ARTISTIC VISION: AUTOMATIC DIGITAL PAINTING USING COMPUTER VISION ALGORITHIMS

by

Bruce Gooch

A thesis submitted to the faculty of
The University of Utah
in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Science

School of Computing

The University of Utah

May 2001

ABSTRACT

This thesis presents a method for creating simulated oil paintings. A raster image is next used as input to an algorithm that produces a painting-like image composed of brush strokes rather than pixels. Ultimately the sequence of brush strokes representing an interpreted image can be rendered in pixel form, however the brush stroke structure is far more compact for storage and transmission. Unlike previous automatic painting methods, this algorithm attempts to use very few brush-strokes. The algorithm achieves economy of brush strokes 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. The process creates images reminiscent of modern realist painters who often want an abstract or sketchy quality in their work.

To Amy,

More than the Sun, the Moon, and all the Stars.

CONTENTS

ACKNOWLEDGEMENTS

Thanks to Amy Gooch, Greg Coombe, Georgette Zubeck, Peter Shirley, Bill Martin, Rich Riesenfeld, Elaine Cohen, Chuck Hansen, Susan Ashurst, David Johnson, Richard Coffey, Colette Mullenhoff, Matt Kaplan, Tom Thompson, Russ Fish the Alpha1 research group, the Vis Sim research group, and the Sci research group. This work was supported in part by DARPA (F33615-96-C-5621) and/or the NSF Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-89-20219). All opinions, findings, conclusions or recommendations expressed in this document are those of the author and do not necessarily reflect the views of the sponsoring agencies.

CHAPTER 1

INTRODUCTION

In many cases simulating reality is not as important as presenting an observer with just enough information to create the illusion of reality. A new realm of computer graphics is fast growing in the attention that it is given, and the amount of research that is being done. This new part of computer graphics has been grouped under the name non-photorealistic rendering (NPR).

Although it is odd that it is defined by what it is not, it does provide a method to categorize work that has previously been thought of as an amusing distraction. The driving force of computer graphics since its conception has been realism. The term photorealistic rendering has been used to describe this large area of computer graphics. An excellent definition of the phrase photorealistic rendering by analyzing the phrase in parts:

photo: derived from the Greek word phos, meaning light or produced by light

realistic: fidelity in art and literature to nature or to real life and to accurate representation without idealization

rendering: in computer graphics, refers to the process where a representation of a virtual scene is converted into an image for viewing.

In photorealistic rendering the approach is usually physically based, duplicating the light, material properties, reflections, and refractions of objects. Whereas NPR involves stylization and communication, usually driven by human perception. NPR brings together art and science, concentrating less on the process and more on the the communication content of an image. Photorealistic rendering can be thought of as objective, whereas NPR is mostly subjective. In photorealistic rendering, it is hard to neglect detail, in fact the highest level of detail is generally preferred. However, the level of detail in NPR runs the range and the level is usually adapted across the image to focus the viewers attention. Imagery generated by artists provides information about objects that may not be readily apparent in photographs or real life. That the same goal should apply to computer generated images is the driving force behind NPR.

As we research NPR, we try to evaluate the following important ideas that tend to turn up in any discussion of NPR. Is it difficult to imitate the decisions of a real artist? Can we give computer programs artistic inventiveness? Can programs be expressive in themselves? Researchers have observed that applying one "blanket" or "blind" technique to an image can create pleasing results. But programs that take over all artistic decisions are generally not robust. Is this due only to complexity? Can creativity really be automated? One observation is that computers can attend to fine detail and repetitive tasks, but without a user, simulating artistic expression is difficult and, some hold, impossible.

Many computer graphics researchers are exploring NPR techniques as an alternative to realistic rendering. More importantly, NPR is now being acknowledged for its ability to communicate serious ideas. Techniques which have long been used by artists can be applied to computer graphics to emphasize specific features, expose subtle attributes, and omit extraneous information. One key idea that is often missed when scientists try to codify art is that one can not just apply any NPR filter to any image or scene and produce a work of art. The benefits of NPR are maximized when one thinks about the subject matter, the scene composition, and the purpose of the image.

Art relies on representation and abstraction to communicate complex visual ideas. 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. This work explores an automated painting algorithm that attempts to achieve economy, particularly in its use of brush strokes. Paintings with economy may be useful for creating real paintings using robots, for creating physical painting "replicas" using molded canvases that include brush stroke features creating painterly digital images, and for compression of painterly images.

This thesis discusses the creation of digital paintings. This process can be divided into three steps: i) parsing the input raster image into segments, ii) converting these segments into brush strokes, iii) rendering these brush strokes into an image. The first two steps may also be combined into a single step. Artists sometimes use a similar process called under painting in constructing paintings. An artist will divide their canvas into dark and light regions and paint a rough background painting composed of "fills" of these regions. This process is called under painting. Regions of similar color and intensity are then painted on top of the under painting. Finally highlights are painted.

CHAPTER 2

BACKGROUND

Relatively little work dealing with art has been published in the computer graphics literature. Several researchers have attempted to optimize how well an image communicates [18, 24], others have borrowed principles from technical illustration [37]. Some have examined how cinematographers develop animation sequences [46, 22]. Creating an environment with pleasing lighting has also been automated [23].

2.1 Art and Computer Graphics

William Horton [18] wrote about using graphics in the context of visual communication. His paper gives lists of rules for avoiding common blunders when creating images. He also points out methods that advertisers use to mislead us with images. Simple rules include avoiding the use of every color available, making sure the background does not obscure the foreground, and integrating images with text for a pleasing visual effect.

Paul Lester [24] wrote about visual communication with computer images. Lester stresses that the message must come before the visual. He makes examples of the movies "Tron" and "The Last Star Fighter" as pictures were the images are great but the message, in these cases the story, is not compelling. Lester points out four visual cues that the brain is designed to notice: color, form, depth, and movement. He also notes that each visual cue has a set of symbolic components that are agreed upon by cultural groups. Lester asserts that, by understanding and manipulating these symbols, images and animations can be created that provide superior communication content.

Seligmann and Feiner [37] created a system based on the idea that an illustration is a picture that is designed to fulfill an communicative intent. They assert that the the purpose of illustration is to show an object's material, size, orientation, and maybe how to use it. The purpose is to display not only geometric and material information, but to also provide the viewer with information about the object's features, physical properties, and abstract properties. "Illustration objects are generated based on both the representation of the physical object as well as the communicative intent." Their Intent-Based Illustration System(IBIS) uses a generate-and-test approach to consider how the final illustration will look. For each illustration, there are several methods and several evaluators. By performing the permutations of the methods and then evaluating them by the "rules," IBIS automatically generates the image that would "look" best.

He et al. [46] presented a paradigm for automaticly generating complete camera specifications for capturing participants in virtual environments in real time. Their paradigm is based on methods film directors use to to communicate interaction between actors in a movie. Very specific interaction modes, such as two or three person conversations, are encoded as finite state machines. These finite state machines control the placement of virtual cameras in the environment and influence the position of the "virtual actors." The paper goes on to discuss a number of film heuristics and explains how they are encoded into finite state machines.

Kawai et al. [23] presented a method for designing the illumination for an environment using optimization techniques applied to a radiosity system. Based on user specified constraints and objectives an optimization of lighting parameters is performed for the illumination of the environment. Their system solves for the "best" possible settings for: light source emissivities, element reflectivities, and spotlight directionality parameters so that the design goals, such as to minimize energy or to give the room an impression of privacy, are met. Their system employs an object space perceptual model based on work by Tumblin and Rushmeier [43] to account for psychophysical effects such as subjective brightness and the visual adaptation level of a viewer.

2.2 Art and Perceptual Psychology

Compositional rules are drawn from both the art and psychology literature. Although the psychology community has learned a great deal about some aspects of artistic composition [30, 40, 51], much of their knowledge is not yet specific enough to allow automation. Another problem encountered was that people who write about art do not seem to know how the physical act of producing a work of art is performed. Of course the contrapositive is also true, people who produce great works of art tend not to write about how they accomplished the feat.

V.S. Ramachandran and Willian Hirstein's [30] paper "The Science of Art: A Neurological Theory of Esthetic Experience" presents a theory of human artistic experience and its underlying neural mechanisms. They propose that any theory of art or human nature must have three components: first what they call universal rules or principles (logic), second why did these rules evolve and why they have the form that they do (evolutionary rationale); and third how is the brain circuitry involved. The paper begins by listing "Eight Laws of artistic experience" and a set of heuristics that artists seem to employ. They are however somewhat lacking in direct experimentation that would back up all of their claims.

Robert Solso's book "Cognition and the Visual Arts" [40] is an excellent resource for anyone studying how artists manipulate the various methods humans perceive art. The book is organized as a text on the visual system, with the addition of works of art that demonstrate all of the concepts presented. This book works as a wonderful proof of concept, artists do use the fundamentals of the visual system in their work. However, this book provides little in the way artistic technique for achieving these effects.

Richard Zakia's book "Perception and Imaging" is another great source for finding proof of concept ideas about the human visual system and how we perceive art. [51] Zakia has also organized his book along the lines of a visual perception book, but he uses the paradigm of a photographer to explain the manipulation of the perceptual effects. Like Solso, this book also suffers from the problem of the author communicating "see what I did" instead of "here is how you can do this".

2.3 Artistic Composition from Art Literature

Rules for artistic composition fall into four basic categories; format (image size, shape, and orientation), viewpoint, layout, and lighting. The subject of format is covered by Ray Bethers [4] who suggests that the format of a painting should be established at the beginning of a composition. Landscape formats should be used with horizontal objects or scenes, and portrait formats with vertically composed images. Sander [35] and Arnheim [2] showed that the *golden ratios* seem to be preferred as a choice for format proportions. A thorough investigation of viewpoints was recently carried out by Blantz et al. [5]. Palmer et al. [27] found that humans preferred views that are off-axis. Verfaillie [45] discovered that a three-quarter view of a familiar object is preferred. Edelman [13] showed that preferred views for unfamiliar "nonsense" objects may also exist. Clifton [9] shows how good layout can be achieved using rules of thirds and fifths, and also gives rules for dividing the image plane based on the viewpoint. Rudolf Arnheim [2] in his latest book shows that humans prefer objects to be balanced in terms of their center of gravity. Previously Arnheim [1] demonstrated that objects should be repelled form the direct center of the image. Hunter and Fuqua [20] give an algorithmic method, based on photographic techniques, to light an object in a way that reveals shape without making the lighting visually dominate the image.

Ray Bethers' book "Composition in Pictures" [4] is a standard text for universities and artists in general. Bethers demonstrates numerous artistic principles including; composition and subject can be separated, composition can vary with the same subject, compositions can be borrowed, and the idea that nature can be confusing but images need not confuse. Bethers gives a few techniques for achieving “good” composition and then spends the remainder of the book discussing pitfalls to be avoided by artists. Bethers contends that the format of the image should be established at the start of the artistic process. He also provides rules for choosing the format, vertical figures in vertical formats, and horizontal figures in horizontal formats. Bethers also contends that simple color schemes can have more complex composition, whereas complex colors force the use of simple composition. Bethers further states that the corners of an image are natural arrows that tend to draw the gaze off the image; therefore care must be taken to avoid anything in the image which points into a corner.

Friedrich Sander was a perceptual psychologist who did some of the early work in how humans perceive lines relative to each other. His results showed that humans prefer images organized around the “Golden Rectangle.” [35]

Rudolf Arnheim’s book “The Power of the Center: A Study of Composition in the Visual Arts” [2] demonstrates where objects in a painted scene should be place in order to aid in the communication of the images message. Arnheim states that the critical details in the image should be placed in the center of the image or just off center. He states that the gaze of the viewer passes across the center of an image most often as the eye scans an image. Therefore items placed in the center are most likely to have high “gaze times.” Arnheim agrees with Bethers when he states that corners of an image attract they gaze and must be visually blocked to avoid having the viewers gaze leave the image.

Blantz et al. [5] investigated preferred views for three-dimensional familiar and nonsense objects using psychophysical experiments conducted on computer graphic displays. These displays allowed participants to rotate shaded three-dimensional models in real time. Two experiments were performed. In the first experiment participants adjusted the view of each object to find the view they considered best for taking a picture of the object. In the second experiment participants were asked to mentally image an object based on the objects name alone, then they were presented with at three dimensional computer model of the object and asked to move it to the same view as their mental image. Both experiments showed a large degree of consistency across participants in terms of the preferred view for a given object.

Palmer et al. [27] performed a series of experiments in order to confirm the idea of canonic forms, such as object views, as memory representatives. In their classic experiment participants were shown photographs of common objects, such as a horse, and asked to name the object as quickly as possible. The idea being that those views for which the object was named the fastest are the canonical views of the object. These views tended to be off axis, and slightly above the object. In a second experiment participants were shown photographs of objects and asked which of the photographs was the best photo of the object. Not surprisingly the same views as for the first experiment were found.

Verfaillie and Boutsen [45] constructed a set of full-color images for use in research on the effects of in-depth rotation on visual identification of three-dimensional objects. The total number of images amounts to 714. Their article contains ratings of the ”goodness” of each view, based on Thurstonian scaling of subjects’ preferences in a paired-comparison experiment. The article also reports on an exploratory cluster analysis on the scaling solutions. This analysis indicates that the amount of information available in a given view generally is the major determinant of the goodness of the view. For instance, objects with an elongated front-back axis tend to cluster together and the front and back views of these objects, which do not reveal the object’s major surfaces and features, are evaluated as the worst views.

Jack Clifton’s book “The Eye of the Artist” [9] is written by someone who knows how to produce high quality artwork, and is written with the beginning artist in mind. This book is an excellent source of information on the subject of artistic composition. Clifton provides explicit drawings of the rules of third and fifths, and shows the problems that can occurs if the rules are broken. He also shows examples of “good” artwork were the artist has broken these rules to suit the communication goal of the artwork. Clifton also gives other composition rule concerning viewpoint and perspective as well as depth partitioning, and mistakes to avoid.

Rudolph Arnheim’s book “Art and Visual Perception: A Psychology of the Creative Eye” [1] is the standard reference of artistic knowledge, and Arnheim is one of the accepted experts in the field. In terms off artistic composition Arnheim expounds on: object placement in the image plane, drawing objects away form the corners of an image, placing objects in the picture plane to aid the viewer in “reading” the image, and visual balance in images. According to research sited by Arnheim objects in an image should be placed slightly off center in order to add visual tension to the image. Yet objects should not be placed in the corner in order to avoid the gaze off the image problem. In addition Arnheim maintains that focal point of the image should be in the upper right quadrant of the image. This he says set up a kind of diagonal left to right tension in the image that is favored in cultures with left-to-right text. Arnheim also maintains that visual balance, which he defines as intensity balance, left to right and top to bottom in images, makes for better images.

Studio photographers face the practical problem of illuminating objects in a way that reveals shape without making the lighting visually dominate the image. Hunter and Fuqua [20] give an algorithmic way to light an object that avoids this problem.

2.4 Computer Graphic Painting

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 or drawing. An overview and history of digital painting and techniques is provided by Smith [39].

2.5 Simulating Painting Mediums

Work intended to simulate artistic mediums can be further divided into those that 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 [42]. Later Pham augmented this algorithm using B-splines and offset curves instead of a polyline to achieve a smoother brush path [29]. Williams provides a method of merging painting and sculpting by using the raster image as a height field [47].

Steven Strassmann [42] presents one of the first ideas for modeling brushes and ink. The brushes are modeled by an array of pixels, and the brush stroke by a cubic B-spline, in order to render the stroke the pixel array is dragged along the spline curve. By varying pixel parameters during the brush stroke a user is able to simulate stroke shading, and dry brush effects.

Binh Pham [29] presented some incremental changes is Strassmann’s paper, using offset curves instead of a pixel array. Brush strokes are modeled using a technique based on variable offset approximation of uniform cubic B-splines. The trajectory of a brush stroke is represented as a three-dimensional cubic B-spline and each bristle as a three-dimensional offset cubic B-spline. Pham maintained that his technique facilitated the process of inputing data, simplified the computation task, and provided advantages in the animation of brush strokes. ”

Lance Williams [47] presented a method by which digital painting could be extended into the third dimension. He also provided sculpting tools for for this type of modeling. The key to this approach was to use hardware to display the raster field as a height field. Thus making flat images three dimensional based on their luminance.

Smith [39] points out that by using a scale-invariant primitive for a brush stroke, multiresolution paintings can be made. Berman et al. [3] showed that multiresolution painting methods are efficient in both speed and memory usage. Perlin and Velho [28] used multiresolution procedural textures to create realistic detail at any scale or dimension. This work emphasizes that digital paintings stored as strokes may be useful for transmitting stylized images across a network.

Smith [39] provides a method to distinguish between the various meanings that digital painting may have. Each type of painting is discussed in its multiresolution generalization. The taxonomy splits painting into discrete and continuous categories and each of those into maxing and nonmaxing subcategories. An interesting aspect of this paper is that, by using a scale invariant primitive for a brush stroke continuous (i.e., multiresolution), paintings could be made. None of the automatic painting programs, with the possible exception of Curtis et al. [12], have achieved this.

Berman et al. [3] describe a representation for multiresolution images and methods for creating such images using painting and compositing operations. The methods are easy to implement and efficient in both memory and computational speed. Only the detail present at a particular resolution is stored. They also allow a user to display and edit portions of a multiresolution image at an arbitrary size.

Perlin and Velho [28] presented a system that used procedural multiresolution paint textures. Texture elements are linearly combined to create complex composite textures that continue to refine themselves when viewed at successively greater magnification. They demonstrated several examples of procedural textures and showed how to create painting effects with them.

Several authors have simulated the interaction of paper or canvas and a drawing or painting instrument. Cockshott [10] simulated the substrate, diffusion, and gravity in a physically-based paint system. Curtis et al. [12] modeled fluid flow, absorption, and particle distribution to simulate watercolor. Sousa and Buchanan [41] simulated pencil drawing by modeling the physics and interaction of pencil lead, paper, and blending tools.

Tunde Cockshott's [10] thesis presents a good method for simulating the physical process of painting on a computer. A novel paradigm, "Wet and Sticky", is proposed which models the physical and behavioral characteristics of paint rather that just its color properties. The model includes canvas or substrate, paint, gravity, and diffusion. Both the method of approaching the problem and the software engineering are sound. Many improvements could be made. In fact his system could actually run in real time now.

The computer-generated watercolor work by Curtis et al. [12] creates a very high-end paint program which generates pictures by interactive painting, or automatic image "watercolorization." Given a three-dimensional geometric scene, they generate mattes isolating each object and then use the photorealistic rendering of the scene as the target image. The authors studied the techniques and physics of watercolor painting and developed algorithms that depend on the behavior of the paint, water, and paper. They provide information on watercolor materials and effects of dry-brush, edge darkening, backruns, granulation and separation of pigments, flow patterns, glazing, washes.

Sousa and Buchanan [41] present a graphite pencil renderer. They break down the problem of simulating pencil drawing into four parts: simulating the drawing materials, modeling the drawing primitives, simulating the basic rendering techniques used by artists and illustrators familiar with pencil rendering, and modeling the control of the drawing composition. They present nonphotorealistic graphite pencil rendering methods for outlining and shading. They also present drawing steps from preparatory sketches to finished rendering results. They demonstrate the capabilities of our approach with a variety of images generated from three-dimensional models.

2.6 Simulating Painting Techniques

The works discussed above are concerned with the low-level interaction of pigment with paper or canvas, but other authors aid a user in the creation of an art work, or automate the process altogether. Haeberli [15] built a paint system that re-samples a digital image based on a brush, and then automated this system using a second control image. Wong [50] built a system for charcoal drawing that prompts the user for input at critical stages of the artistic process. Salisbury et al. [33, 32] created an interactive system which allows users to paint with stroke textures to create pen-and-ink style illustrations. Winkenbach et al. [49] itemize rules, regulations, and traditions of hand-drawn illustrations and incorporated a large number of those principles into an automated rendering system. Meier [26] produced painterly animations using a particle system. Litwinowicz [25] produced impressionist-style video by re-sampling a digital image and tracking optical flow. Hertzmann [17] refined Haeberli's technique by using progressively smaller brushes to create a hand-painted effect from a photograph automaticly. Gooch et al. [14] automaticly generated technical illustrations from polygonal models of CAD parts.

Paul Haeberli [15] created a paint program which allows the user to manipulate an image using tools that can change the color and size of a brush, as well as the direction and shape of the stroke. The goal of his program is to allow the user to communicate surface color, surface curvature, center of focus, and location of edges, as well as eliminate distracting detail, provide cues about surface orientation, and influence viewers perception of the subject. In creating this program, Haeberli studied the techniques of several artists. He observed that traditional artists exaggerate important edges. Where dark areas meet light areas, the dark region is drawn darker and light region is drawn lighter. This causes the depth relationship between objects in a scene to be more explicit where they overlap.

Eric Wong's [50] thesis presents an interactive method for generating charcoal style drawn portraits. At various stages of the artistic process the user is polled for input on image segmentation, and artistic parameters such as line weight. One thing to note is that quite a bit of user interaction is needed at each step in the process. An excellent feature of this work is the idea of having the level of abstraction match the stage of the design process. That is, the early steps in the process look rough, and the rendering style becomes progressively better in later stages of the process. The software engineering on this thesis was well done, and the process is broken down well. Automation of the entire process may only depend on earlier stages being automated.

Salisbury et al. [33, 32, 34] designed an interactive system which allows users to paint with stroke textures to create pen-and-ink style illustrations. Using "stroke textures," the user can interactively create images similar to pen-and-ink drawings of an illustrator by placing the stroke textures. Their system supports scanned or rendered images which the user can reference as guides for outline and tone (intensity) [33]. In their paper, "Scale-Dependent Reproduction of Pen-and-Ink Illustrations" [32], they gave a new reconstruction algorithm that magnifies the low-resolution image while keeping the resulting image sharp along discontinuities. Scalability makes it really easy to incorporate pen-and-ink style image in printed media. Their "Orient able Textures for Image-Based Pen-and-Ink Illustration" [34] paper added high-level tools so the user could specify the texture orientation as well as new stroke textures. The end result is a compelling two-dimensional pen-and-ink illustration.

Winkenbach et al. [49] itemized rules, regulations, and traditions of hand-drawn illustrations to render a scenes. Their process involves computing visible surfaces and shadow polygons. The polygons are projected to Normalized Device Coordinate space and then used to build a two-dimensional BSP tree and planar map. Visible surfaces are rendered, and textures and strokes applied to surfaces using set operations on the two-dimensional BSP tree. Afterwards, outline strokes are added. Their system allows the user to specify where the detail lies. They also take into consideration the viewing direction of the user, in addition to the light source. They are limited by their library of "stroke textures." Their process takes about 30 minutes to compute and print out the resulting image.

Winkenbach et al. [48] render a single pen-and-ink style image for parametric free-form surfaces, using "controlled-density hatching" in order to convey tone (intensity), texture and shape.

Their paper provides a highly detailed algorithm on drawing lines (strokes) which gradually disappear in light areas of the surface or where too many lines converge. They use a planar map constructed from the parametric surfaces to clip strokes and generate outlines. The planar map is not constructed from three-dimensional BSP Trees, but by the following method. They tessellate every object and then compute the higher-resolution piecewise linear approximations for all silhouette curves of meshed objects. They then use two-dimensional BSP trees to render shadows [8].

Barbara Meier [26] presented a technique for rendering animations in which every frame looks as though it was painted by an artist. She models the surfaces of three-dimensional objects as three-dimensional particle sets. Particles are generated by tessellating a surface into triangles. The surface area of each triangle is computed and particles are randomly distributed. The number of particles per triangle is equal to a ratio of the surface area of triangle to the surface area of the whole object. To maintain coherence from one shot of the object to the next, the random seed is stored for each particle. Particles are transformed into screen space and sorted with regard to the distance from viewer. The particles are then painted with two-dimensional paint strokes, starting farthest away from viewer, moving forward, until everything is painted. The user determines artistic decisions like light, color, brush stroke, similar to most paint programs. The geometric lighting properties of the surface control the appearance of the brush strokes.

Peter Litwinowicz [25] created a system that takes as input video or digital photos and generates impressionistic images and animations, great for commercial community, but provides little more than pretty pictures. Litwinowicz blurs each frame and then uses a Sobel filter to find edges. Litwinowicz's animations are mostly frame-to-frame coherent, but have a tendency to be noisy and jumpy in areas with high frequency frame to frame change.

Aaron Hertzmann [17] presented a method for creating a painted image with a hand-painted appearance from a photograph. Images are built up from a series of spline brush strokes. The painted image is rendered 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. This method demonstrates a technique for painting with strokes aligned to normals of image gradients.

Gooch et al. [14] presented a paradigm for the display of technical illustrations in a dynamic environment. This display environment includes all of the benefits of computer generated technical illustrations. Such as a clearer picture of shape, structure, and material composition than traditional computer graphics methods.

It also includes the three-dimensional interactive strength of modern computer graphics display systems. This is accomplished by using algorithms for real time drawing of silhouette curves which solve a number of the problems inherent in previous methods.

The algorithms described in this thesis 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.

CHAPTER 3

ALGORITHM

The steps of the algorithm follow:

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 [36]. Hence source images are segmented based on pixel luminance, and color brush strokes using the color ratios suggested by Schlick [36]. 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 [39].

Each segment is independent; two segments of the same luminance are treated independently (i.e., the letters *T* and *H* in Figure 3.1). A more sophisticated segmentation strategy could be used without changing the rest of the 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 work and are covered in the next three chapters.

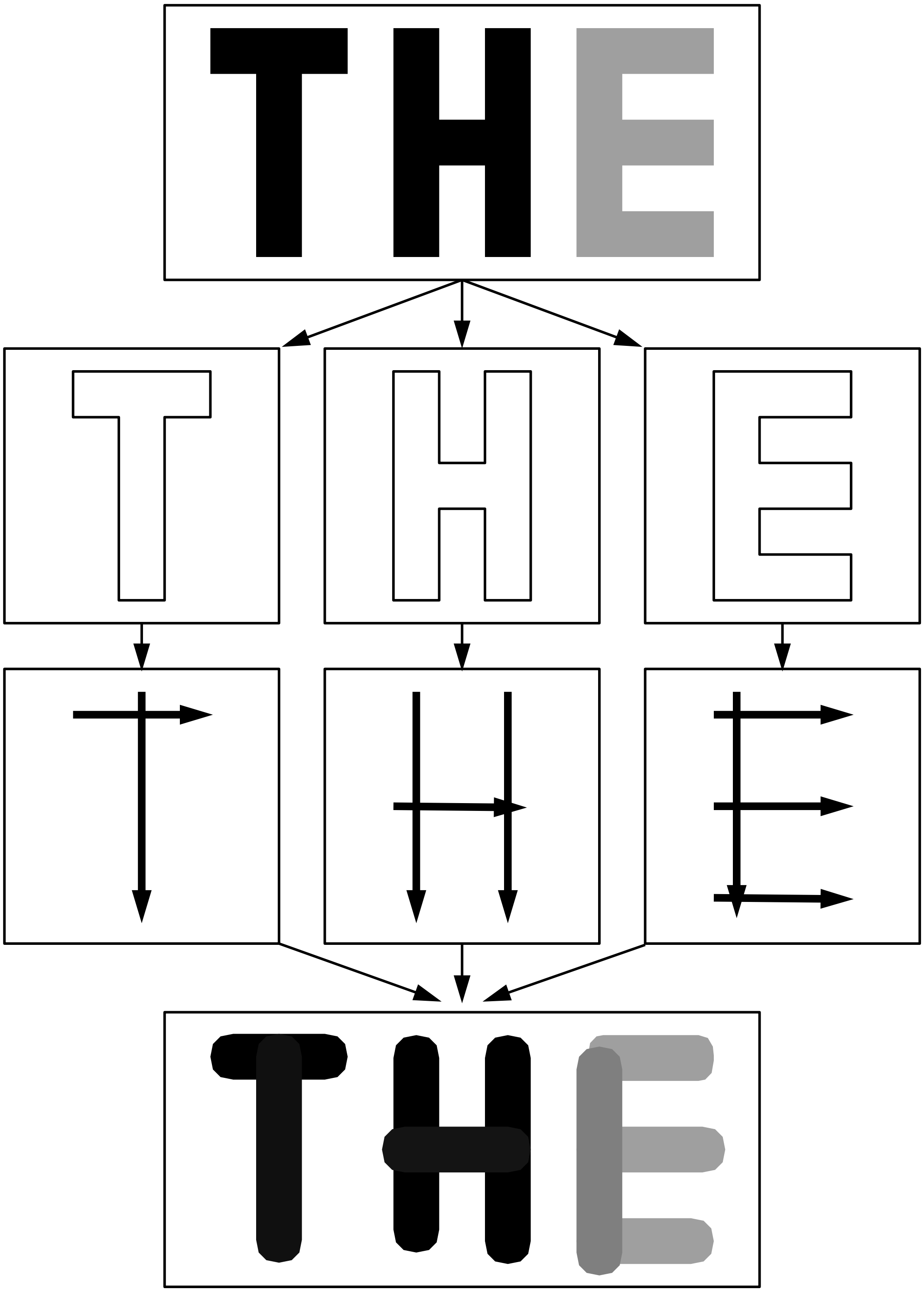

Figure 3.1. *The basic steps in the algorithm. First the image is segmented using flood filling. Next each segment is independently decomposed into brush strokes using vision algorithms and an approximate medial axis transform. Each brush stroke is then "rendered" into a raster image.*

Modified versions of Strassman [42] and Pham's [29] algorithms are used to render brush strokes to the screen. Strassman and Pham modeled sumi-e brushes that taper on and taper off to a point during a brush stroke. I 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-purpose brushes combining some of the best features of flat and round brushes [39]. 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 3.2.

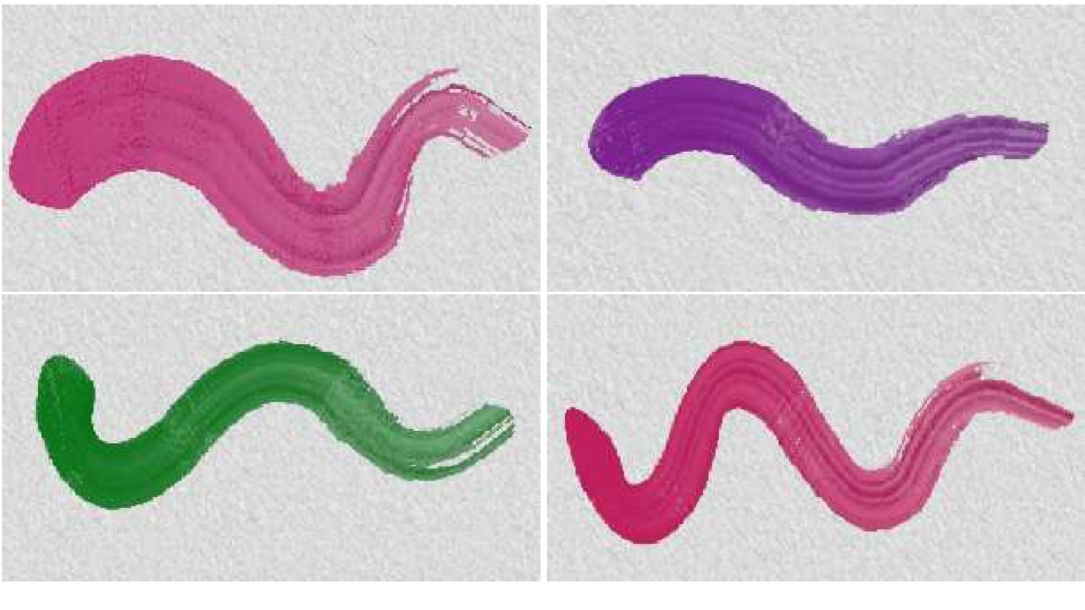

Figure 3.2. Digitally simulated brush strokes modeled on a Filbert brush.

CHAPTER 4

SEGMENTATION AND SMOOTHING

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

4.1 Hole Filling

The first vision algorithm run on a segmented region is a hole filling algorithm an example image can be seen in Figure 4.1. 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*.

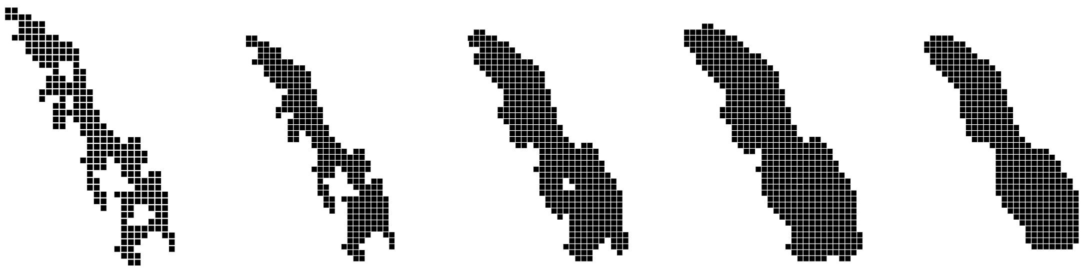

Figure 4.1. An example of a segmented region. The region after the hole filling algorithm has been run. The region after being grown, the region after having been shrunk.

4.2 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:

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

were r is the current threshold, I_0 is the initial intensity, and n is the number of intensity levels [16].

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. Examples of segmentation by intensity are shown in Figure 4.2.

In practice regions with fewer than 25 pixels tended to disappear 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.

4.3 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 as seen in Figure 4.1. 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 chapter with good results.

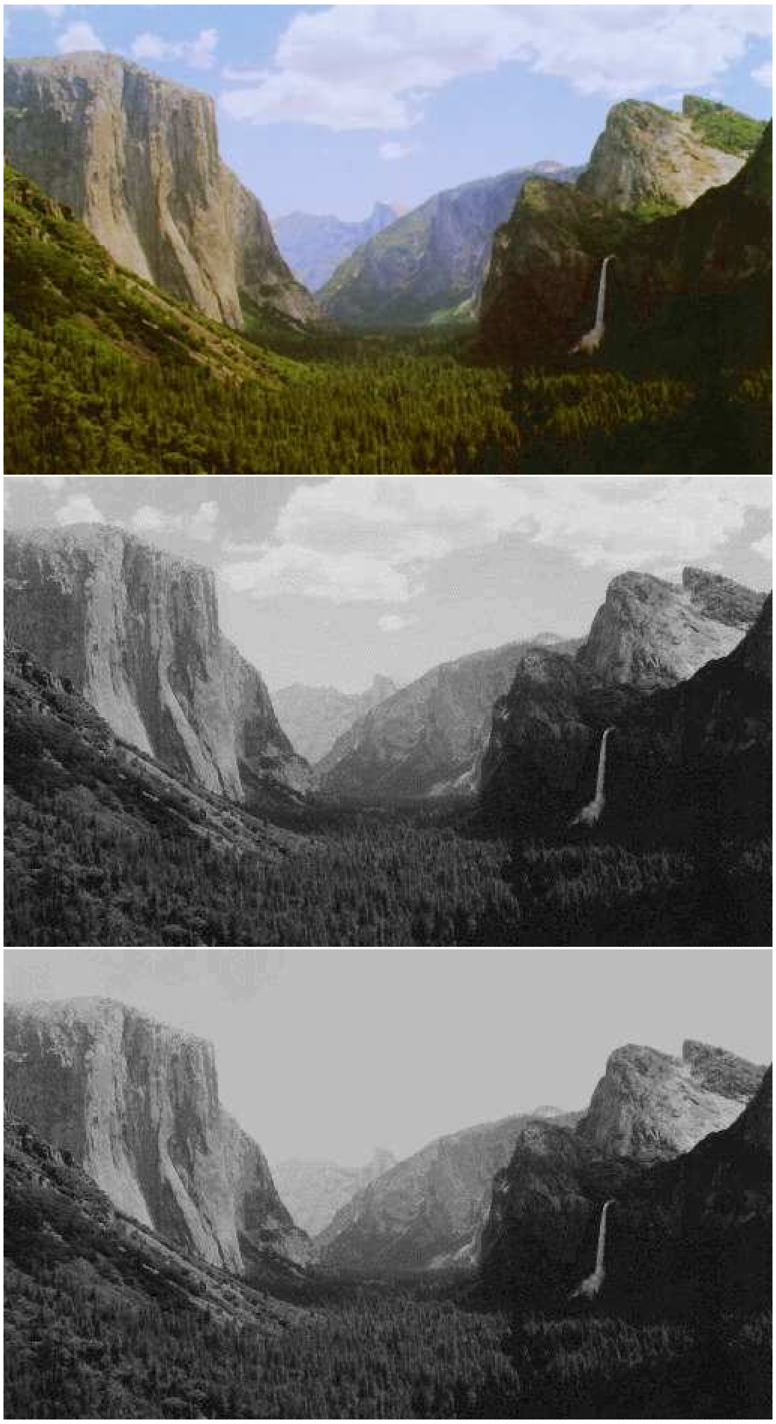

Figure 4.2. An example of varying the number of gray levels in the segmentation and the resulting images. From top to bottom; the source image, image segmented using 72 gray levels, image segmented using 48 gray levels. Note that the clouds have faded into the sky in the 48 gray level image. The effects of this type of segmentation artifact is reflected in the final painted image.

CHAPTER 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 [6], and has been shown to be useful in coarsely approximating two-dimensional [21] and three-dimensional objects [19]. The medial axis has also been shown to be a good approximation of the way the human visual system perceives shape [7].

Previous automatic painting methods use a hierarchy of image grids to segment the source image into strokes. I 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 do not preserve the connectedness of the medial axis lines. Thinning algorithms, like Rosenfeld's parallel algorithm [31], 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. 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 [21]. 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 5.1 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

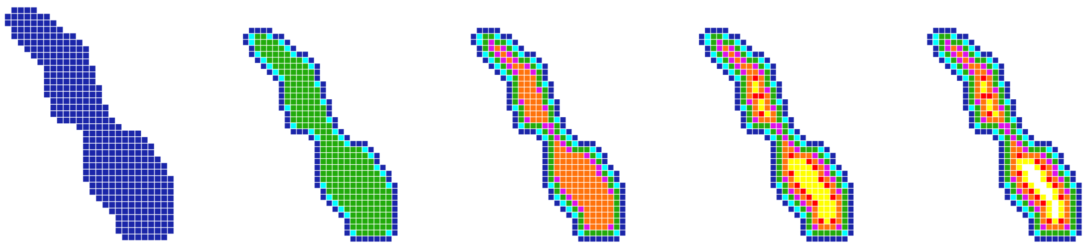

Figure 5.1. An example of a distance transform on a region. First all pixel values in the region are initialized to 1 if they are in the segmented region, 0 if they are not. Next multiple passes are made over the region. At each pass if a pixels neighborhood is nonzero, and contains values that are all less than or equal to the current value of that pixel a value is added to the contents of the pixel. This example shows the increasing value of the pixels by changing their color values to warmer values.

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, the ridge set is treated as a binary image and a thinning algorithm is run over the set. We use Rosenfeld's parallel thinning algorithm [31], 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 the 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 guaranteed 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. The following is a fast method for determining whether a pixel neighborhood is 8-simple.

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

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 $V = \{v \mid v \in \text{8-nbd}\}$ and let $E = \left\{e_{ij} \mid \|i-j\| \leq \sqrt{2}\right\}$. This is illustrated in Figure 5.2.

The graph G represents the 8-connectedness paths of the neighborhood. The intersection of the input set N with the graph G results in a new graph, $G'\,(V,\ E)$. Now the test for 8-simpleness just becomes a test for the connectedness of the planar graph G'. This means Euler's Relationship for connected planar graphs can be used. This 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 $E_i = \left\{j \mid \|i-j\| \leq \sqrt{2}\right\}$. 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.

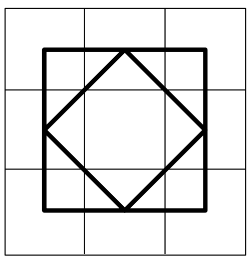

Figure 5.2. The graph representation of an 8-neighborhood.

CHAPTER 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 6.1. These tokens can next be 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.

I 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 [11]. 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

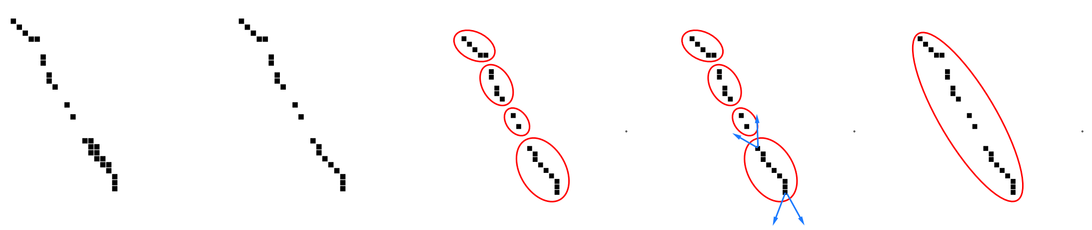

Figure 6.1. An example of ridge set extraction, thinning and grouping. First the ridge set is extracted from the distance transformed image. Next the thinning algorithm is applied. The resulting ridge set is next grouped into tokens, and these tokens are merged into a stroke.

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 6.1 This token set can be grouped into strokes.

6.2 Grouping Tokens via Moments

The first method explored for grouping tokens into strokes was taking 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. 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 the color of the token points from the original image is sampled as well.

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 [11]. 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 that 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.

6.3 Grouping Tokens via Search Cones

I 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 versus 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 6.2. 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.

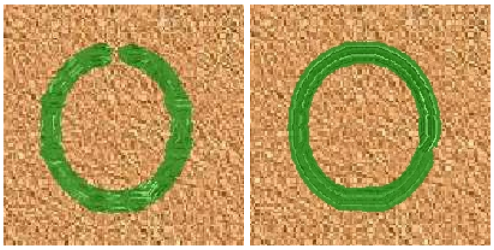

Figure 6.2. The medial transform algorithm is very susceptible to noise, and therefore can generate a large number of strokes in a single segmented region. The connect strokes algorithm attempts to link these small strokes into larger smooth strokes. This example shows a region painted with, left image 62 strokes, and without, right image 1 stroke, the use of the connection algorithm.

CHAPTER 7

RENDERING IMAGES

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 [42] or Pham's [29] algorithms are used to render brush strokes.

7.1 Stroke Representation

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 7.1 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 spatcial 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

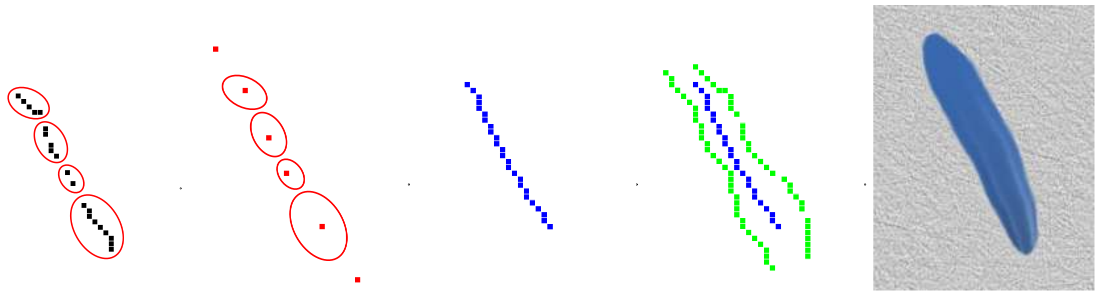

Figure 7.1. An example of the method for drawing strokes based on moment tokens. First, points are grouped into tokens and the moment of the group is taken Second, points are replaced by the moment centroid and additional points are added to the beginning and end of the token list Third, the points are 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 than the stroke that results form the method demonstrated below.

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 7.2. When a low Alpha value is used, causeing 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.

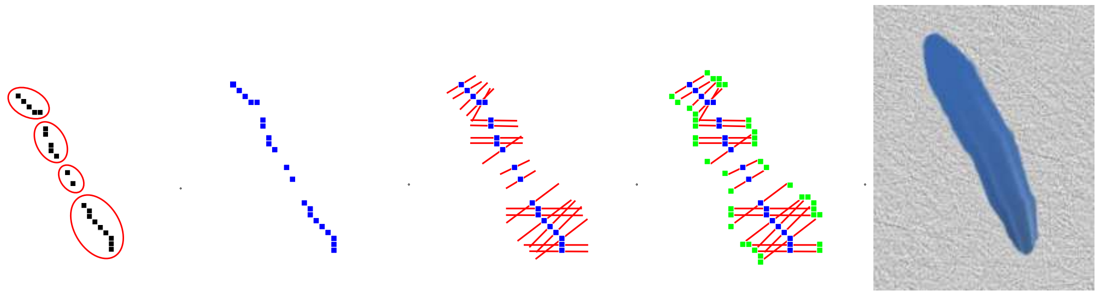

Figure 7.2. An example of the method for drawing strokes based on line lists. First, points are grouped into tokens. Second, tokens are grouped into a list of points. Third, for every point a normal line is found. Forth, based on the normal directions and the width at each point edge points for the stroke 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.

7.2 Brush Effects

Painting effects such as brush artifacts, paint mixing between layers, and stroke connection are common to both rendering methods. Brush artifacts are simulated by uniformly preterbing the brush color examples are shown in Figure 7.3. 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.

Visual effects of the grouping algorithms discussed in the previous section are demonstrated in Figure 7.4. 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 as seen in Figure 7.5.

7.3 Under Painting

Under painting is simulated by allowing the user to render strokes on top of another image. Meyer [26], Hertzmann [17] and Shiraishi, et al. [38] discuss under painting in their work. Meyer rendered brush strokes on an a background image. Hertzmann and Shiraishi, 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 Shiraishi. In addition the under painting can be used for artistic effect as in

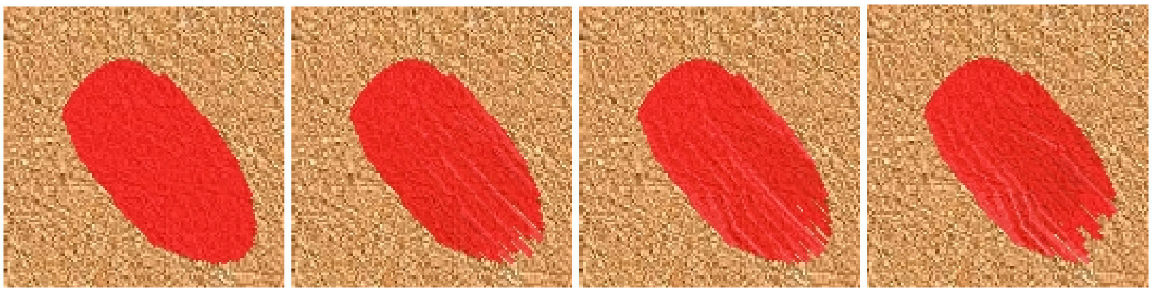

Figure 7.3. Depending on the viscosity of the paint a ridge artifact of varying height can be created when the paint is allowed to dry. This effect is simulated by adding light and dark bands to the stroke. A dry brush effect is simulated by halting the drawing of the brush stroke in a random manner. In the system these effects are adjustable by the user.

Figure 7.4. 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.

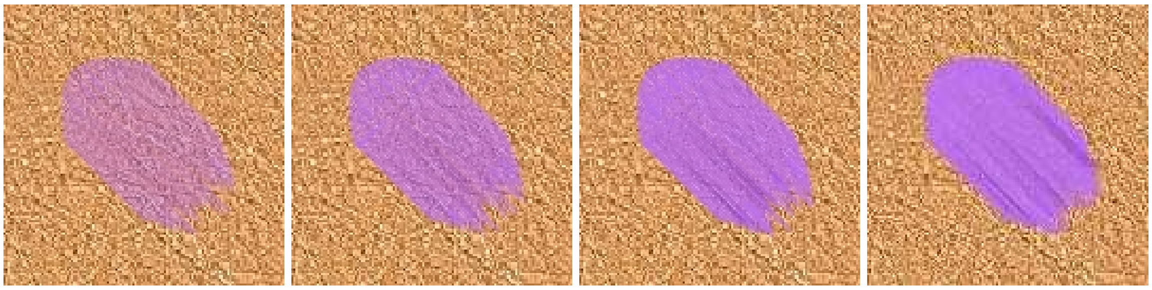

Figure 7.5. Alpha blending is used to simulate paints of various opacity. Paints with a high opacity will show the underlying substrate while paints with a low opacity will block the view of the substrate. The user adjustable Alpha parameter controls the percentage of blending between the under painting and the brush strokes.

Figure 7.6. One can also use the output of the system as the input to another image allowing a painting to be built up in layers.

7.4 Effects of Segmentation

As discussed in Chapter 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 7.4.

7.5 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 underlying 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. In this manner 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 Figure 7.7.

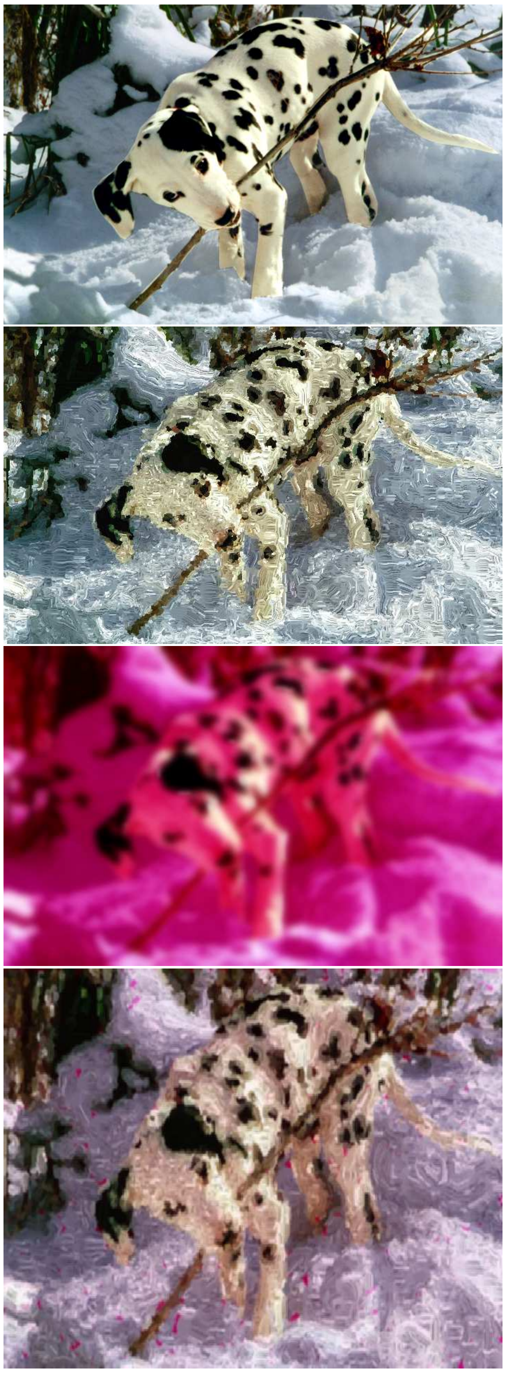

Figure 7.6. Under painting is a method used by artists to block in colored regions of a painting. An under painting is simulated by allowing the user to render strokes on top of another image. This example shows a source image. Next a painting made from this image using the source image as an under painting is shown. 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 under painting. This technique can be expanded upon to create images with multiple layers.

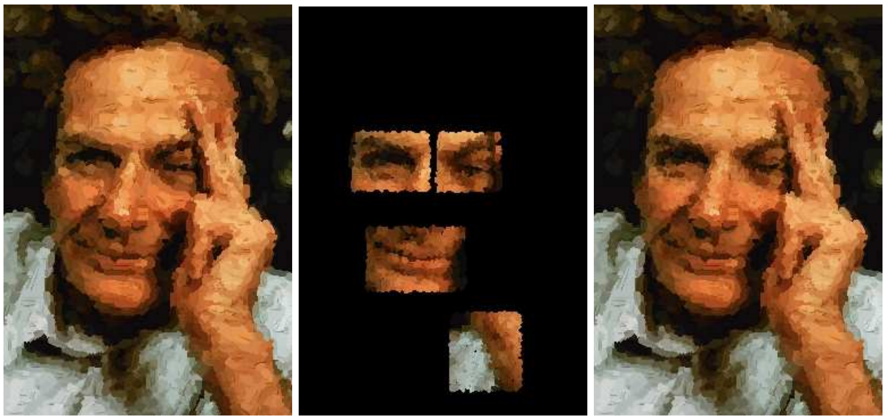

Figure 7.7. An example of user directed enhancement. The Feynman portrait is deemed by the user to lack detail in regions surrounding the eyes, mouth, and hand. The user selects these regions, and raise the segmentation level. New higher frequency strokes are drawn over the original strokes, hopefully improving the result.

CHAPTER 8

CONCLUSION AND RESULTS

The method described herein demonstrates the basic goal of keeping the number of brush strokes 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 for which 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.

Much of the following work is invited by the results that hve been presented. Productive future work could include improvements to every stage of the algorithm. Better segmentation, such as that given by anisotropic diffusion [44], would give immediate improvements in linking brush strokes 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 the algorithm which currently must account for noisy input. A simple improvement would be more sophisticated ordering of brush strokes, such as optimizing order based on edge correlation with the original image. Once brush stroke order is known, more physically-based paint-mixing would give a look more reminiscent of oil painting. The system could benefit from a user-assisted stage at the end to improve brush stroke ordering. An estimate of foveal attractors in the image could allow brush stroke size to be varied with probable viewer interest. Most challenging, the method could probably be extended to animated sequences, using time-continuous brush-stroke maps to ensure continuity. However, it is not clear how such animated sequences would look, or to what extent they would be useful. Perhaps the most exciting potential future effort is to create actual stroke-based hard copy using robotic or other technology. Additional examples of paintings made using this system are shown in Figures 8.1, 8.2, and 8.3.

Figure 8.1. An example of a painting of the authors wife's cat Molly.

Figure 8.2. An example of a painting of a vacation photograph taken at Monument Valley.

Figure 8.3. An example of a still life painting. Source image courtesy of the Cornell Program of Computer Graphics.

REFERENCES

[1] ARNHEIM, R. *Art and Visual Perception: A Psychology of the Creative Eye*. University of California Press, 1974.

[2] ARNHEIM, R. *The Power of the Center*. University of California Press, 1988.

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

[4] BETHERS, R. *Composition in Pictures*. Pitman Publishing Corporation, 1964.

[5] BLANZ, V., TARR, M. J., AND BULTHOFF, H. H. What object attributes determmine canonical veiws. *Perception 28*, 5 (1999), 575–600.

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

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

[8] CHIN, N., AND FEINER, S. Near real-time shadow generation using bsp trees. *Computer Graphics (Proceedings of SIGGRAPH 89) 23*, 3 (July 1989), 99–106. Held in Boston, Massachusetts.

[9] CLIFTON, J. *The Eye of the Artist*. North Light Publishers / Westport Conn., 1973.

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

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

[12] 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.

[13] EDELMAN, S., AND BULTHOFF, H. Oreientation dependence in the recognition of familiar and novel views of three-dimensional objects. *Vision Research 32*, 12 (1992), 2385–2400.

[14] 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.

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

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

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

[18] HORTON, W. Top ten blunders by visual designers. *Computer Graphics 29*, 4 (Nov. 1995), 20–24.

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

[20] HUNTER, F., AND FUQUA, P. *Light Science and Magic: An Introduction to Photographic Lighting*. Focal Press, 1997.

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

[22] KARP, P., AND FEINER, S. Issues in the automated generation of animated presentations. *Graphics Interface '90* (May 1990), 39–48.

[23] KAWAI, J. K., PAINTER, J. S., AND COHEN, M. F. Radioptimization - goal based rendering. *Proceedings of SIGGRAPH 93* (August 1993), 147–154. ISBN 0-201-58889-7. Held in Anaheim, California.

[24] LESTER, P. M. Digital literacy: Visual communication and computer images. *Computer Graphics 29*, 4 (Nov. 1995), 25–27.

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

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

[27] PALMER, S., ROSCH, E., AND CHASE, P. Canonical perspective and the perception of objects. *Attention and Performance 9* (1981), 135–151.

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

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

[30] RAMACHANDRAN, V., AND HIRSTEIN, W. The science of art a neurological theory of aesthetic experience. *Journal of Consciousness Studies 6*, 6-7 (1999), 15–51.

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

[32] SALISBURY, M., ANDERSON, C., LISCHINSKI, D., AND SALESIN, D. H. Scale-dependent reproduction of pen-and-ink illustrations. *Proceedings of SIGGRAPH 96* (August 1996), 461–468. ISBN 0-201-94800-1. Held in New Orleans, Louisiana.

[33] SALISBURY, M. P., ANDERSON, S. E., BARZEL, R., AND SALESIN, D. H. Interactive

pen-and-ink illustration. *Proceedings of SIGGRAPH 94* (July 1994), 101–108. ISBN 0-89791-667-0. Held in Orlando, Florida.

[34] SALISBURY, M. P., WONG, M. T., HUGHES, J. F., AND SALESIN, D. H. Orientable textures for image-based pen-and-ink illustration. *Proceedings of SIGGRAPH 97* (August 1997), 401–406. ISBN 0-89791-896-7. Held in Los Angeles, California.

[35] SANDER, F. Gestaltpsychologie und kunsttheorie. ein beitrag zur psychologie der architektur. *Neue Psychologische Studien 8* (1931), 311–333.

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

[37] SELIGMANN, D. D., AND FEINER, S. Automated generation of intent-based 3d illustrations. *Computer Graphics (Proceedings of SIGGRAPH 91) 25*, 4 (July 1991), 123–132. ISBN 0-201-56291-X. Held in Las Vegas, Nevada.

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

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

[40] SOLSO, R. L. *Cognition and the Visual Arts*. MIT Press/Bradford Books Series in Cognitive Psychology, 1999.

[41] 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.

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

[43] TUMBLIN, J., AND RUSHMEIER, H. E. Tone reproduction for realistic images. *IEEE Computer Graphics & Applications 13*, 6 (November 1993), 42–48.

[44] 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.

[45] VERFAILLIE, K., AND BOUTSEN, L. A corpus of 714 full-color images of depth-rotated objects. *Perception and Psychophysics 57*, 7 (1995), 925–961.

[46] WEI HE, L., COHEN, M. F., AND SALESIN, D. H. The virtual cinematographer: A paradigm for automatic real-time camera control and directing. *Proceedings of SIGGRAPH 96* (August 1996), 217–224. ISBN 0-201-94800-1. Held in New Orleans, Louisiana.

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

[48] WINKENBACH, G., AND SALESIN, D. H. Rendering parametric surfaces in pen and ink. *Proceedings of SIGGRAPH 96* (August 1996), 469–476. ISBN 0-201-94800-1. Held in New Orleans, Louisiana.

[49] WINKENBACH, G., AND SALESIN, D. H. Computer-generated pen-and-ink illustration. *Proceedings of SIGGRAPH 94* (July 1994), 91–100. ISBN 0-89791-667-0. Held in Orlando,

Florida.

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

[51] ZAKIA, R. D. *Perception and Imaging*. Focal Press Publications, 1997.