resolution (brushstrokes) and limited chromatic resolution (palette). Economy is a quality of many good paintings, and refers to the use of only those brushstrokes 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 brushstrokes. Paintings with economy may be useful for creating real paintings using robots, creating physical painting "replicas" using molded canvases that include brushstroke features[1], creating painterly digital images, and for compression of painterly images.

There are two main tasks involved in the creation of a digital painting. First is the creation of brushstroke positions. The second is the "rendering" of brushstrokes into pixel values. If the brushstroke positions are manually created by a user, then this is a classic "paint" program. If the brushstroke positions are computed algorithmically, then this is an "automatic" painting system. In either case, once the brushstroke geometry is known, the brushstrokes must then be rendered, usually simulating the physical nature of paint and canvas [4, 15, 21].

The economy of painting is determined when brushstroke paths and widths are created. We present an algorithm that carefully chooses brushstroke 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-brushstrokes 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 brushstroke. We use a variation of standard algorithms to render these brushstrokes.

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 brushstrokes, 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.

[1]The Artgraphs company has a process to create such painting replicas: www.artagraphs.com.

Figure 1: *An overview of the painting process. Top to bottom: The source image. The source image segmented into tokens. The tokens assembled into ordered lists. The final painting with the token lists rendered as brush strokes.*

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. A thorough overview and history of digital painting and techniques is provided by Smith [19].

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 unique raster algorithm [21]. Later Pham augmented this algorithm using b-splines and offset curves instead of a polyline to achieve a smoother brush path [15]. Williams provides a method of merging painting and sculpting by using the raster image as a height field [23].

Smith points out that by using a scale-invariant primitive for a brush stroke, multi-resolution paintings can be made [19]. Berman et al. showed that multi-resolution painting methods are efficient in both speed and memory usage [1]. Perlin and Velho used multi-resolution procedural textures to create realistic detail at any scale or dimension [14]. This 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].

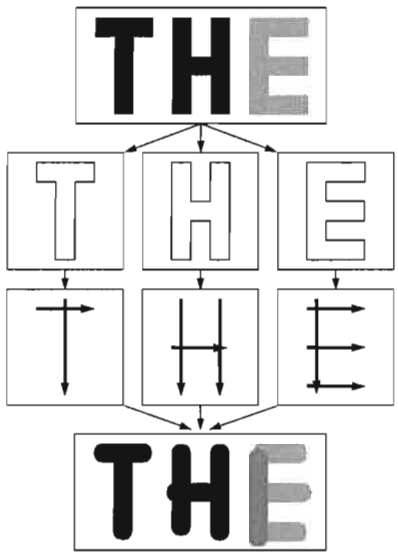

Figure 2: *The basic steps in the algorithm. The image is segmented and each segment is independently decomposed into brushstrokes. Each brushstroke is then "rendered" into a raster image.*

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 [20].

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 [24]. Meier produced painterly animations using a particle system [13]. Litwinowicz produced impressionist-style video by re-sampling a digital image and tracking optical flow [12]. Hertzmann refined Haeberli's technique by using progressively smaller brushes to create a hand-painted effect from a photograph automaticly [9]. Gooch et al. automaticly 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 brushstrokes.
3. Render brushstrokes in some order.

Artists are taught to paint by first producing tonal sketches of the scene they are painting [19]. In addition, recent work in computer graphics has shown that a first order approximation to the tone mapping operator should probably be achromatic [17]. We therefore segment an image based on pixel luminance, and color brushstrokes using the color ratios suggested by Schlick [17]. We allow a user to set the number of intensity thresholds the image will be segmented into. A set of approximately perceptually uniform grey levels is then created, and the image is segmented. 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.

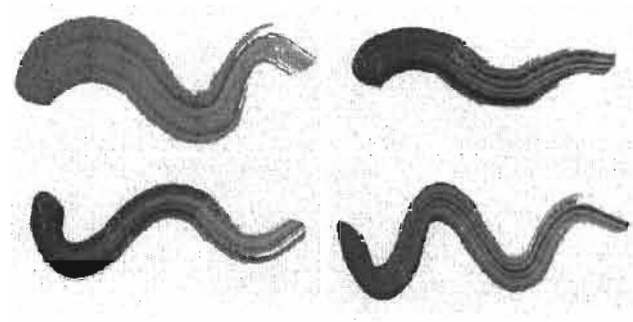

Figure 3: *Colored brush strokes modeled on a Filbert brush.*

Brushstroke path generation is the most complex part of our system. The algorithm 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 laced together into tokens. The tokens are then spatially sorted into ordered lists. The algorithm also estimates the "thickness" along the central axis to control brush width. The details of the brushstroke generation are the main contribution of this paper and are covered in the next section.

Strokes in our system are drawn top-to-bottom and left-to-right to emulate a right handed painter standing at an easel. The topmost end of the brushstroke is used as the comparison point. We use a modified version of Strassman [21] and Pham's [15] algorithms to render brush strokes to the screen. We use the center points of the tokens as the control points for cubic b-spline curves. We also make a spline curve of the widths of the stroke at the control points so that the widths will blend nicely.

Strassman and Pham modeled sumi-e brushes which taper on and taper off to a point during a brush stroke. We instead choose to model a Filbert brush used in oil painting. Filbert brushes are made as round brushes with round tips, and then flattened. They are good all around brushes combining some of the best features of flat and round brushes [19]. To model a Filbert brush, we constrain the "taper-on" to a circular curve and the "taper-off" to a parabolic curve. We also add an additional control point to the beginning and end of the stroke list to account for the length lost in building the b-spline. The additional front point is found by constructing a unit vector in the direction from the second control point to the first, scaling this vector by the width at the first control point, and adding this to the first control point. The additional end control point is found in a similar fashion.

4 Stroke path generation

The input to this process is a set of image segments, which are connected regions of similar intensity. The output of this process is a set of brush strokes with control paths and widths. The key to this process is to determine the dominant orientation of the segment, and to use this to create a brush path and width that closely follows the shape of the segment.

Originally, we attempted to use image moments for stroke path

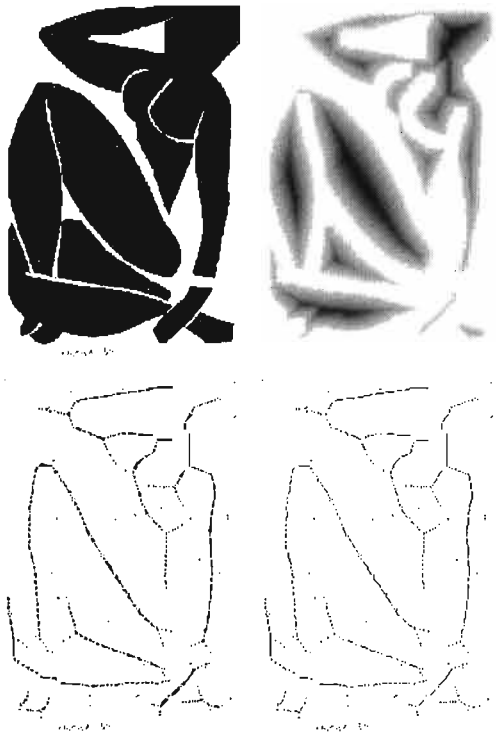

Figure 4: *Stages of transforms. Upper-left: input image. Upper-right: distance transform. Lower-left: Ridge-set extraction. Lower-right: After thinning.*

generation, inspired by Shira et al [18]. Image moments will find the dominant orientation and eccentricity of each segment and are scale and rotation invariant. Unfortunately they do not give information about how a stroke would vary along that orientation. We attempted to use a hierarchy of image grids and moments but found that the results were highly sensitive to the underlying grid structure. The problems we encountered with grids can be appreciated by studying the sign painter images in 1. The grid size should be small enough to recognize the hole in the *A*, yet large enough to regonize the *I* as a single stroke. We found this to be impossible for all but a small set of grid alignments and sizes.

The problems with grid artifacts led us to the medial axis transform. Like image moments, the medial-axis transform yields scale and rotation invariant measures for a segment, but the medial-axis is independent of a grid structure. In addition the transform yields width information along the medial axis. The path-generation step proceeds as follows:

1. Perform a distance transform to compute a set of ridge points which approximate the medial axis of a segment.

2. Use a thinning algorithm to remove artifacts from the ridge set.

3. Group the ridge set points into tokens, and merge these tokens into strokes.

4.1 Medial Axis Transform

The purpose of this step is to obtain a good approximation of the medial axis, or skeleton, of an image segment. The medial axis was first presented by Blum [2], and has been shown to be useful in coarsely approximating 2D [11] and 3D objects [10]. It has also been shown to be a good approximation of the way the human visual system perceives shape [3]. Although there are several equivalent definitions, we think of the medial axis as *the loci of the centers of all circles that are tangent to the object boundaries at more than one point.*

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. Thinning algorithms, like Rosenfeld's parallel algorithm [16], 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, this can often result in losing important information. Another drawback is that these algorithms do not produce information about the distance to the boundary, which is needed for brush stroke width estimation. 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. The double lines result from medial axis lines which should fall between pixels.

4.2 Hybrid Method

As we cannot rely entirely upon one method, we attempt to combine the positive aspects of both techniques into a hybrid method. We first apply the distance transform to extract a discrete approximation of the medial axis called the ridge set. We 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 with reduced sensitivity to noise along the boundary.

For each pixel in the image, we compute the shortest distance to the boundary using a distance transform [11]. The distance transform assigns a distance of one to each pixel on the boundary, a distance of two to the next layer in, and so on until we reach a pixel which is closer to another boundary (Figure 4). Given the binary image representation f_0 we iteratively compute $f_1 \ldots f_n$ at each pixel (x, y) using the following equations [11]:

$$f_i(x, y) = f_0(x, y) + min(f_{i-1}(p, q))$$

$\forall (p, q)$ in 8-neighborhood of (x, y). This algorithm terminates when $f_n = f_{n-1}$. The ridge points now satisfy the equation:

$$f_n(x, y) \geq f_n(p, q))$$

$\forall (p, q)$ in 8-neighborhood of (x, y). This process 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. Notice 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.

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. The connectivity problems are covered in Section 4.3. We use Rosenfeld's parallel thinning algorithm [16], 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. We used a fast algorithm to test the 8-simpleness of a pixel (Appendix). The thinning algorithm eliminates most double-lines and other noise from the ridge set. The thinning algorithm typically requires 2-3 passes over the binary ridge set data.

This combination of algorithms yields a set of approximate medial axis points. We then group spatially coherent points into small

tokens consisting of three or four points, as shown in Figure 1. A token is a small rectangle that encodes color, position, orientation, length, and width information. Grouping the medial axis points into tokens facilitates merging and tends to smooth small perturbations.

4.3 Merging Overview

We use the information in the set of tokens to create a set of brush strokes. This process is a variant of Prim's minimum spanning tree algorithm [5]. We create a set of links connecting every pair of tokens that are within a distance tolerance. We then compute edge weights by maximizing the desired stroke properties such as orientation and color. A priority queue is used to select the highest-weighted edge e_{ij}. The algorithm then attempts to merge that edge with the existing strokes. We represent a stroke as an ordered set of tokens $S = \{t_0, \ldots, t_n\}$. A merge is successful only if both tokens being merged are at the beginning or end of a stroke.

4.4 Edge Weighting

The crucial component of this algorithm is the weighting of the edges. The optimum weighting function would assign a weight of one to every link which preserved the continuity of the encompassing stroke, and a weight of zero to all other tokens. We devised a simple weighting function which behaves well in a wide range of cases. Our weighting function is:

$$w_{ij} = w_d proximity_{ij} + w_o orientation_{ij} + w_c intensity_{ij},$$

where

$$proximity_{ij} = \frac{l_i + l_j}{2\|p_i - p_j\|},$$

where l_i is the length of token i, p_i is the center of token i,

$$intensity_{ij} = \begin{cases} 1 & \text{if } |intensity_i - intensity_j| < c_{tol} \\ 0 & \text{otherwise} \end{cases}$$

and

$$orientation_{ij} = |\, \mathbf{v} \cdot \mathbf{h} \,|$$

given $\mathbf{h}$ as the normalized half-vector of the two vectors defined by the tokens (along the medial axis) and $\mathbf{v}$ as the normalized vector between token centers. We use weights $w_d = 0.55$, $w_o = 0.35$, and $w_c = 0.10$. The weight values, along with the c_{tol} value, are parameters that can be adjusted by a user to obtain various artistic effects. For example, increasing w_d will weight the spatial locality of moments more than the orientation, this will result in short, crooked or twisting strokes. Increasing w_o will emphasize long smooth strokes.

The merging process will generate a set of strokes which cover the original set of tokens. Each token in the stroke has a width. By using a spatial b-spline to connect token centers and a scalar b-spline to interpolate widths, we create a smoothly varying form which approximates the shape of the segment and maps directly to a brush stroke.

5 Results

We use a simple three step algorithm for rendering brush strokes that mimics the process a painter might use. First we prepare the substrate to be painted on. Next we make an under-painting. Finally we render our brush strokes onto the image. A full run of our algorithm is shown in Figure 5. The under-painting is done

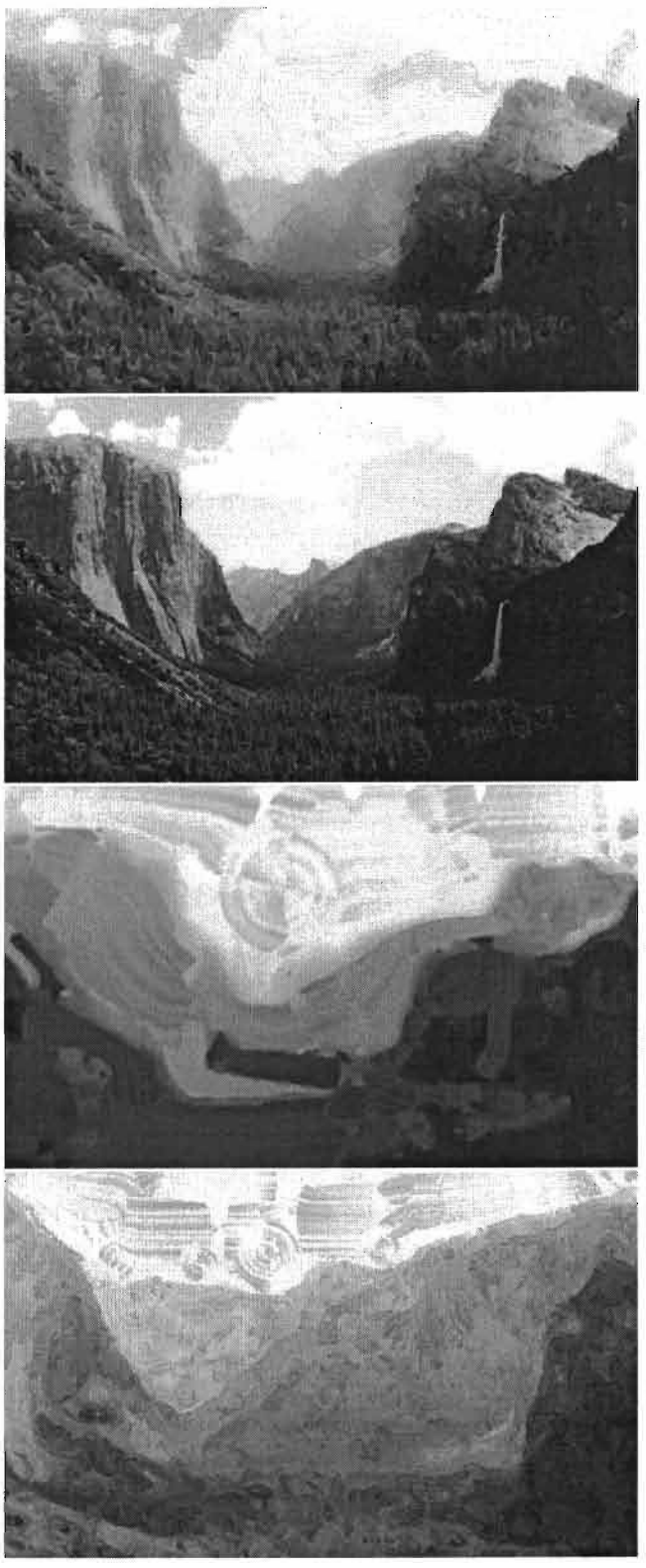

Figure 5: *Top to bottom: final painting with 2035 brushstrokes, original photo, under-painting, close-up.*

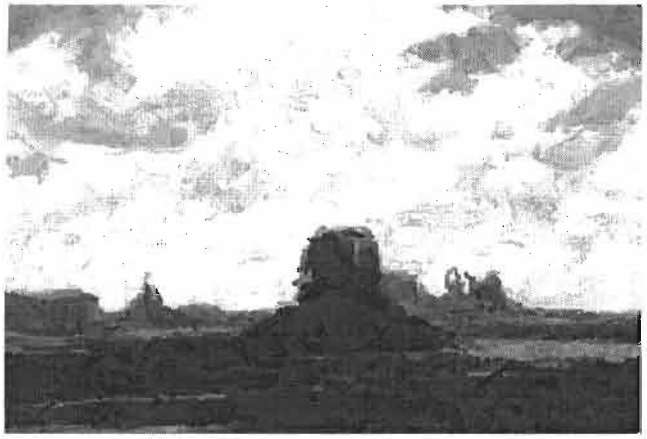

Figure 6: *Monument Valley.*

Figure 7: *Still life (original photo courtesy Cornell Program of Computer Graphics).*

with very coarse strokes and ensures that gaps between final brush strokes look reasonable. The final painting has 2035 brush strokes.

For the substrate of our images we use tileable cloth or paper textures. An under-painting is created by blurring the original image, segmenting the blurred image with a small (5 - 9) number of intensity levels and painting the segmented regions. An unblurred image is then segmented at the user defined number of intensity levels and painted over the under-painting. Our only attempt to emulate paint mixing is a weighted-average alpha blend of the substrate and under-painting at $\alpha = 0.1$ and of the paint strokes with the under-painting at $\alpha = 0.8$.

6 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

Figure 8: *Grand Tetons.*

to every stage of the algorithm. Better segmentation, such as that given by anisotropic diffusion [22], 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 fovial attracters 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.

Appendix: Thinning Algorithm

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, $N = \{i \mid i \in$ 8nbd of c, $i \in S\}$

output boolean simple, not_simple

Definitions

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).

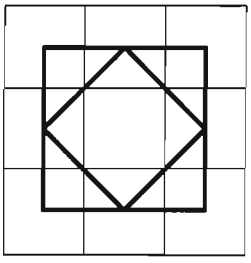

Figure 9: The graph representation of an 8-neighborhood.

Observations

Construct a graph $G(V, E)$ as such: let $V = \{v \mid v \in \text{8-nbd}\}$ and let $E = \{e_{ij} \mid \|i - j\| \leq \sqrt{2}\}$. This is illustrated in Figure 9.

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'. 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) $v - 1 \leq e$.

Method

Represent $G(V, E)$ as $E_i = \{j \mid \|i - j\| \leq \sqrt{2}\}$. Then,

```
∀i ∈ N {
  if ( E_i ∩ N = φ ) return not_simple; // isolated pixel
  else edges += degree( E_i ∩ N );
}
if ( edges ≥ v − 1 ) return simple;
else return not_simple
```

Notes

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.

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

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

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

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

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

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

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

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

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

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

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

[20] 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.

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

[22] 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.

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

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