


國立交通大學

資訊科學系

碩士論文



以感知為基礎的非寫實描繪架構：
模擬映像派畫作

A Perceptually-Based NPR Framework for Synthesizing
Impressionist Paintings

研究生：祝琪

指導教授：施仁忠 教授

中華民國九十四年六月

以感知為基礎的非寫實描繪架構：模擬映像派畫作

研究生: 祝琪

指導教授: 施仁忠 教授

國立交通大學資訊科學系



呈現畫家風格描繪演算法已經吸引很多學者的注意，不過，目前的演算法既不是有系統的也不易評估。在這本論文中，我們提出了一種以感知為基礎用於呈現畫家風格描繪演算法的非寫實描繪架構。這個架構由三個子系統組成：原始物件映射系統、描繪系統及畫作實作系統。原始物件映射系統負責各種場景物件(二維區域，一維曲線和零維點)之間的映射；描繪系統負責將各種場景物件用畫家風格來描繪；畫作實作系統給予真正的畫作物理模擬。每個子系統對映到 C++ 中的一個抽象類別，使用者可以繼承這些類別來實作出自訂的風格。使用這個框架，呈現不同畫家風格的描繪可以在同一個系統中被實現。另外，這個架構的劃分是基於真實畫家的作畫過程，所以其結果的產生更具有美學價值。在本論文中，我們藉由運用此架構來合成印象派畫作以顯示它的效力。

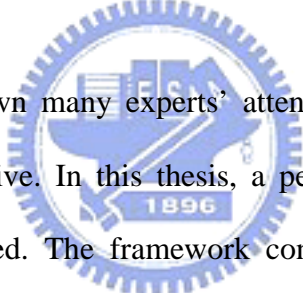
A Perceptually-Based NPR Framework for Synthesizing Impressionist Paintings

Student: Qi Zhu

Advisor: Dr. Zen-Chung Shih

Department of Computer and Information Science
National Chiao Tung University

ABSTRACT

The logo of National Chiao Tung University is a circular emblem with a gear-like border. Inside the circle, there is a stylized building and the year '1896' at the bottom. The text 'NCTU' is visible at the top of the inner circle.

Painterly rendering has drawn many experts' attention. However, the state of the art is neither systematic nor evaluative. In this thesis, a perceptually-based NPR framework for painterly rendering is presented. The framework consists of three sub-systems: primitive mapping system, rendering system and mark system. The primitive mapping system responds for mapping between various scene primitives (2D region, 1D line and 0D point); the rendering system takes charge of synthesizing different painterly styles based on various scene primitives; the mark system gives the actual physical implementation of the strokes generated by the rendering system. Each sub-system is written as an abstract class in C++ language which can be inherited by user to do some custom control. Using this framework, different painterly algorithm can be implemented in this system. Besides, the division of this framework is based on the actual process of painters which means the results generated can be more evaluative in esthetics aspect. In this thesis, Impressionist paintings are synthesized by this framework to show its effectiveness.

Acknowledgements

First of all, I would like to thank to my advisor, Dr. Zen-Chung Shih, for his help and supervision in this work. Also, I thank for all the members of Computer Graphics & Virtual Reality Lab for their comments and instructions. Finally, special thanks go to my family, and the achievement of this work dedicated to them.



Contents

摘 要	I
ABSTRACT	II
ACKNOWLEDGEMENTS	III
CONTENTS	IV
LIST OF TABLES	VI
LIST OF FIGURES	VII
CHAPTER 1 INTRODUCTION	1
1.1 MOTIVATION	1
1.2 OVERVIEW	3
1.3 THESIS ORGANIZATION	6
CHAPTER 2 RELATED WORKS	7
2.1 PAINTERLY RENDERING	7
2.2 MARK SYSTEM	9
2.3 NPR FRAMEWORK	10
CHAPTER 3 PRINCIPLES OF MAKING IMPRESSIONIST PAINTINGS	11
3.1 QUICK DRAWING	11
3.2 SELECTION OF COLOR	12
3.3 USING OF TACHE	14
CHAPTER 4 PRIMITIVE MAPPING SYSTEM	16
4.1 EXTENDEDNESS	16
4.2 SELECTION OPERATORS	20
4.3 CONVERSION OPERATORS	22
4.4 SHAPE MODIFIER	23
4.4.1 <i>Extract 1D Primitive</i>	23
4.4.2 <i>Modify Shape</i>	25
CHAPTER 5 RENDERING SYSTEM	27
5.1 BASIC RENDERING FUNCTION	28
5.2 QUICK DRAW RENDERING FUNCTION	29
5.3 POINTILLIST RENDERING FUNCTION	30

CHAPTER 6 MARK SYSTEM	33
6.1 BRUSH MODEL	34
6.1.1 <i>Stroke Cross Section</i>	34
6.1.2 <i>Stroke Path</i>	39
6.2 BRISTLE AND CANVAS INTERACTION.....	42
6.3 OIL PAINTING EFFECTS	42
CHAPTER 7 RESULTS.....	47
7.1 ORDINARY AND ADVANCE LAYERING	47
7.2 QUICK DRAWING	53
7.3 POINTILLIST	57
CHAPTER 8 CONCLUSIONS AND FUTURE WORKS	63



List of Tables

TABLE 1 RESULT STATISTICS.....62



List of Figures

FIGURE 1 THE PROPOSED FRAMEWORK.....	5
FIGURE 2 SUNRISE IMPRESSION, CLAUDE MONET.	12
FIGURE 3 BLUE AND ORANGE ARE USED TOGETHER TO HEIGHTEN TONE AND SUGGEST BRILLIANCE AND VIBRANT CONTRAST. (A) PIERRE-AUGUSTE RENOIR, <i>BOATING ON THE SCEINE</i> , DETAIL. (B) CLAUDE MONET, <i>LAVACOURT UNDER SNOW</i> , DETAIL.....	14
FIGURE 4 CLAUDE MONET, THE GARE SAINT LAZARE, DETAIL PHOTOGRAPHED IN RAKING LIGHT.....	15
FIGURE 5 REPRESENTATION OF EXTENDEDNESS. SPAN IS REPRESENTED AS A PAIR OF LINES WITH SAME COLOR. IF THE SPAN IS TOO SMALL, THESE TWO LINES ARE MERGED.	18
FIGURE 6 EXTENDEDNESS AND SHAPE DIRECTIONS. (A) EXTENDEDNESS; (B) SHAPE DIRECTION WITH SPANS OF EXTENDEDNESS NOT BEING REVERSED; (C) SHAPE DIRECTION WITH SPANS OF EXTENDEDNESS BEING REVERSED.	20
FIGURE 7 EXTRACT 1D PRIMITIVE FROM 2D PRIMITIVE. TWO SPAN ARE CHOSEN: [135,200,0.56], [319,359,0.76]. THE BLUE LINE REPRESENTS THE 1D PRIMITIVE.	23
FIGURE 8 EXTRACT BOUNDARY LINE FROM 2D PRIMITIVE. (A) INPUT 2D PRIMITIVE; (B) EXTRACTED BOUNDARY SPLINE (RED LINE) WITH BOUNDARY POINTS (YELLOW POINTS); (C) PART OF ORIGINAL BOUNDARY POINTS (594 POINTS); (D) PART OF TRIMMED BOUNDARY POINTS (411 POINTS); (E) SPLINE AND SELECTED CONTROL POINTS.	25
FIGURE 9 MODIFIED SHAPE OF BOUNDARY LINE. MODIFY FUNCTION: SINE FUNCTION WITH AMPLITUDE 0.05 AND FREQUENCY 10.....	26
FIGURE 10 SYNTHESIZED IMAGES BY BASIC RENDERING FUNCTION WITH FOUR LAYERS.	29
FIGURE 11 QUICK DRAW OF A GOOSE, THREE LAYERS.	30
FIGURE 12 POINTILLIST RENDERING FUNCTION. (A) ORIGINAL IMAGE. (B) SYNTHESIZED RESULT OF FIRST POINTILLIST RENDERING FUNCTION. (C) SYNTHESIZE RESULT OF SECOND POINTILLIST RENDERING FUNCTION.	32
FIGURE 13 MARK SYSTEM ARCHITECTURE.	34

FIGURE 14 (A) ROUND AND (B) FLAT STROKE CROSS SECTIONS CONSIST OF LOW SAMPLING POINTS (ROUND: 16 X 8; FLAT: 16 X 16).	35
FIGURE 15 BRISTLE DISTRIBUTION IN ROUND STROKE CROSS SECTION.	35
FIGURE 16 THE CORRESPONDENCE BETWEEN BRISTLE INDEX AND BRISTLE LOCATION. INDICES OF THE FIRST ELEVEN BRISTLE ARE SHOWN IN THIS FIGURE.	36
FIGURE 17 TWO SET OF SAMPLING RESULTS, EACH CONSISTS OF TWO SAMPLING (16X8, 32X16). (A) UNIFORM SAMPLING OF <i>RADIUS_RATIO</i> RESULTS UN-UNIFORM DISTRIBUTION OF BRISTLES; (B) TRANSFER <i>RADIUS_RATIO</i> BY POWER FUNCTION $y = x^{0.7}$ RESULTS UNIFORM DISTRIBUTION OF BRISTLES.	37
FIGURE 18 BRISTLE DISTRIBUTIONS IN FLAT STROKE CROSS SECTION. $ AB = ED $.	38
FIGURE 19 THE CORRESPONDENCE BETWEEN BRISTLE INDEX AND BRISTLE LOCATION. INDICES OF THE FIRST SEVEN BRISTLE ARE SHOWN.	39
FIGURE 20 CARDINAL SPLINE CONSTRUCTED BY FIVE CONTROL POINTS WITH DIFFERENT SMOOTH PARAMETERS (A) 10 (B) 2 RESPECTIVELY.	40
FIGURE 21 TESSELLATED STROKES WITH DIFFERENT TESSELLATION SCALE. (A) ROUND CROSS SECTION WITH TESSELLATION SCALE 8. (B) ROUND CROSS SECTION WITH TESSELLATION SCALE 1; (C) FLAT CROSS SECTION WITH TESSELLATION SCALE 8; (D) FLAT CROSS SECTION WITH TESSELLATION SCALE 1.	41
FIGURE 22 REAL OIL PAINTING STROKES. (A) PURE COLOR INTENSITY VARIANT RANDOM. (B) HEIGHT GAP EFFECT RANDOM.	43
FIGURE 23 TWO RANDOM SYSTEMS: (A) PLAIN STROKE. (B) PURE COLOR INTENSITY VARIANT EFFECT. (C) PLAIN HEIGHT. (D) HEIGHT GAP EFFECT.	46
FIGURE 24 ORIGINAL INPUT IMAGE.	48
FIGURE 25 FINAL IMAGE COMPOSED OF FIVE LAYERS. THE WHOLE IMAGE IS TREATED AS ONE PRIMITIVE.	49
FIGURE 26 FINAL IMAGE CONSTRUCTED BY FIVE STAGES. FIVE PRIMITIVES: ENTIRE IMAGE, SKY, GRASS, HAYSTACK AND SHEEP.	51
FIGURE 27 ORIGINAL INPUT IMAGE.	54
FIGURE 28 FINAL IMAGE COMPOSED OF TWO STAGES. FOURTEEN PRIMITIVES: ENTIRE IMAGE AND THIRTEEN GEESE.	55
FIGURE 29 ORIGINAL INPUT IMAGE.	58
FIGURE 30 FINAL IMAGE COMPOSED OF TWO STAGES. SIX PRIMITIVES: SKY, TREE, HAYSTACK, GRASS, GRASS SHADOW, AND ROAD.	59

FIGURE 31 FINAL IMAGE COMPOSED OF TWO STAGES WITH ANOTHER POINTILLIST STYLE. SIX
PRIMITIVES: SKY, TREE, HAYSTACK, GRASS, GRASS SHADOW, AND ROAD.61



Chapter 1

Introduction

1.1 Motivation



Painterly rendering has been studied for many years. A typical painterly rendering problem is to take an ordinary image (probably generated from a digital camera) as input and generate another image as output which represents a particular painting style. The input can also come from other format, e.g. 3D polygon mesh, which provides more information. Usually, such algorithms are hard-wired to their inputs. Thus, it is difficult to integrate these algorithms. Besides, these algorithms generate various painting style by changing parameters. However, these parameters are related to their implementations rather than to the painting style. The consequence is: a. changing these parameters is not intuitive; b. the esthetics value of the results is questioned; c. the integration of different algorithms is not easy. Due to these limitations, a general framework for painterly rendering is required which is the goal of this thesis.

The framework contains three sub-systems: primitive mapping system, rendering system, and mark system. This is inspired by the book “Art and Representation” written by John Willats [19]. As Willats wrote in this book, the goal of this book is to “describe the representational systems in pictures and the functions of these systems, independently of any historical or developmental consideration”. Willats divided the process of painting into two systems: the drawing system and the denotation system. The drawing system maps spatial relations in the scene into corresponding relations in the picture (e.g. perspective projection) while the denotation system maps scene primitives into corresponding picture primitives, such as regions, lines and points.

Recently, Frédo Durand [3] extended Willats’ framework into four sub-systems: spatial system, primitive system, attribute system and mark system. The spatial system is similar to Willats’ drawing system. The primitive system maps primitives in the object space to those in the picture space. The attribute system assigns visual properties to picture primitives. The mark system implements primitives in location defined by spatial system with attributes defined by attribute system.

The framework presented in this thesis is similar to Durand’s work. The spatial system is omitted currently – perspective projection is used. The mark system is similar to Durand’s mark system. We merge Durand’s primitive system and attribute system to one, the primitive system and add in the rendering system. The reason is that assigning visual properties to picture primitives is different from depicting it. The rendering system deals with how to depict a picture primitive. Besides, we think primitive mapping is not simply choosing among different primitives. It also deals with mapping of attributes in primitives. For example, impressionist prefers high tone pure color and this is suitable in primitive mapping – map ordinary color in primitive to high tone pure color.

1.2 Overview

As mentioned before, painterly rendering algorithm has a variety of inputs. To construct a framework, we should unify these inputs. We define the input of our framework contains a hierarchy of object. Each object which contains the following information: object information (such as object name, type, importance, etc), object relation information (such as shadow object, parent object, child objects, etc) and primitive information (each object contains several primitives, in this thesis we only deal with two dimensional primitives which are 2D region, 1D line and 0D point).

There are a lot of algorithms dealing with converting raw inputs to our input. For example, image segmentation algorithm can extract objects from an image while the polygon mesh input implicitly has this information. Line detection algorithms, for example, silhouette edge detection algorithms, and crease edge detection algorithms, can extract 1D primitive from polygon mesh input. Actually, a lot of NPR algorithms are dealing with this converting operation. However, their algorithms are hard-wired to the way of extracting information from input. It is hard to integrate such algorithms. Besides, it is not intuitive for other field experts who want to give guidelines to these algorithms. It is common because computer experts usually don't know artistic rules well. We separate the input from our framework. This let us focus on synthesizing painterly rendering effects.

When the input is available, it is fed to primitive mapping system. A *Primitive* object has two essential properties: shape and color. As a framework, we do not restrict their formats. We use the most general format: shape is represented as a two-dimensional mask and color is represented as a two-dimensional color buffer in which pixels use the RGB color model. Besides, every primitive has a property called as *Extendedness* [19]. It is a description of object shape properties based on human's perception of shape. We use Extendedness to

synthesize the quick drawing effect of Impressionist paintings. The primitive mapping system maps one primitive to another primitive. The mapping can be between the same primitive type, e.g. 2D region to 2D region; it also can be between different primitive types. Details of this part will be discussed in Chapter 4.

After primitive mapping is done, user can choose a subset of mapped primitives and sort them into a partial order which will be fed to the rendering system. The manipulation of mapped primitive sequence to the one fed to rendering system is not restricted by our framework. User can do whatever they like. In fact, this part involves artist's creation and their own opinion which can not be done by computer.

The rendering system responds to choose *Rendering Function* for each mapped primitive. Each rendering function takes mapped primitive as input. Besides, it can access global information – input and Canvas object which is used in mark system. It also can access other information. We do not restrict either the type of information or the amount of information. All extra information is treated as void pointer and the recognizing of these pointers is done in user defined custom rendering functions. The output of this system is a list of stroke definitions. Besides, rendering function can generate stroke definitions in several stages and the previous stage result (Canvas object) will affect the generation of next stage. Details of this part will be discussed in Chapter 5.

The stroke definitions generated by rendering system are fed to the mark system. Mark system in our framework is composed of two objects: stroke and canvas. Details of this part will be discussed in Chapter 6.

The following graph shows the whole framework:

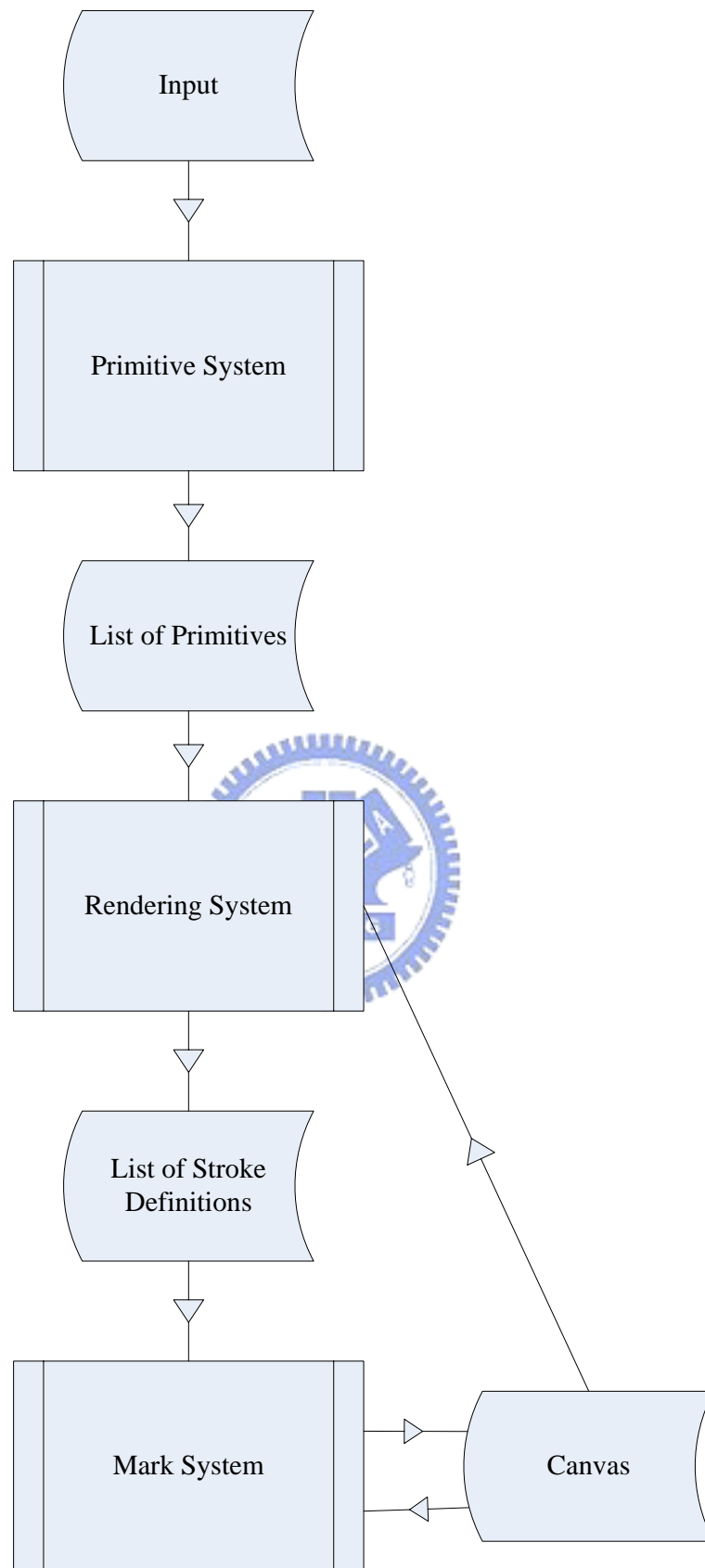


Figure 1 The proposed framework.

1.3 Thesis Organization

The rest of this paper is organized as follows. In Chapter 2, related works are presented. Principles of making Impressionist paints are introduced in Chapter 3. Chapter 4, 5, and 6 give detail explanations of each sub-system of our framework. The results are shown in Chapter 7. Finally, conclusions and future works are presented in Chapter 8.



Chapter 2

Related Works

2.1 Painterly Rendering

Painterly rendering algorithms have been studied for years. [6, 7, 8, 9, 10, 11, 12, 13, 14, 17, 20] The purpose of this kind algorithm is to synthesize images with painting effects provided by the given input. The input can be two-dimensional images or three-dimensional polygon meshes. For example, Wang [17] presents a method for converting a photo to a synthesized painting by an example painting. His algorithm uses a hierarchical patch-based approach to synthesize directional textures. Actually the synthesizing work is done by texture mapping. Our work is close to another branch of painterly rendering algorithms: stroke based painterly rendering algorithms. A classic paper was written by Haeberli [6]. Its main contribution is to provide a paradigm for painterly rendering: the synthesized image is formed by an ordered collection of brush strokes. Each of which has color, shape, size and orientation. By controlling these strokes, various painting effects can be easily created. As a matter of fact, a lot of painterly rendering algorithms follow this paradigm. Besides, Haeberli explored many possibilities above painterly rendering: the stroke definition may come from user's interaction (a mouse in his paper), it can also come from image processing algorithms on the input image.

There are at least four characteristics of stroke based painterly algorithms: a. Determination of stroke definition, i.e. how to choose stroke location, direction, color and shape; b. Input of algorithm; c. Interaction with user; d. Goal of algorithm, i.e. what painterly style does this algorithm try to synthesize. Litwinowicz [10] transforms a video clip into animations that have Impressionist effect. Strokes are clipped by edges detected in the source image in order to maintain silhouettes and other features of image. Stroke orientation is normal to the gradient direction of the original image. Strokes are also added and deleted from frame to frame by using optical flow fields to maintain temporal coherence. Hertzmann [8] presents a multi-layering painting algorithm. The synthesized image is composed of several layers which correspond to the practice of a painter. These layers start with a rough sketch drawn with large size stroke. Then, the stroke size becomes progressively smaller and is placed only in areas where the sketch differs from the source image. Hertzmann also presented an algorithm for generating long curved strokes. The stroke path will follow the normal of image gradient and ends when a maximal steps defined by user is reached. Besides, various painting styles can be synthesized by changing the parameters. Barbara [12] presents an algorithm for rendering animations in painterly style. The input is a three-dimensional polygon mesh. He models surfaces as three-dimensional particles. Each of which will be placed a stroke. The attribute of a stroke is determined by the geometric and lighting properties of surfaces. For example, color is directed from frame buffer; orientation is normal to image (current frame) gradient; size is from a user defined reference picture. Georges [20] uses “stroke texture” to simulate the principles of traditional pen-and-ink illustration in an automated rendering system. The input is also a three-dimensional polygon mesh. User can use lines to indicate the important part so that more strokes will be placed at this part.

As discussed above, most algorithms focus on characteristics a and b. Characteristic c is rare due to their goal is an automatic system. However, due to the complexity of artist's

painting process and weak power of computer artificial intelligence nowadays, the esthetics value of result is questioned. Hence, most algorithms just claim their results are like certain painting style without any proof. As mentioned by Hertzmann in his survey paper on stroke based rendering [9], aesthetic decisions made by artist is important otherwise all algorithms is useless. Thus, how can an artist get involved in synthesizing process is the key element of a success painterly algorithm. Unfortunately, most algorithm are hard-wired to their codes which neither provides much variation on possible painting styles nor suitable for non-computer-expert such an artist to guide the synthesizing process. This is why we want to develop a general framework.

2.2 Mark System

The physical implementation of a stroke is the mark system. Some algorithms just use a texture to represent a stroke while others use brush model. Brush model is also used in our approach. Strassmann [15] developed a one-dimensional brush model to synthesize Chinese Ink Painting or Japanese Sumi-e. He defines a one-dimensional array of bristles as the painting brush. Each bristle has its own properties such as ink supply and position. While painting, the brush moves along the path and leaves footprints on the canvas. Strassmann also defined several parameters to simulate painting styles. For example, the ink quantity is used to simulate dry brush effect. Weng [18] introduced a two-dimensional brush model. It is similar to Strassmann's except his use of an ellipse as the contact region. Bristles are distributed uniformly in the contact region. The advantage is elliptical contact region is better for turning effects. In this thesis, we adapt Weng's stroke model and add one more contact region, the rectangle. Both are commonly used in Impressionist paintings. In oil paintings, painter usually paints several layers to complete their work. Cassidy [2] uses fluid computation on

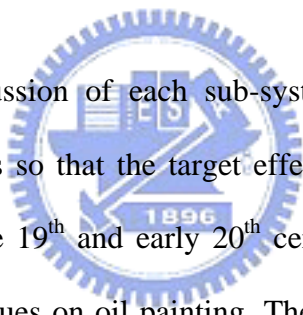
one wet layer and unlimited dry layers which are an ordered set of translucent glazes to simulate water color. Our mark system consists of one dry layer and one wet layer.

2.3 NPR Framework

Stéphane [4] presents a programmable interface to non-photorealistic line drawing. By given three-dimensional polygon mesh, a view map contains lines and their topology is extracted. After that, there are three user definable modules are applied: selecting, chaining and splitting. Each of these modules refines the lines which will be drawn on the final image. The advantage of this framework is much flexible to synthesize many line drawing styles. Many new effects are possible in this framework, such as advanced layering, control over stroke topology, stroke geometry displacement, and density control. Their idea comes from shading language in photorealistic rendering. Upstill [16] gives the spirit of shading language: “The key idea in the RenderMan Shading Language is to control shading, not only by adjusting parameters, but by telling the shader what you want it to do directly in the form of a procedure.” We think this is what current painterly rendering algorithms lack.

Chapter 3

Principles of Making Impressionist Paintings



Before the detailed discussion of each sub-system, we first introduce principles of making Impressionist paintings so that the target effects are clear. The impressionism took place in France during the late 19th and early 20th centuries by a group of artist who have similar approaches and techniques on oil painting. They challenged the traditional academic system which was controlled, deliberated and marked by an absolute precision of finish in their times. Oppose to the academic system, they prefer outdoor scenes and directly responds what they perceived in the weather condition right away, regardless of object's traditional shape. The Impressionist paintings have three characteristics: quick drawing, selection of color, and using of tache. All of these characteristics are for one good – capture the intangible fleeting effects of light and atmosphere in paintings.

3.1 Quick Drawing

The most remarkable characteristic of Impressionist painting probably is the sketching effects due to quick drawing. Because the Impressionists want to capture the constantly changing lighting effects outdoors, painting speed is essential if the artist try to capture a convincing effect. Thus, they should draw quickly: quickly and directly without attempting to develop the detail and careful finish. Another reason for sketching effects is Impressionist emphasize color instead of shape. They treat scene in front of them as a collection of color patch rather than objects. Thus, they usually ignore the true boundary of objects in their paintings. That's why they usually received the criticism such as “nothing but sketches”, “satisfied with a first impression” and praise such as “sincerity” and “truth”.

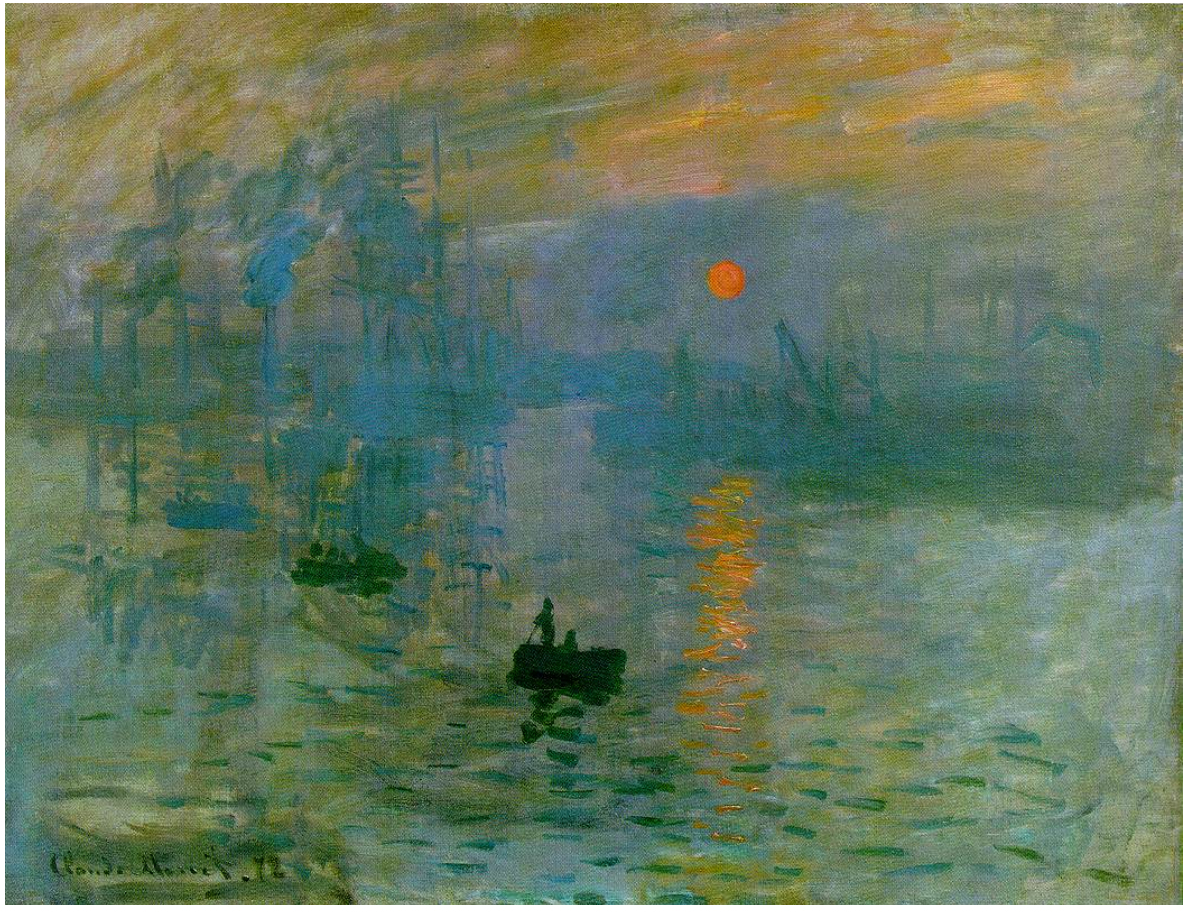


Figure 2 Sunrise Impression, Claude Monet.

3.2 Selection of Color

Impressionists emphasize the color. First of all, the color of priming is important to Impressionist. They often use the ground color to represent part of the finished painting. Then, Impressionists prefer using high tone pure color. Earth pigments were used with discretion. Moreover, complementary colors are preferred by Impressionist.

A common misconception is that the Impressionist painters exploited a process of 'optical mixing' by using separate touches of pure prismatic colors that fused in the eye of the beholder. To explain this clearly, we first discuss the relation between primary and complementary colors. Isaac Newton had shown in 1666 that white light could be separated into its component colors by means of a prism. However, this color mixture model is only suitable for additive color such as sun and monitor. In fact, color matching experiments in 17th century by Thomas Young and other experts in which three primary colors are placed on a disc and spin it fast show that the result is not white but only grey. The reason is that mixture of pigment belongs to subtractive or more complex mixture model. In fact, a strong color tends to irradiate its surroundings with its own complementary. Therefore if two complementary colors are placed together they can enhance each other. That's the true reason Impressionists prefer placing complementary colors adjacently. Another reason is using complementary colors to present vibrant struggling light rays outdoors. French poet Jules Laforgue wrote: "The Impressionist sees light as bathing everything not with a dead whiteness, but rather with a thousand vibrant struggling colors of rich prismatic decomposition".

Colored shadow features consistently and prominently appear in Impressionist paintings, especially those depicting sunlit scenes of snow and landscape. In summer landscapes purplish red is used for shadows of trees and dull violets are used for the shadows of rocks or across dusty roads. Manet said: "I have finally discovered the true color of the atmosphere. It's violet."

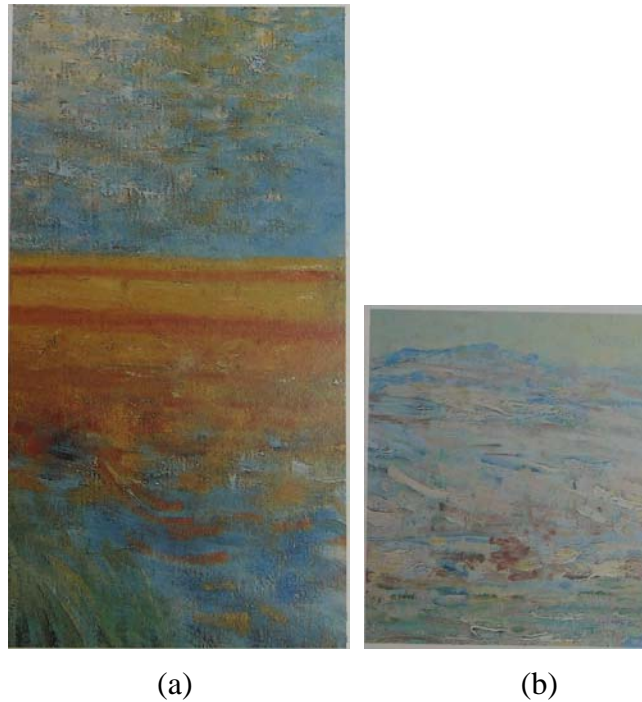


Figure 3 Blue and orange are used together to heighten tone and suggest brilliance and vibrant contrast. (a) Pierre-Auguste Renoir, *Boating on the Seine*, detail. (b) Claude Monet, *Lavacourt under Snow*, detail.

3.3 Using of Tache



Tache, the colored patch or stroke, is the most distinctive feature of Impressionist paintings. Unlike academic painting in which forms could be smoothly modeled, Impressionist defines forms in clearly differentiated patches of color which gave the effect of light surrounding and reflecting off objects rather than the specific shapes of the objects themselves. In fact, tache is not simply a color sensation. It has weight, thickness and texture. All of these affect the viewer's perception of its color. Whirling, flickering dashes of paint were perfectly suited for capturing the intangible qualities of light, air, cloud and vapor. However, it is impossible to categorize the types of tache marks made by Impressionist. It is difficult even to trace the development of a single painter.

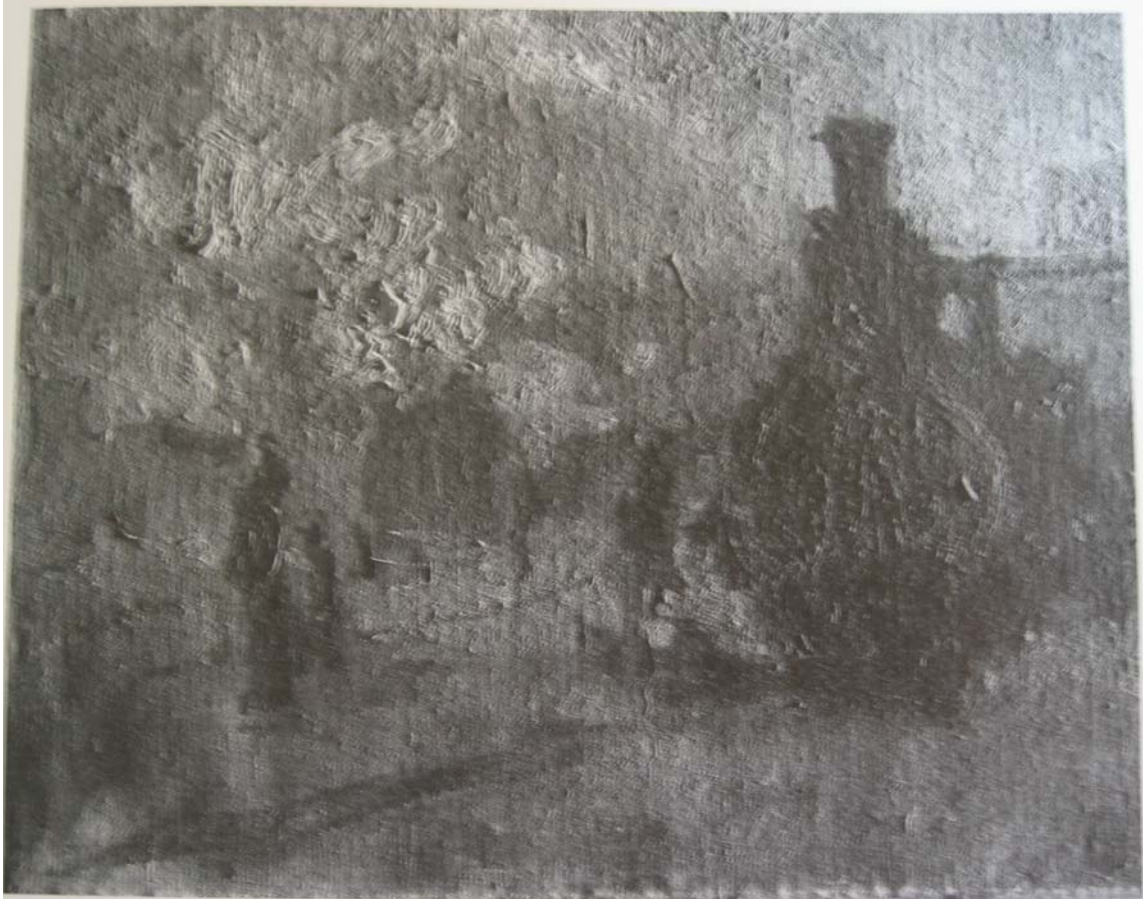


Figure 4 Claude Monet, The Gare Saint Lazare, detail photographed in raking light.



Chapter 4

Primitive Mapping System

A Primitive has two essential properties: shape and color. As mentioned in Chapter 1, shape is represented as a two-dimensional mask and color is represented as a two-dimensional color buffer in which pixels use the RGB color model. Besides, every primitive has a property called extendedness [19]. There are three kinds of primitives available in our framework: two-dimensional primitive, one-dimensional primitive, and zero-dimensional primitive. The basic information a one-dimensional primitive is the control point information, including location, size, and color. Zero-dimensional primitive has point information including location, size, and color.

The primitive mapping system maps one primitive to another primitive. There're four kinds of mapping operation in our frame work: selection, conversion, shape modifier, and color modifier. Selection operation enables user to select primitive in input. Conversion operation convert primitive of one type to another type. Shape modifier changes the shape property of primitive. Color modifier changes the color property of primitive.

4.1 Extendedness

Willat [19] use the concept of extendedness to describe human's perception of shape. The extendedness specifies the relative extensions of primitive in different directions in space. For example, a ball equally extends itself from its center to all direction. Thus its extendedness can be notated as 3_{111} where the suffix 111 implies the ball extends itself in all three main directions: X, Y, and Z axes. A plate extends itself in two directions: X and Y axes. Thus its extendedness is 2_{11} . A stick extends itself only in one direction. Thus its extendedness is 1_1 . The number 3, 2, and 1 before extendedness suffix stand for the dimensional index of a primitive. It determines the number of dimensions within which a primitive can potentially be extended.

We extend Willat's extendedness concept and use it to synthesize Impressionist's quick drawing effect. The extendedness in our framework is defined as a list of spans each of which has start direction, end direction, and intensity. The following two figures show the representation of extendedness of a 2D primitive man and a 2D primitive goose. The representation is the form of $[a, b, c]$ where a and b are the start and end direction in degree, c is the intensity normalized in $[0, 1]$. The primitive man has three spans: $[83, 100, 0.77]$, $[252, 264, 0.82]$, $[278, 291, 0.76]$ and the primitive goose has five spans: $[135, 200, 0.56]$, $[202, 211, 0.54]$, $[252, 252, 0.53]$, $[292, 292, 0.53]$, $[319, 359, 0.76]$.



Figure 5 Representation of extendedness. Span is represented as a pair of lines with same color. If the span is too small, these two lines are merged.

To find the extendedness, we first find the center of the primitive. Then we sample all direction from the center and calculate its intensity. At last we merge adjacent directions into spans according their intensities and select several major spans among them. The detail algorithm is follows:

```

Find_Extendedness(Primitive  $p$ )
{
  // 1. Find center of primitive.
   $center \leftarrow$  find center of primitive  $p$ 
  // 2. Sample all direction.
  for (all points  $po$  in primitive) {
     $direction \leftarrow$  find angle between line  $po$ - $center$  and X-axis
     $intensity\_by\_direction[direction]++$ 
  }
  for (all directions:  $d$  is from 0 to 359) {
     $begin\_direction \leftarrow d$ 
     $gap\_left \leftarrow MERGE\_GAP\_SIZE$ 
     $spanned \leftarrow 0 : intensity \leftarrow 0$ 
    // 3. Merge adjacent direction.
  }
}

```

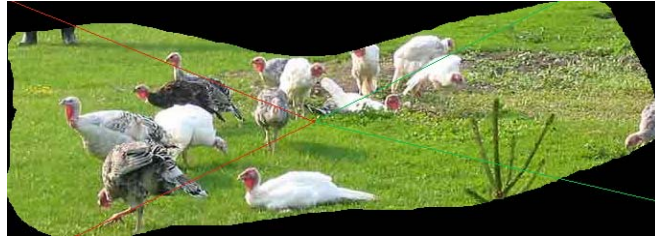
```

while (gap_left>0 && (MERGE_MAX_SIZE<0 || spaned<MERGE_MAX_SIZE)
{
    if (intensity_by_direction[d] < max_intensity * LEN_THRESHOLD)
        gap_left--
        intensity ← intensity + intensity_by_direction[d]
        spaned++ : d++
    }
    d--
    // 4. Trim un-necessary merge.
    while (intensity_by_direction[d] < max_intensity * LEN_THRESHOLD) d--
        end_direction ← d
        intensity ← intensity / (end_direction - begin_direction)
    // 5. Generate one span.
    one major span generated: [begin_direction, end_direction, intensity]
}
}

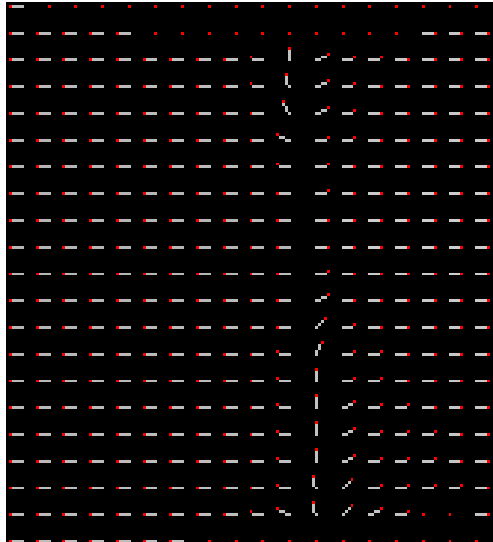
```

Algorithm A

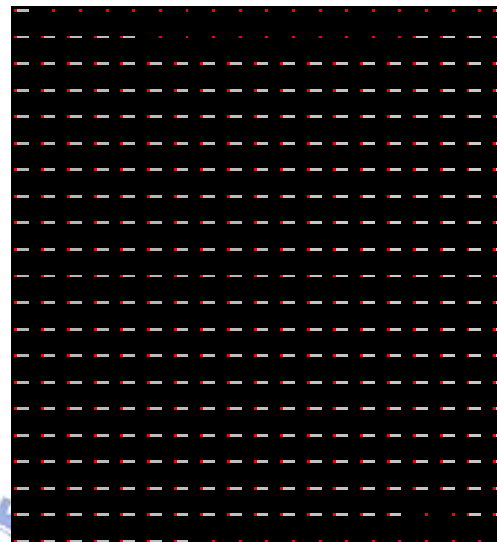
Three parameters mentions in figure 6 are *MERGE_GAP_SIZE*, *MERGE_MAX_SIZE*, and *LEN_THRESHOLD* whose values are 5, -1, and 0.5 in our implementation respectively. They are suitable for many kinds of shapes. After the extendedness is generated, it will be used to generate “shape direction” for every point in primitive. For points inside major span, their shape directions are the same as their span direction. For points outside major span their shape direction are the blended result of the directions of their adjacent spans. However, before we generate shape direction, we should reverse a few span directions. Consider the following situation:



(a)



(b)



(c)

Figure 6 Extendedness and shape directions. (a) Extendedness; (b) Shape direction with spans of extendedness not being reversed; (c) Shape direction with spans of extendedness being reversed.

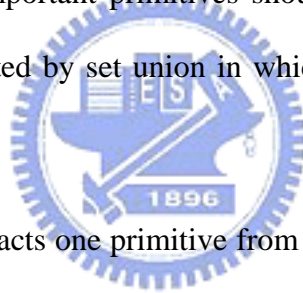
Shape direction in Figure 6 is not natural. In fact, shape direction of the rectangle-like shape in Figure 6 should be horizontal at every point. Thus, we should reverse span direction if the angle between two span direction A and B is larger than $\pi - A$ and B.

4.2 Selection Operators

There are four kinds of selection operators: selecting, merging, subtracting and sorting. The selecting operator takes a primitive as input and decides whether to select it according to the information contained in the primitive. For example, selection may depend on the importance value or type of an object where the primitive belongs to. Users can implement their own selection operator based on complex functions. Several build-in selection operators

are available such as selection by object importance, selection by object id, and selection by object type.

Usually, painters usually paint the canvas in several layers. They use large cross-section stroke in early layers and small cross-section size stroke in latter layers. Besides, in early layers, objects of a scene are treated as one object. While in latter layers, important objects are treated repeatedly. Some painterly rendering approach simply places stroke on canvas from large cross-section stroke to small cross-section stroke referring to the whole image [8]. In these algorithms, the whole image is treated as an object. We provide merging operator which takes two primitives as input and the merged result as output. Users are allowed to customize their layering behavior by merging unimportant primitives. For example, far primitives can merge together and near or important primitives should be treated separately. The build-in merging operator is implemented by set union in which the shape mask and color buffer of primitive are treated as sets.



Subtracting operator subtracts one primitive from another. It is clear that the background primitive should not contain the foreground primitive. However, in order to reduce user's work, the background primitive in input usually contains foreground primitive. Thus, we need subtraction operator. This operator is also used to co-operate some kind of rendering function. For example, the quick-draw drawing function needs that the primitive has never been drawn before. Thus, subtracting operator will be applied to another other primitive which contains this primitive. The build-in subtraction operator is implemented by set subtraction in which the shape mask and color buffer of primitive are treated as sets.

The order of primitive drawing heavily influences the result. We provide sorting operator to determine the order of primitives. The input of sorting operator is two primitives and the output is the comparing result: smaller or equal, bigger, and un-comparable. This will result in

a partial order of primitives in which primitives of the same anti-chain are in the same painting layer. There is no un-trivial build-in sorting operator because this operation involves the creativities of painters which should be fully determined by users.

4.3 Conversion Operators

Conversions between different types of primitives are necessary in some situations. For example, painter usually uses just a single stroke to depict a far object. A two-dimensional primitive should be converted to a one-dimensional primitive. Conversion operator deals with this kind of conversion. Since there are three kinds of primitives, there are six conversions totally: $2D \rightarrow 1D$, $2D \rightarrow 0D$, $1D \rightarrow 2D$, $1D \rightarrow 0D$, $0D \rightarrow 2D$, and $0D \rightarrow 1D$. In our framework, 1D and 0D primitives contain their corresponding 2D primitive information. In other words, the 2D primitive is the generalization of 1D and 0D primitive. Thus the conversion of $1D \rightarrow 2D$ and $0D \rightarrow 2D$ are trivial. The conversion of $1D \rightarrow 0D$ can be done by $1D \rightarrow 2D \rightarrow 0D$. Similarly, the conversion of $0D \rightarrow 1D$ can be done by $0D \rightarrow 2D \rightarrow 1D$. So, there're only two conversions need to be implemented: $2D \rightarrow 1D$ and $2D \rightarrow 0D$.

The key idea of conversion from 2D primitive to 1D primitive is to find the main axis (1D primitive) of a region (2D primitive) such that this main axis can represent the region. We use the information from extendedness to find this main axis. First, we choose an extendedness span with maximal value of area span multiplying intensity. Then, another extendedness span is chosen. The restriction is its value of area span multiplying intensity should be large as possible and the angle between these two spans should be larger than 90^0 . After these two spans are chosen, the main curve may grow little by little from the center of the primitive following the direction of two spans.

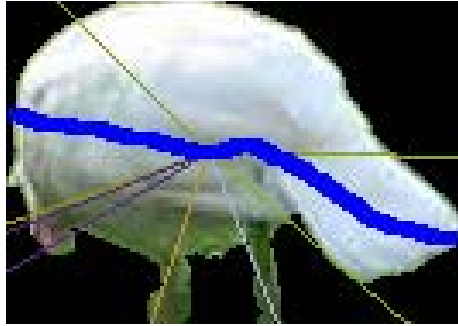
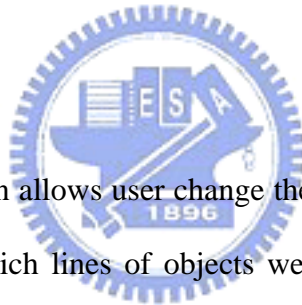


Figure 7 Extract 1D primitive from 2D primitive. Two span are chosen: $[135,200,0.56]$, $[319,359,0.76]$. The blue line represents the 1D primitive.

Conversion from 2D primitive to 0D primitive is easy. The center of 2D primitive is used to represent the center of 0D primitive. The radius of the bounding circle of 2D primitive is used to represent the size of 0D primitive. The color of 0D primitive is the average color of the 2D primitive.

4.4 Shape Modifier



Shape modifying operation allows user change the shape of a primitive. This is common in Van Gogh's painting in which lines of objects were twisted to represent painter's inner feelings.

4.4.1 Extract 1D Primitive

To reduce user's work, the input usually contains only two-dimensional primitives. Thus, we should extract one-dimensional primitive, i.e. lines, from two-dimensional primitive. There are two kinds of extracted 1D primitive: boundary line and internal line. The definition of boundary line is clear. The internal line will be discussed in Chapter 5.

Identify which point in 2D primitive belongs to boundary line is easy while how to sort them is not trivial. We have to sort these boundary points so that a subset of these points are able to be selected as the control points of cardinal spline. We randomly choose a boundary point and find its neighboring boundary point by following two sets of directions: (a) up, right

up, right, right down, down; (b) down, left down, left, left up, up. The neighboring boundary point is used to find another neighboring boundary point. If we can't find such a point, we change from one direction set to another direction set. This process stops when both two directions are tried and no un-visited boundary points found. However, this approach will not work if there exists unnecessary boundary points. Unnecessary boundary points are defined as removing them will still makes boundary line closed. Thus we must trim these points before sorting.

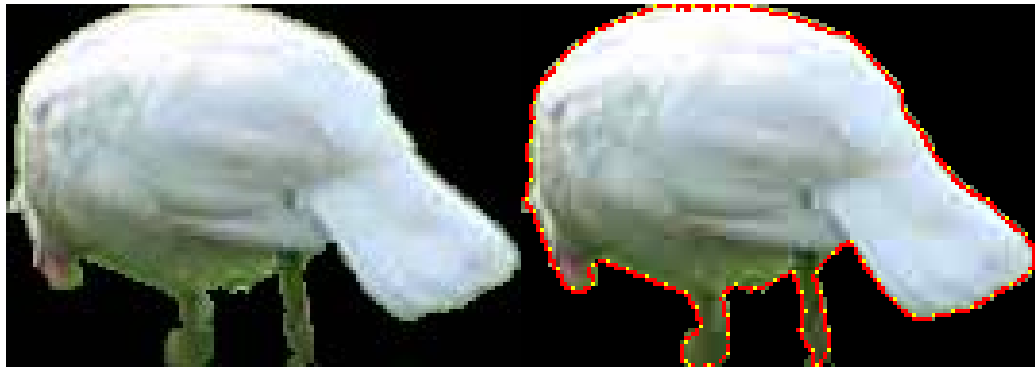
After sorting, a subset of boundary points are selected as control points used by Cardinal spline. An initial subset of boundary points is chosen. Then, boundary points are added to this subset if it can refine the spline with control points in current subset. The following algorithm extracts 1D primitive from 2D primitive:

```

Extract_1D_Primitive(Primitive2D p)
{
    // 1. Find original boundary points.
    un-sorted_boundary_points ← find boundary points
    // 2. Trim un-necessary boundary points.
    for (all points point in un-sorted_boundary_points)
        if (trim point will still make the boundary line be closed)
            trim point from un-sorted_boundary_points
    // 3. Sort boundary points.
    start_point ← choose a start boundary point
    add start_point to sorted_boundary_points
    while (true) {
        next_point ← un-visited neighbor boundary point of previous_next_point
        following current direction
        // direction set one { up, right up, right, right down, down }
        // direction set two { down, left down, left, left up, up }
        if (no next_point available and both two direction set are tried) break;
        else if (no next_point available) reverse direction
    }
    // 4. Choose a subset of boundary points to be the control points
    control_point_set ← Choose an initial subset of sorted boundary points
    while (spline by control_point_set can not be refined)
        add a boundary point to control_point_set between two control point if it can
        refine the spline
}

```

Algorithm B. Extract boundary line.



(a)

(b)



(c)

(d)



(e)

Figure 8 Extract boundary line from 2D primitive. (a) Input 2D primitive; (b) Extracted boundary spline (red line) with boundary points (yellow points); (c) Part of original boundary points (594 points); (d) Part of trimmed boundary points (411 points); (e) Spline and selected control points.

4.4.2 Modify Shape

After the 1D primitive of boundary lines are obtained, the shape of the spline can be modified. We define the modifying function take location of a point on spline as input and a

scalar as output. We move control point along the direction which is normal to its tangent direction by a distance controlled by this scalar. Changing the shape of a primitive will cause some new region belongs to the primitive. Thus, we should define the color distribution of this new region. Besides, the shape and color distribution of its neighbor primitive should be changed correspondingly. Otherwise holes between them will appear. Because the change of primitive shape is not large compared to the shape of primitive itself, the redistribution of color is simply done by enlarging original primitive and cutting it by the changed boundary line.

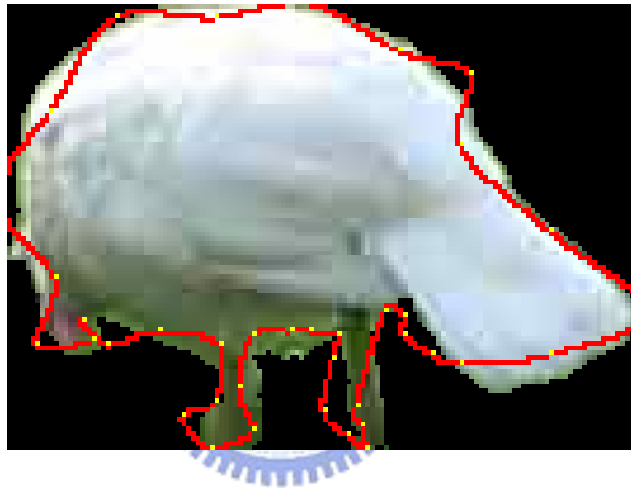


Figure 9 Modified shape of boundary line. Modify function: sine function with amplitude 0.05 and frequency 10.

Chapter 5

Rendering System

The rendering system responds for choosing rendering function for each mapped primitive. Each rendering function takes mapped primitive as input and generates stroke definitions. Stroke definition contains stroke path, stroke cross section per control points, initial bristle properties, and physical effect relative parameters. All of these are necessary to mark system. Besides, rendering function can access global information – input and Canvas object which is used in mark system. It can also access other information. The generation of stroke definition is divided into several stages. Stroke definitions generated from each stage are fed to mark system and its results will affect the next stroke definition generation stage.

We implemented several rendering functions: basic, quick draw, and pointillist. The basic rendering function places stroke on canvas from large stroke cross section size to small stroke cross section size according to the whole primitive. The stroke path direction is guided by primitive image orientation. The quick draw rendering function is used to synthesize quick drawing effect of Impressionist paintings. Primitives are drawn with fewer strokes as possible. The stroke direction usually follows the direction of object shape. We obtain direction of object shape from its extendedness. Pointillist rendering function is used to synthesize

pointillist painting style in neo-Impressionism. How to choose the color of each point is the key concern in pointillist rendering function.

5.1 Basic Rendering Function

The basic rendering function contains three phases. First, a series of brush sizes R_1, R_2, \dots, R_n are determined. This series of brush sizes corresponds to layers of painting from larger brush size to small brush size. For each painting layer, the brush size, canvas, and a blurred image from color buffer of primitives are used to generate stroke definitions. We use Gaussian blur to generate the blurred image and the deviation of Gaussian kernel is proportioned to brush size.

Second, when a layer is about to be painted a list of stroke locations is generated. Each of these locations will be used to generate one stroke definition. To generate these locations, the blurred image is divided into several grids. The number of grids is inversely proportioned to brush size. In each grid, we measure the average color difference between reference image and canvas. When the average difference is larger than some threshold, a stroke location will be generated in this grid. The stroke location will be located in the largest difference point in the grid.

Third, a stroke definition will be generated on the location calculated in previous phase. Stroke color is determined by the color of blurred reference image in stroke location. Stroke size is determined by brush size. Stroke path grows little by little and follows the normal direction of image gradient. Each growing process generates one control point. The growing process is stopped when one of the following two conditions hold: a. a maximal stroke length is reached; b. the difference between stroke color and blurred reference image color is larger than the difference between canvas color and blurred reference image color.



Figure 10 Synthesized images by basic rendering function with four layers.

5.2 Quick Draw Rendering Function

The quick draw rendering function is used to synthesize quick drawing effect of Impressionist paintings, e.g. figure 2. To synthesize this kind of effect, the following objective should be achieved. First, use as fewer strokes as possible to represent the foreground object. These strokes should reveal the shape of the object. The color distribution of foreground object is much less noticeable than its shape in synthesizing. Second, neither of shape and color of the background objects is noticeable in synthesizing.

To achieve the first objective, we change the stroke path generation process which is third phase in basic rendering function. Stroke path no longer follows the normal direction of image gradient. The blending direction of “shape direction” and normal of image gradient is used instead. The shape direction is calculated from primitive extendedness which is explained in Section 4.1. Besides, we use the stencil buffer of canvas to avoid overlapped strokes. To achieve the second objective, we blur the color buffer of primitive belonged to the background object. The blurring operation is done by blending each pixel’s color value with the mean color of the whole color buffer. The blurred color buffer is used as the input to the second phase in basic rendering function.

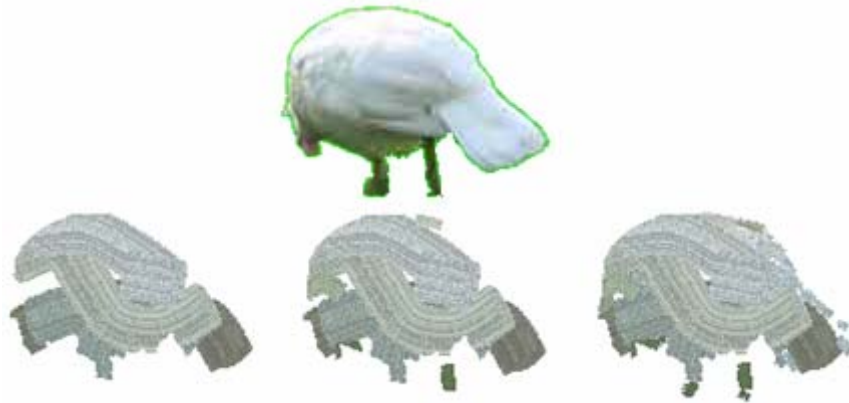


Figure 11 Quick draw of a goose, three layers.

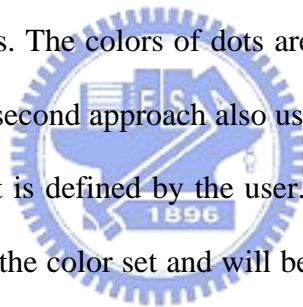
5.3 Pointillist Rendering Function

Pointillist rendering function is used to synthesize pointillist painting style in neo-Impressionism. In Pointillism, small strokes or dots of pigment are used to complete the whole painting. The sizes of colored dots are regular. The color of these dots represents the actual physical composition of the lighting on object surface. For example, a scene of grass in sunlight will be painted in the following steps. First, green pigment is used to present the inherent color of grass. Then, light blue dots are applied to represent the reflection of sunlight

on grass. Three kinds of color dots are used to represent the shadow area of grass: green, red, and light blue. Red is the complementary color of green and Impressionist usually use complementary color to represent object shadow.

Traditional pointillist algorithm use random saturation and intensity color dots to synthesize this effect. However, by the inspection of the above example, the choice of color of each dot is the key concern. The color dot set, for the above example green, red, and light blue for shadow area of grass, represents the composition of light condition on object surface. Thus, random choice is not suitable.

We provide two pointillist rendering functions. The first approach is similar to the traditional algorithm. It uses the ground color to roughly painting the primitive first. Then color dots are placed on canvas. The colors of dots are the quantization of primitive color in their respective locations. The second approach also uses the ground color to roughly painting the primitive. Then, a color set is defined by the user. Color dots are placed on canvas. The colors of dots are chosen from the color set and will be blended with the color of primitive in their respective locations. Currently, user can specify at most three colors in the color set.

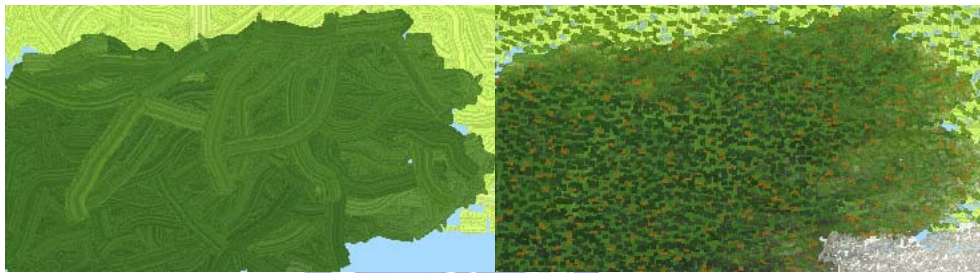




(a)



(b)



(c)

Figure 12 Pointillist rendering function. (a) Original image. (b) Synthesized result of first pointillist rendering function. (c) Synthesize result of second pointillist rendering function.

Chapter 6

Mark System

The list of stroke definitions generated by rendering system is fed to mark system which draws these strokes on canvas. The physical implementation of stroke definition heavily depends on the target painting media. For example, it is obvious that the water color mark system should differ from oil painting mark system a lot. Besides, the convenient usage of mark system is also important. In our framework the stroke definitions are automatically implemented by the mark system. Thus, physically-based mark system or interactive mark system in which too many possibilities exist is not suitable for our framework.

Our mark system basically contains three components: brush model including *Pigment*, *Canvas*, *Stroke Path*, and *Stroke Cross Section: Round CS, Flat CS*; bristle canvas interaction *BCInteraction* and random system *Randomable*. Bristle location is determined by stroke path and stroke cross section. Besides, each bristle contains pigment, so does canvas. The interaction between bristle and canvas occurs on every contact along the stroke path. To synthesize oil painting effects, some properties are made random through random system. The following figure shows the architecture of our proposed mark system.

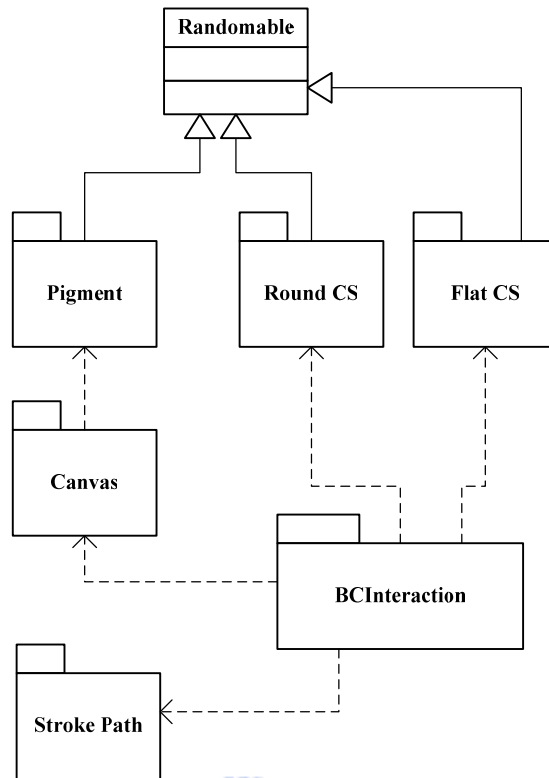
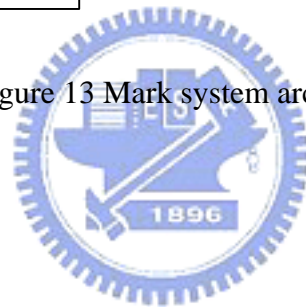


Figure 13 Mark system architecture.



6.1 Brush Model

6.1.1 Stroke Cross Section

It is common for Impressionist to use both round and float-section brushes. Thus, we create two stroke cross section: round stroke cross section and flat stroke cross section which both provide the following information: bristle location based on current brush location, orientation and bristle index; the blended stroke cross section with other cross sections which will be discussed in section 6.2; random interval level which will be discussed in section 6.3.

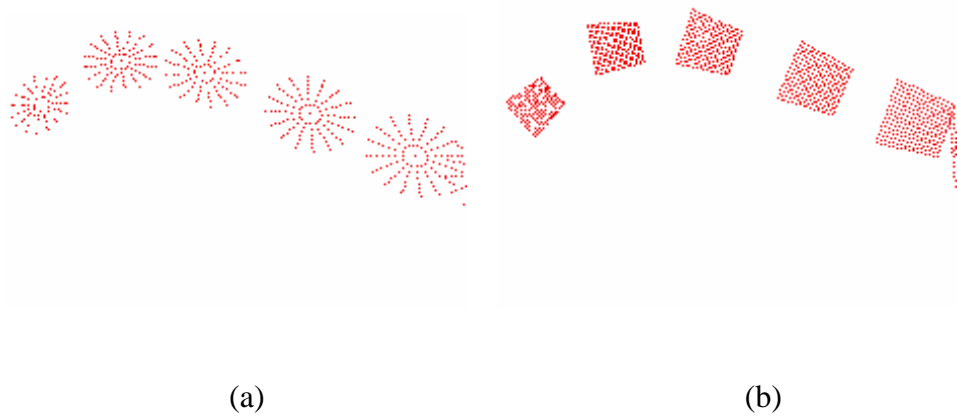


Figure 14 (a) Round and (b) flat stroke cross sections consist of low sampling points (Round: 16 x 8; Flat: 16 x 16).

As shown in Figure ??, the bristle distribution in round stroke cross section is based on the intersection between straight lines radiate from the ellipse center O and concentric ellipses bounded in the outmost ellipse. For example, P is such an intersection.

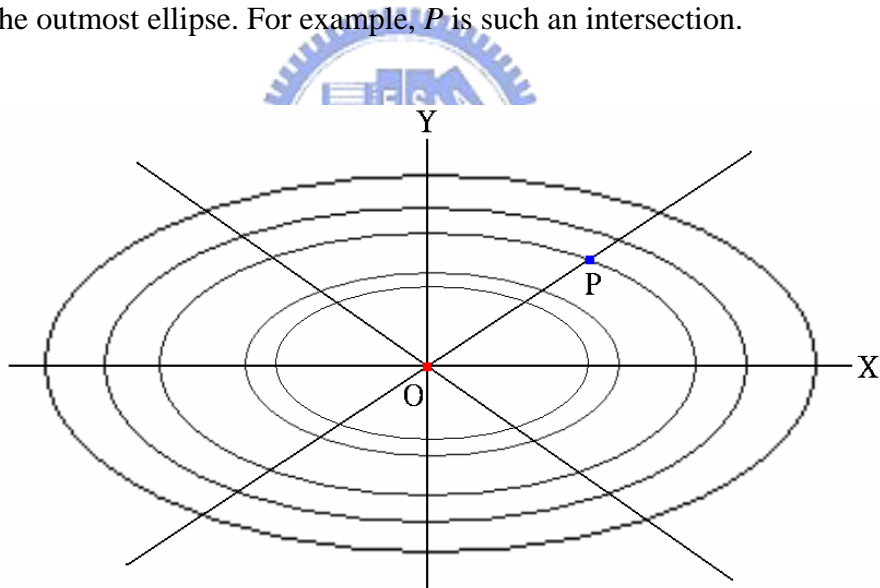


Figure 15 Bristle distribution in round stroke cross section.

The bristle location is calculated by the following formula:

$$\begin{aligned}
 x' &= A_i * \cos(\theta_i) \\
 y' &= B_i * \sin(\theta_i) \\
 x &= \cos(\gamma) * x' - \sin(\gamma) * y' + t_x \\
 y &= \sin(\gamma) * x' + \cos(\gamma) * y' + t_y
 \end{aligned}
 \quad \dots (1)$$

In the above formula, (x', y') is the coordinate of bristle in ellipse coordinate system, i.e. point O is treated as $(0, 0)$. When the brush is placed at (t_x, t_y) in world coordinate system with orientation angle γ , the third and fourth formula will transfer (x', y') into world coordinate system by a rotation and a translation. The first and second formula deal with the bristle location in ellipse coordinate system by given bristle index. The bristle index starts from zero and increases counterclockwise from the inside to outside of an ellipse.

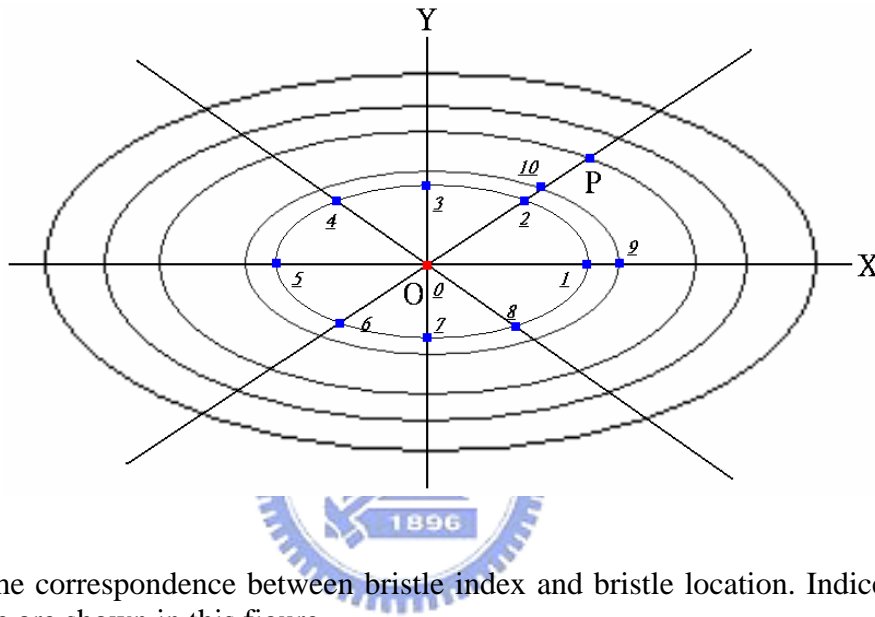


Figure 16 The correspondence between bristle index and bristle location. Indices of the first eleven bristle are shown in this figure.

Thus, take point P as an example, by the given bristle index, we can calculate the angle between line OP and x-axis and the length of line segment OP by the following formulas:

$$radius_index = \text{ceil}\left(\frac{bristle_index}{SAMPLE_ANGLE_SIZE}\right)$$

$$angle_index = bristle_index - radius_index * SAMPLE_ANGLE_SIZE - 1$$

$$radius_ratio = \frac{radius_index}{SAMPLE_RADIUS_SIZE}$$

$$angle_ratio = \frac{angle_index}{SAMPLE_ANGLE_SIZE}$$

$$A_i = radius_ratio * MAJOR_RADIUS$$

$$B_i = radius_ratio * MINOR_RADIUS$$

$$theta_i = angle_ratio * 2 * \pi$$

..... (2)

As shown in the above formula, there are four parameters controlling the distribution of bristles. *MAJOR_RADIUS* and *MINOR_RADIUS* are the major radius and minor radius of outmost ellipse. *SAMPLE_ANGLE_SIZE* and *SAMPLE_RADIUS_SIZE* control the count of sampled point in the contact region of brush.

Due to the nature of ellipse, uniform sampling of *radius_ratio* in Formula set (2) may sample too many points in the inner of ellipse. As a result, the bristle is not uniformly distributed in the contact region. To solve this problem, we define a mapping function from the original *radius_ratio* to the transformed *radius_ratio*. It is a power function and both its domain and co-domain are [0, 1].

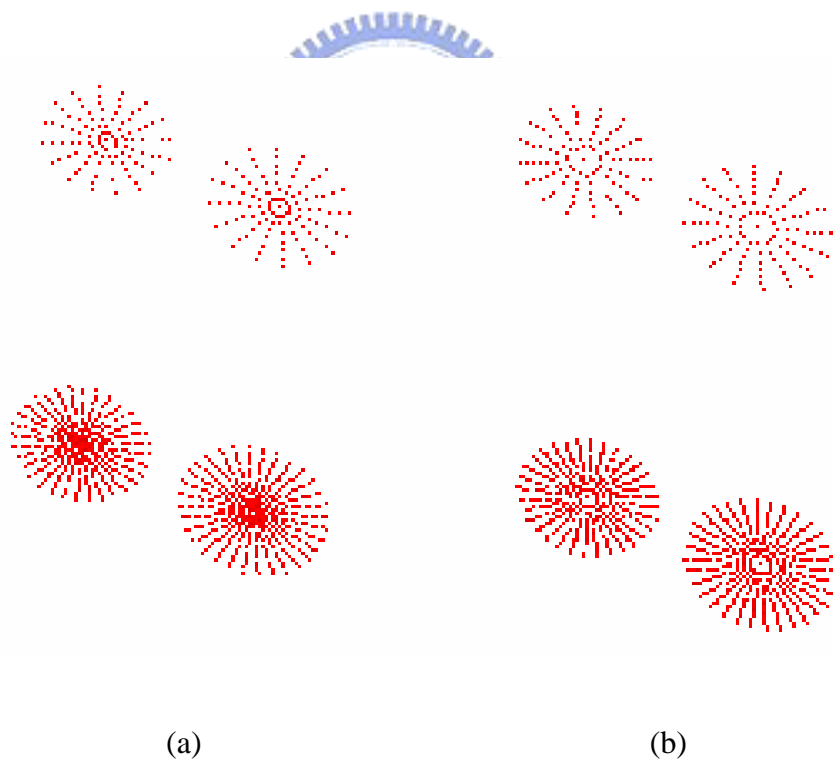


Figure 17 Two set of sampling results, each consists of two sampling (16x8, 32x16). (a) Uniform sampling of *radius_ratio* results un-uniform distribution of bristles; (b) Transfer *radius_ratio* by power function $y = x^{0.7}$ results uniform distribution of bristles.

The flat stroke cross section operates similar to the round stroke cross section. As shown in Figure ??, the bristles distribute in a trapezoid area centered at O with $BOTTOM_LENGTH$ $|AO|$, $UPPER_LENGTH_1$ $|CD|$, $UPPER_LENGTH_2$ $|EF|$ and $HEIGHT$ $|BD|$.

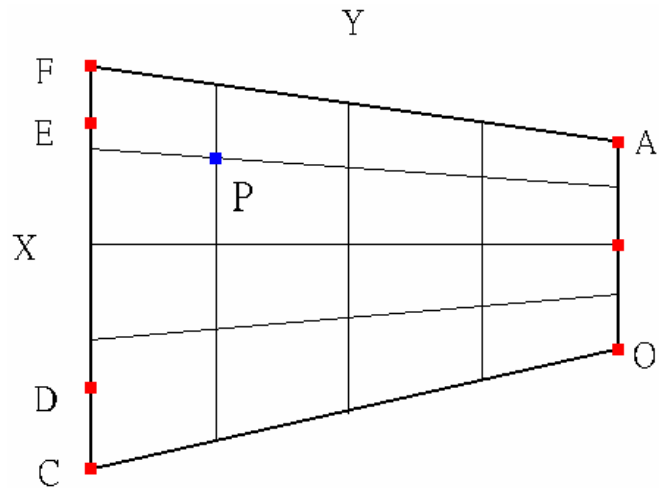


Figure 18 Bristle distributions in flat stroke cross section. $|AB| = |ED|$.

The bristle location is calculated by the following formulas:

$$\begin{aligned}
 x' &= 0 - column_ratio * HEIGHT \\
 y' &= (row_ratio * BOTTOM_LENGTH) * (1 - column_ratio) + \\
 & \left[0 - \frac{BOTTOM_LENGTH}{2} - UPPER_LENGTH_1 + row_ratio \right. \\
 & \left. * (UPPER_LENGTH_2 + BOTTOM_LENGTH + UPPER_LENGTH_1) \right] \\
 & * column_ratio \\
 x &= \cos(\gamma) * x' - \sin(\gamma) * y' + t_x \\
 y &= \sin(\gamma) * x' + \cos(\gamma) * y' + t_y \\
 & \dots (3)
 \end{aligned}$$

In the above formula, (x', y') is the coordinate of bristle in trapezoid coordinate system, i.e. point O is treated as $(0, 0)$. The third and fourth formula is similar to those in Formula (1). The first and second formula deal with bristle location in trapezoid coordinate system by

given bristle index. The bristle index starts from zero and increases from bottom to top and right to left.

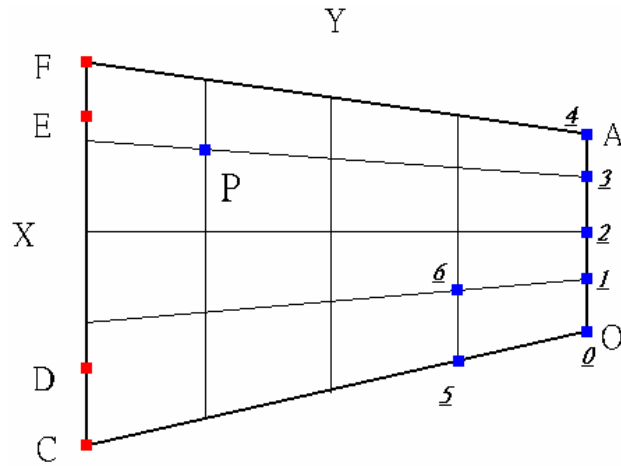


Figure 19 The correspondence between bristle index and bristle location. Indices of the first seven bristle are shown.

Thus, take point *P* as example, by the given bristle index we can calculate *column_ratio* and *row_ratio* in formula set 3 from the following formulas:

$$row_index = bristle_index \bmod SAMPLE_V_SIZE$$

$$column_index = \frac{bristle_index}{SAMPLE_V_SIZE}$$

$$row_ratio = \frac{row_index}{SAMPLE_V_SIZE - 1}$$

Formula set 4.

$$column_ratio = \frac{column_index}{SAMPLE_H_SIZE - 1}$$

SAMPLE_V_SIZE and *SAMPLE_H_SIZE* controls count of sampled point vertically and horizontally in the contact region of brush.

6.1.2 Stroke Path

The stroke path determines the location and orientation of a brush by interpolating a set of control points. Besides, stroke path also provides pressure and active bristle percentage

which will be discussed later. The location information is constructed by Cardinal spline since it will interpolate all control points. In fact, the Cardinal spline is simply constructed by Bezier curve. When the control points are given, additional control points will automatically be inserted into original adjacent control points. Then the original control points and the inserted control points are used together to construct the Bezier curve. Cardinal spline provides one parameter to control how new control points will be inserted. Smaller parameter will result smooth curve.

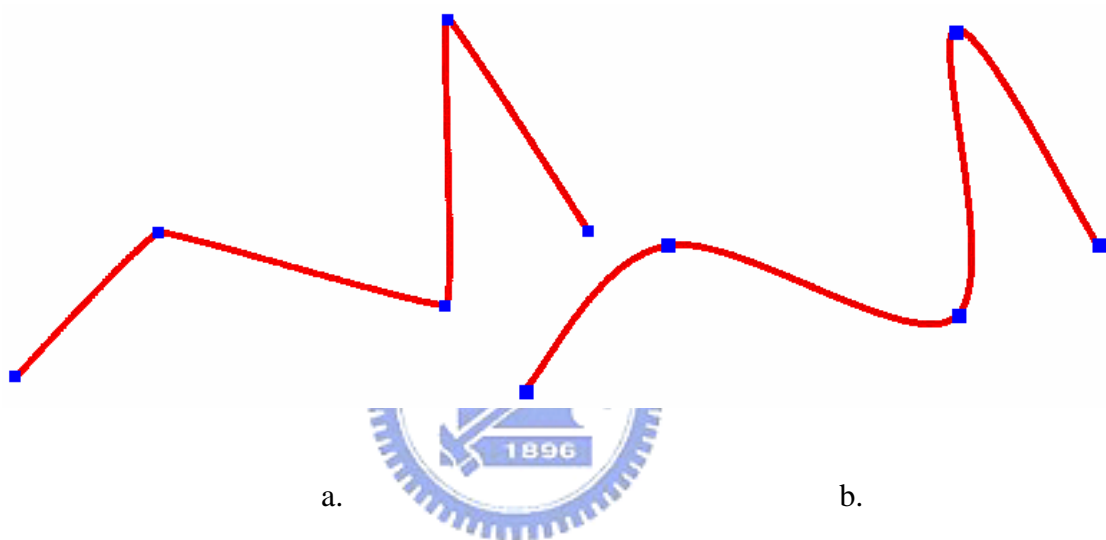


Figure 20 Cardinal spline constructed by five control points with different smooth parameters (a) 10 (b) 2 respectively.

By given control points and the stroke cross sections per control point, the tessellation mechanism will generate the bristle information in any location along the stroke path. Bristle information such as location, orientation, and pressure are linearly interpolated between control points. All of these are done by blending ability of stroke cross section. By given two stroke cross sections and the location orientation information (obtained from the stroke path), the blended stroke cross section will be generated. The tessellation scale can be adjusted to improve the efficiency.

Although the bristle amount of a brush is fixed, the amount of bristles which actually contact the canvas depends on the area of contact region. The larger area is, the more bristles contact the canvas. Hence we add supplementary information per control points: the active bristle percentage (abp). The range of ABP is between zero and one. It depends on the area of contact area. The larger area is, the more percentage is. ABP is calculated once randomly before the tessellation begins.

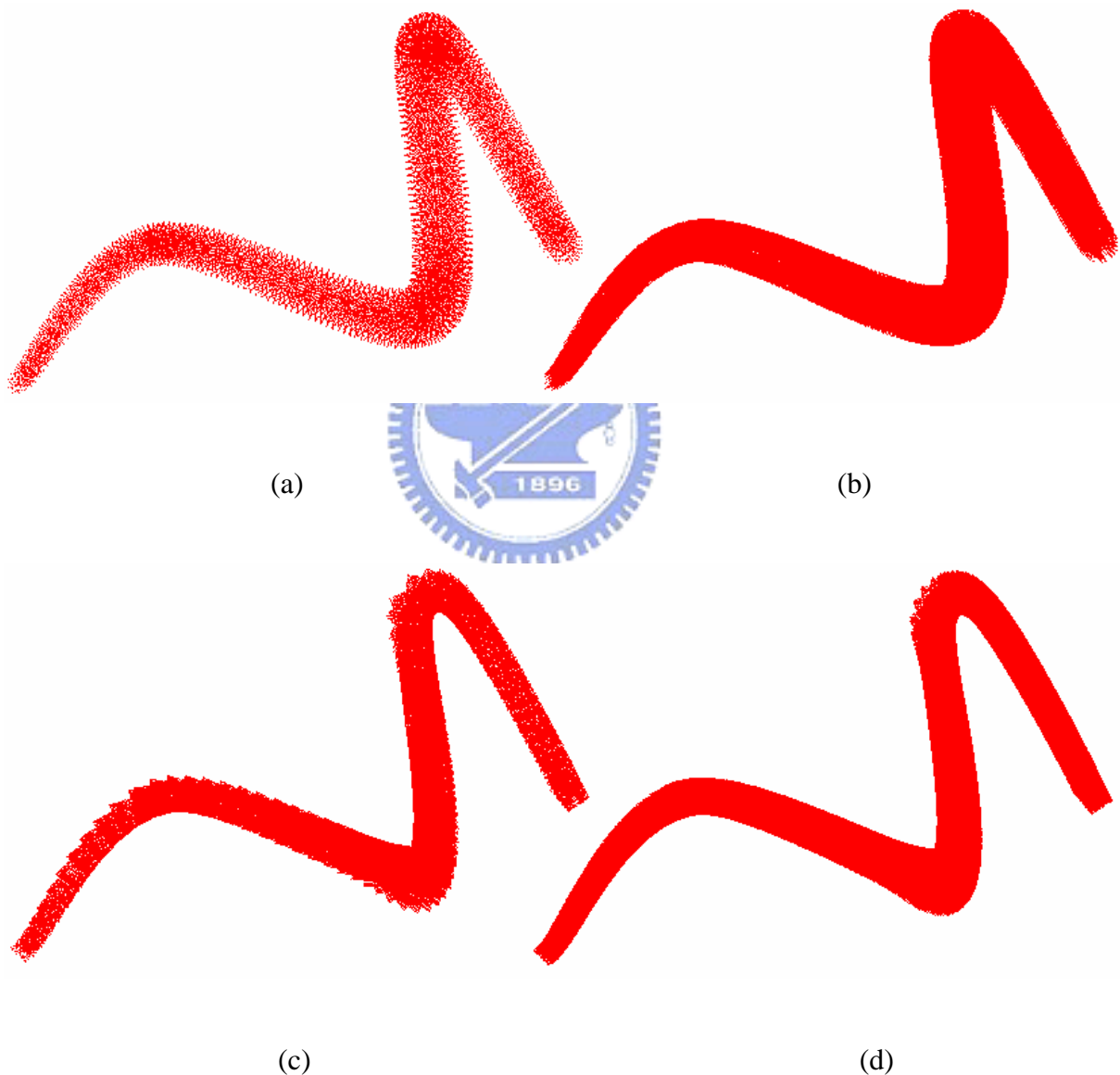
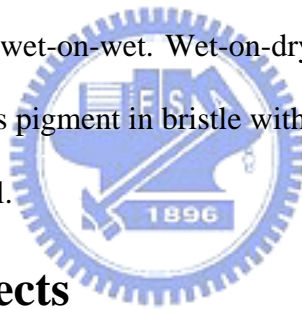


Figure 21 Tessellated strokes with different tessellation scale. (a) Round cross section with tessellation scale 8. (b) Round cross section with tessellation scale 1; (c) Flat cross section with tessellation scale 8; (d) Flat cross section with tessellation scale 1.

6.2 Bristle and Canvas Interaction

When the stroke path, stroke cross section per control point, initial bristle properties are done, they are tessellated by a list of bristle and canvas interactions. The interaction occurs on every touching operation of stroke cross section on canvas along stroke path. Bristle and canvas interaction determines how a bristle interacts with canvas, i.e. the change of properties of bristle and canvas. Canvas and bristle both contains one kind of property: pigment. It has color, quantity, drop scale, and height scale. The quantity, drop scale and the pressure controls the pigment drop amount. The height scale is used for impasto effect of oil painting in which pigment on canvas is three-dimensional, i.e. it has height relate to the canvas plane. The height of a pigment is the multiplication of its quantity and its height scale. There two kinds of interaction: wet-on-dry and wet-on-wet. Wet-on-dry simply drops pigment in bristle onto canvas while wet-on-wet blends pigment in bristle with pigment on canvas. The blending uses subtractive color mixture model.



6.3 Oil Painting Effects

To create real oil painting stroke effects. Two random systems are used: pure color intensity variant random and height gap effect random. Pure color intensity variant concerns the effect that when apply one pure pigment color on canvas, the actual color is not uniform. In fact the intensity of the pigment color varies. Height gap effect concerns the effect that pigment height is not uniform in a stroke. Usually, height of pigment could form several spans. Each span has roughly uniform height while different spans have different heights.



(a)



(b)

Figure 22 Real oil painting strokes. (a) Pure color intensity variant random. (b) height gap effect random.

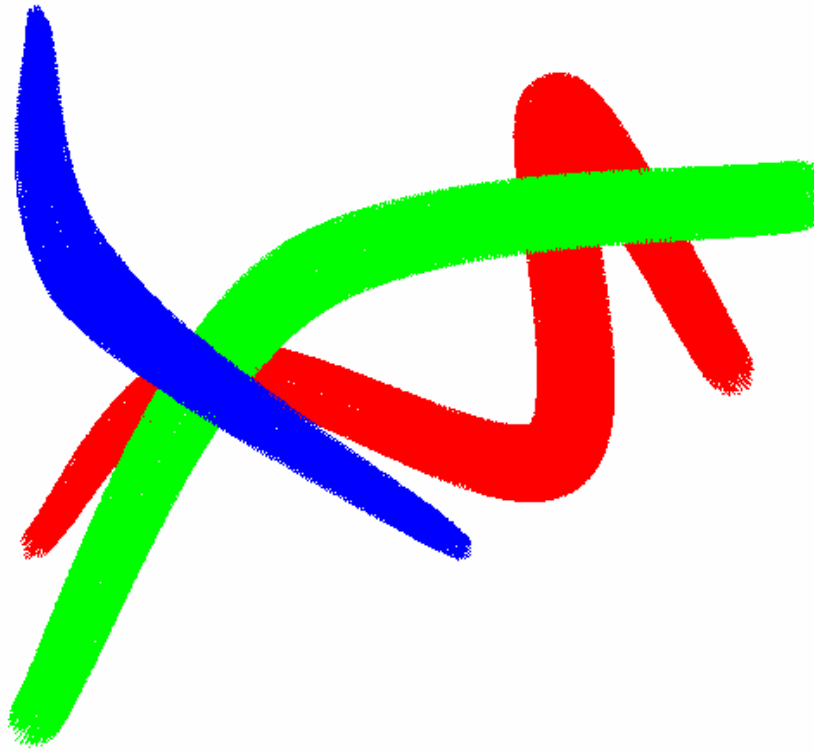
To synthesize pure color intensity variant effect, we randomly change the pigment intensity of initial bristle properties. In RGB color model, changing intensity is done by scaling RGB values identically.

To synthesize height gap effect, we first randomly generate a series span values and then use these span values to change the drop scale of pigment. The drop scale of pigment affects the drop quantity of pigment and thus affects the height of pigment. The generation of span values can be controlled by four parameters: *base_gap*, *min_gap*, *random_gap* and *random_level*. Bristles are collected into the same span according to their y-axis coordinate in local coordinate system. The span size and value are generated from the following formula:

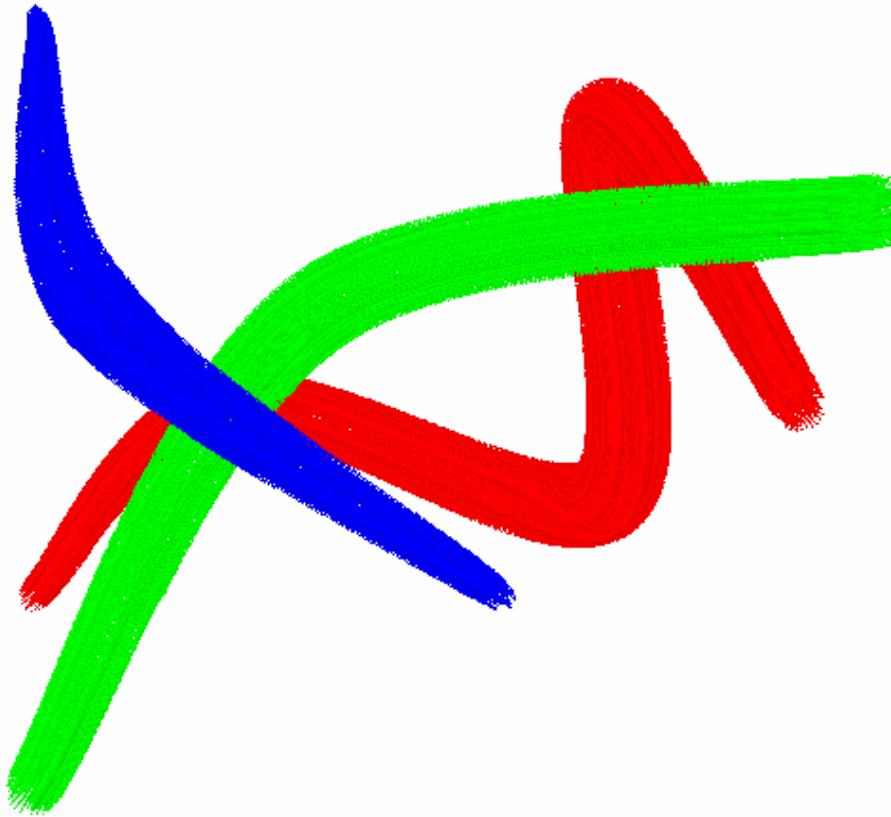
$$\begin{aligned} span_size &= \min(\min_gap, base_gap + Rand(0 - random_gap, random_gap)) \\ span_value &= Rand(0, random_level) \end{aligned}$$

..... (5)

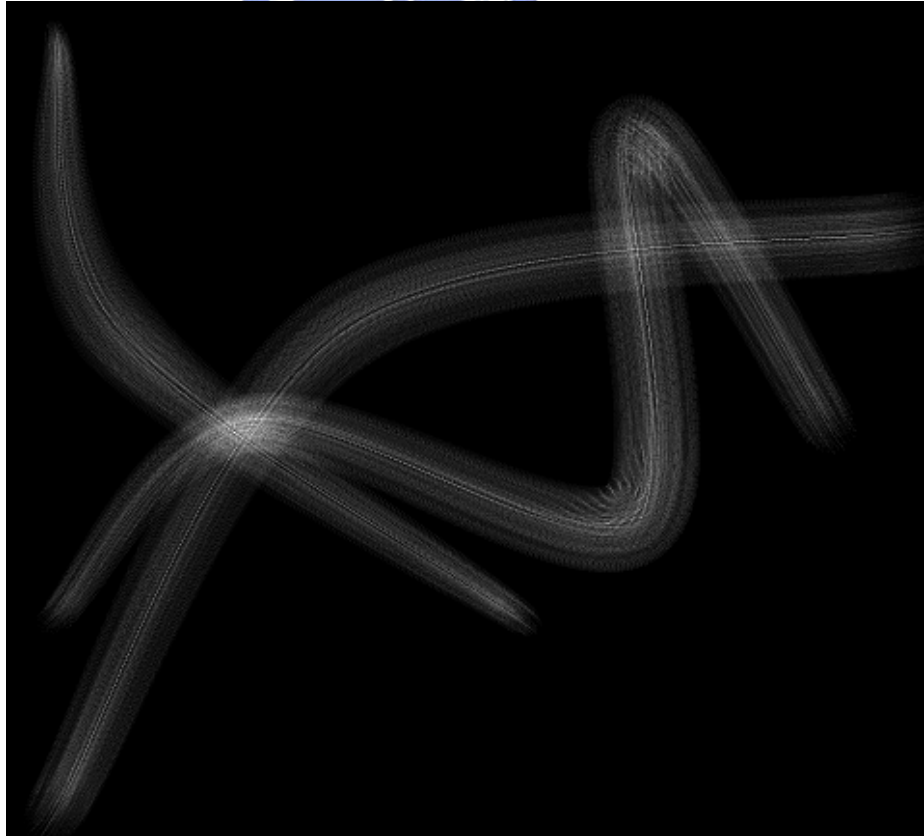
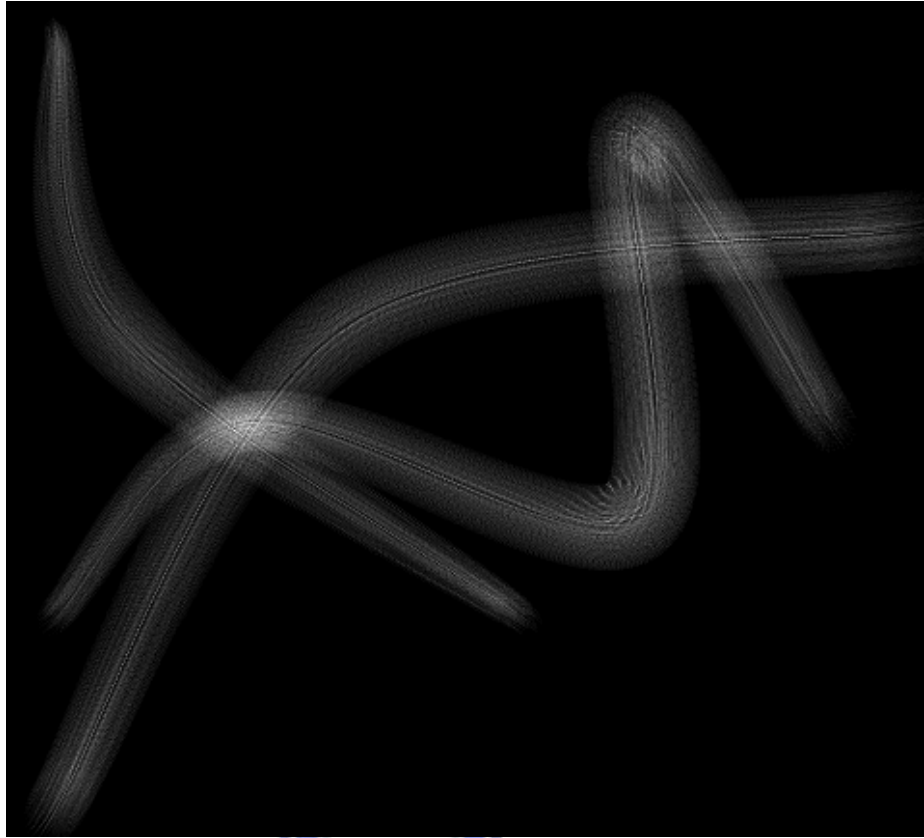
The $Rand(a, b)$ function in formula 5 will generate a random number between a and b .



(a)



(b)



(d)

Figure 23 Two random systems: (a) Plain stroke. (b) Pure color intensity variant effect. (c) Plain height. (d) Height gap effect.



Chapter 7

Results

In this chapter, five result sets are presented. Result set 1 shows an ordinary oil painting effect and advance layering effect. Result set 2 shows the quick drawing effect of Impressionism. Result set 3 shows Pointillist's style which belongs to Neo-Impressionism. The framework is implemented in C++ language on NB with a Pentium 1.5G CPU and 512 MB RAM.

7.1 Ordinary and Advance Layering



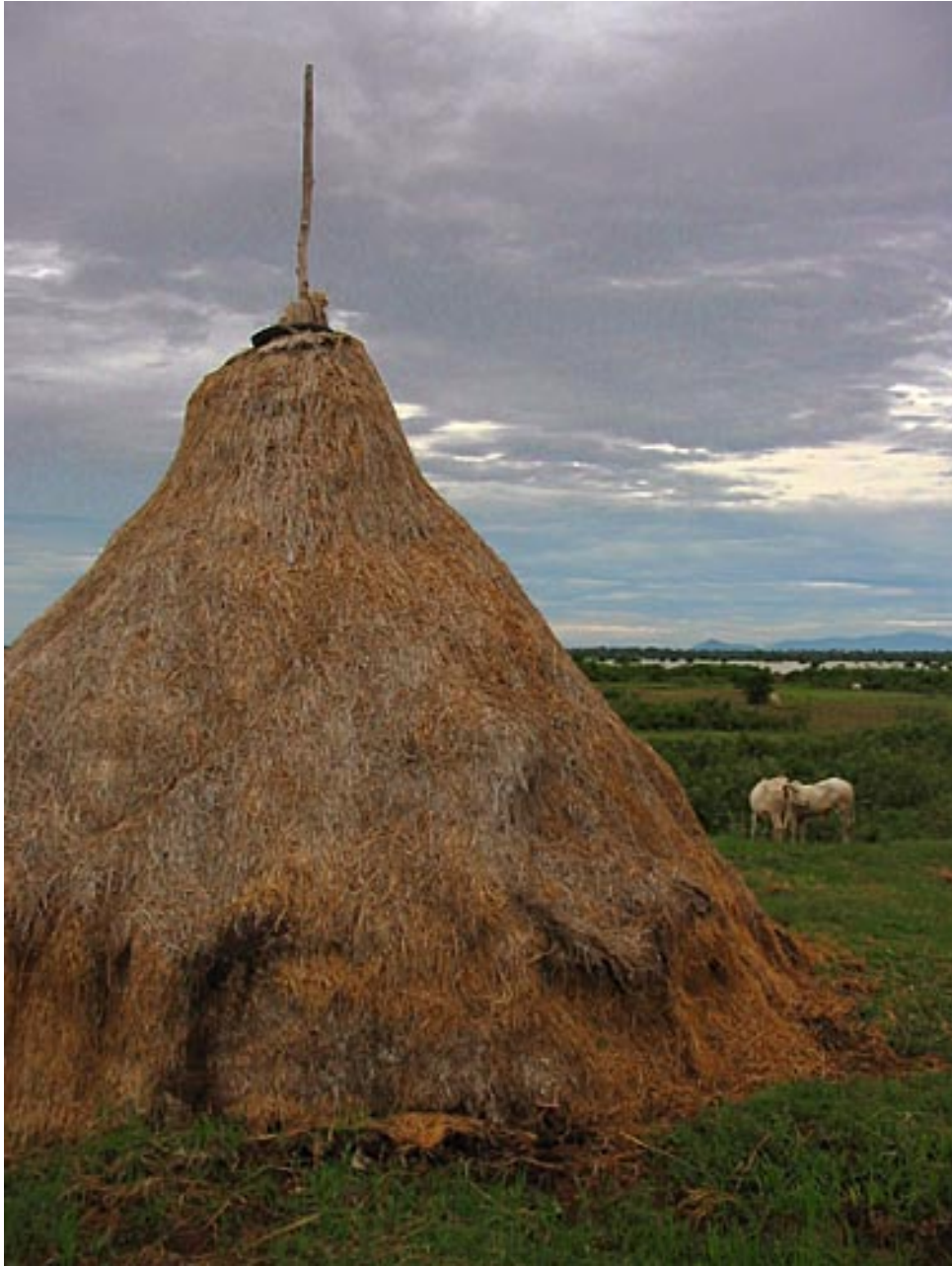


Figure 24 Original input image.

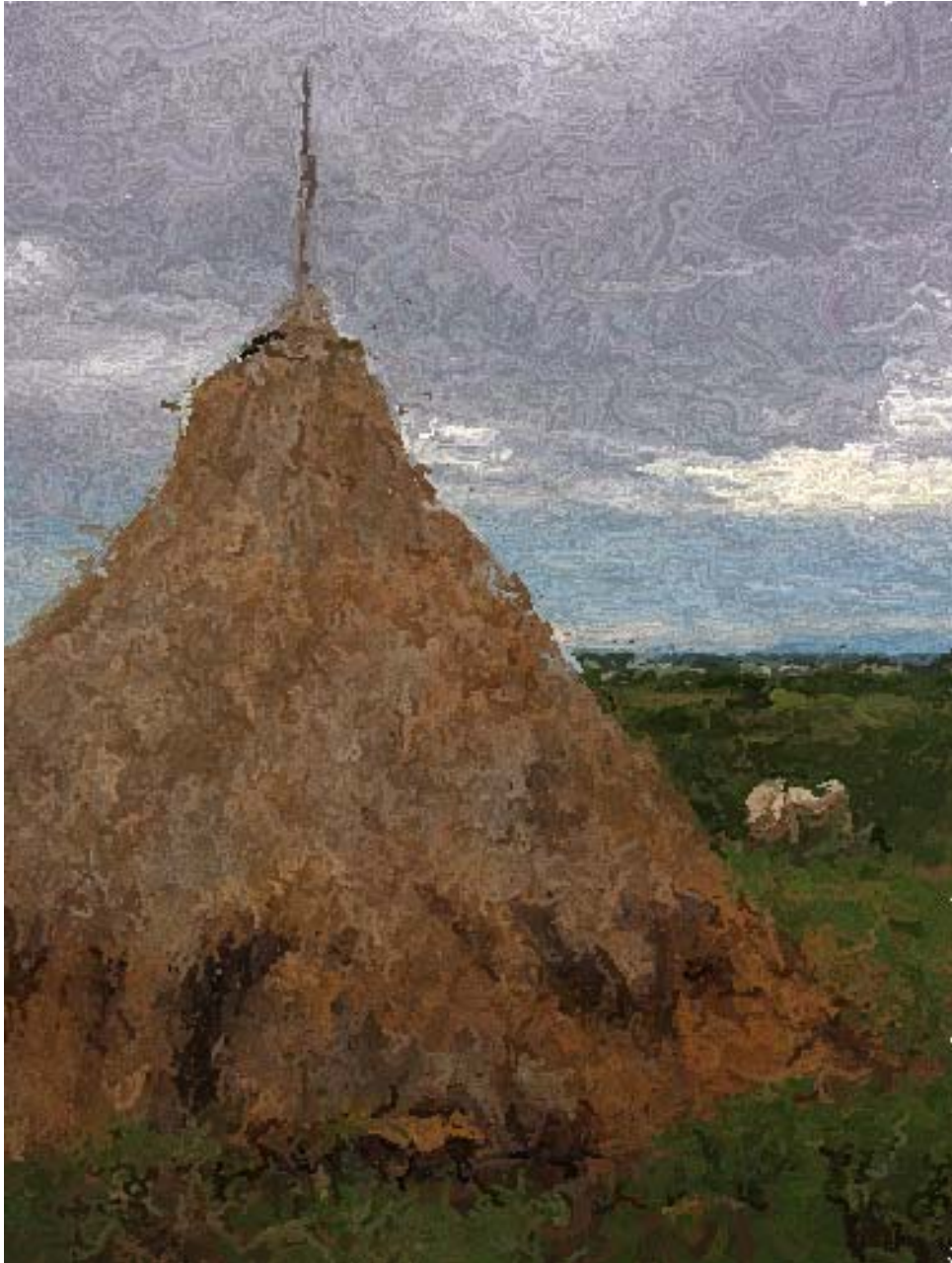


Figure 25 Final image composed of five layers. The whole image is treated as one primitive.



First Layer



Second Layer



Third Layer



Fourth Layer

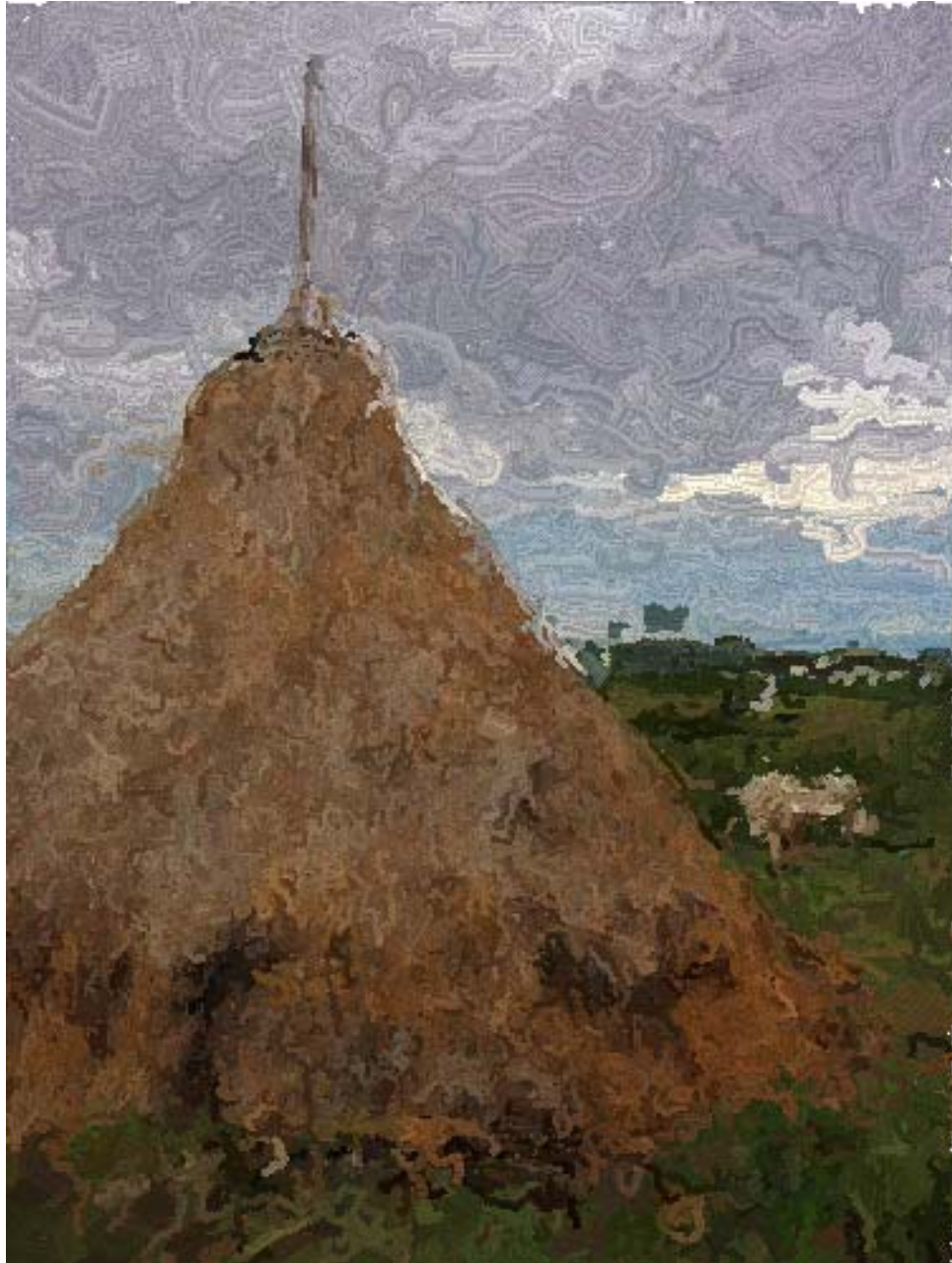


Figure 26 Final image constructed by five stages. Five primitives: entire image, sky, grass, haystack and sheep.



First stage (two layers): the whole image is treated as a primitive.



Second stage (three layers): sky.



Third stage (two layers): grass.



Fourth stage (two layers): sheep.



Fifth stage (three layers):: haystack.

7.2 Quick Drawing



Figure 27 Original input image.



Figure 28 Final image composed of two stages. Fourteen primitives: entire image and thirteen geese.



Final image of first stage (two layers): whole image.



Final image of second stage (two to three layers): thirteen geese.



One goose: second stage (three layers).

7.3 Pointillist





Figure 29 Original input image.



Figure 30 Final image composed of two stages. Six primitives: sky, tree, haystack, grass, grass shadow, and road.



Final image of first stage.



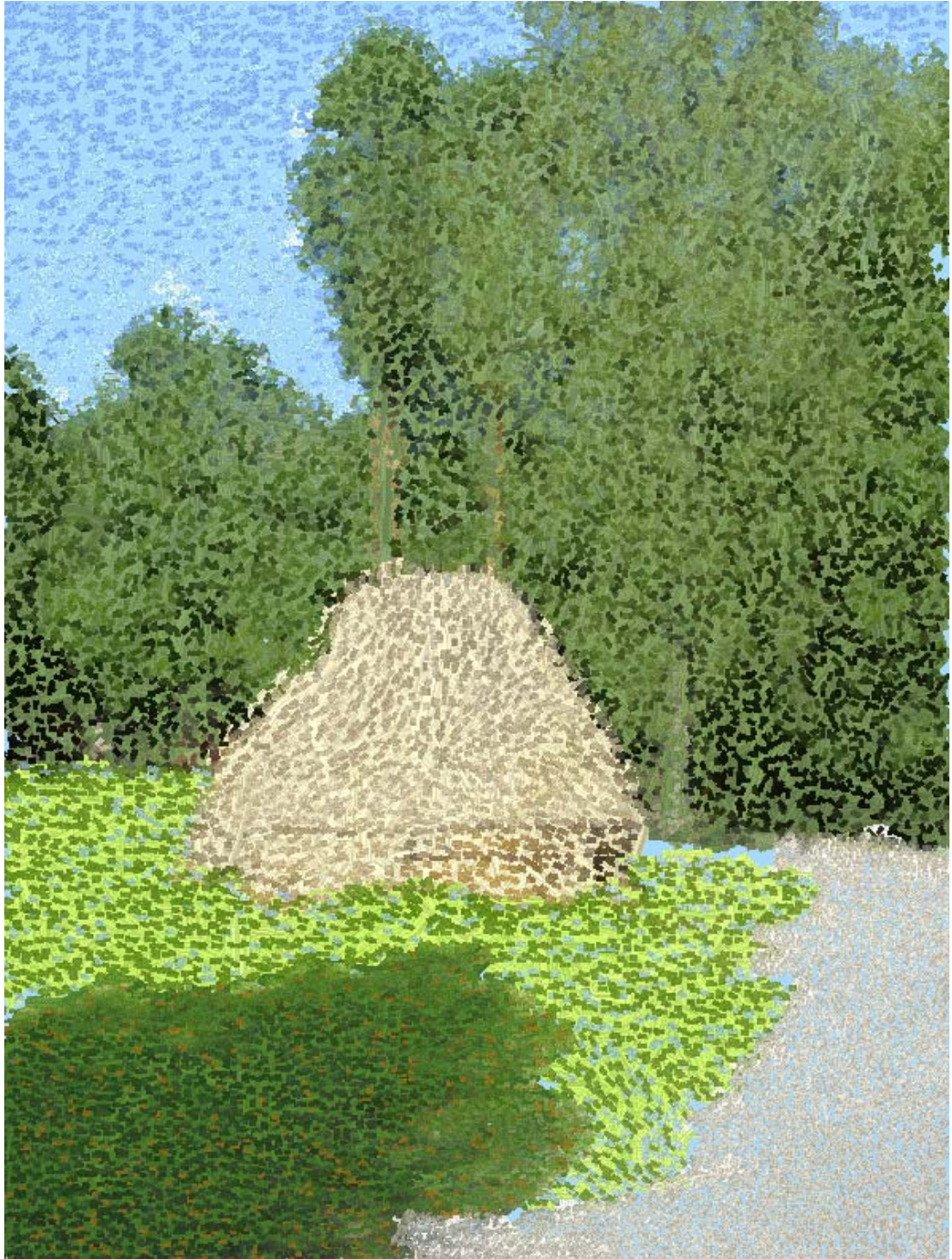


Figure 31 Final image composed of two stages with another Pointillist style. Six primitives: sky, tree, haystack, grass, grass shadow, and road.

	Image Size	Primitive	Layers	Time (seconds)
Basic I	375 x 500	1	4	191
Basic II	375 x 500	5	2, 3, 2, 1	241
Quick Draw	800 x 600	15	2, {2~3,...} ¹	210
Pointillist I	594 x 792	6	{1,...}, {1,...} ²	145
Pointillist II	594 x 792	6	{1,...}, {1,...} ²	144

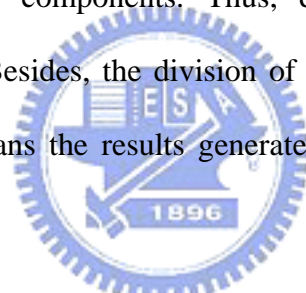
Table 1 Result statistics.



Chapter 8

Conclusions and Future Works

A painterly rendering framework has been presented in this thesis. The framework consists of three sub-systems: primitive mapping system, rendering system and mark system. The framework is written in C++. It is complete and flexible. Any developers can extend this framework by costuming its components. Thus, different painterly algorithm can be implemented in this system. Besides, the division of this framework is based on the actual process of painters which means the results generated can be more evaluative in esthetics aspect.



Impressionist paintings are synthesized by this framework to show its effectiveness in this thesis. We've shown three different styles of Impressionist paintings: original, quick drawing and Pointillism. Several results are presented in each style. Achieving each style needs extending or modifying each components of this framework. The extending or modifying work is clear and intuitive.

In the future, we'll try to enhance our framework in the following aspects:

Add the spatial system to this framework. We omit this part in our thesis. However, it is important especially in oriental paints. I'll try to figure out how the complex projection system is establishing in painting process. Another future work is to improve the mark system. So far, we just synthesized two effects of oil painting. We'll do more in the future. Having a GUI interface such like Render Monkey is also a future work.

References

- [1] BOMFORD D., KIRBY J., LEIGHTON J. and ROY A., “Art in the Making Impressionism”, *The National gallery, London in association with Yale University Press*, 1990.
- [2] CURTIS C. J., ANDERSON S. E., SEIMS J. E., FLEISCHER K. W., SALESIN D. H., “Computer-Generated Watercolor”, *Proceedings of ACM SIGGRAPH 97*, 1997.
- [3] DURAND F., “An Invitation to Discuss Computer Depiction”, *Proc. 2nd Intern. Symposium on Non-Photorealistic Animation and Rendering*, 2002.
- [4] GRABLI S., TURQUIN E., DURAND F., X F., “Programmable Style for NPR Line Drawing”, *Rendering Techniques 2004 (Eurographics Symposium on Rendering)*, 2004.
- [5] HAASE C. S. and MEYER G. W., “Modeling Pigmented Materials for Realistic Image Synthesis”, *Master's Thesis, University of Oregon, August 1991*, 1991.
- [6] HAEBERLI P., “Paint by numbers: abstract image representations”, *Proceedings of ACM SIGGRAPH 86*, 1986.
- [7] HAYS J., ESSA I., “Image and Video Based Painterly Animation”, *The 3rd International Symposium on Non-Photorealistic Animation and Rendering*, 2004.
- [8] HERTZMANN A., “Painterly Rendering with Curved Brush Strokes of Multiple Sizes”, *Proceedings of ACM SIGGRAPH 98*, 1998.
- [9] HERTZMANN A., “A Survey of Stroke-Based Rendering”, *IEEE Computer Graphics & Applications, Special Issue on Non-Photorealistic Rendering. July/August*, 2003.
- [10] LITWINOWICZ P., “Processing Images and Video for An Impressionist Effect”, *Proceedings of ACM SIGGRAPH 97*, 1997.
- [11] LEE M., KOWALSKI M. A., TRYCHIN S. J., BOURDEV L. D., GOLDSTEIN D., and HUGHES J. F., "Real-Time Nonphotorealistic Rendering", *Computer Graphics: Proceedings SIGGRAPH, Aug 1997. Annual Conference Series 1997. ACM Siggraph*, 1997.

- [12] MEIER B. J., "Painterly Rendering for Animation", *Proceedings of ACM SIGGRAPH 96*, 1996.
- [13] SALISBURY M., ANDERSON C., LISCHINSKI D., and SALESIN D. H., "A Resolution-Independent Representation for Pen-and-Ink Illustrations", *Dept. of Computer Science and Eng. U. of Washington. Technical Report UW-CSE-96-01-02*. Jan 1996.
- [14] SALISBURY M. P., WONG M. T., HUGHES J. F., and H D., "Orientable Textures for Image-Based Pen-and-Ink Illustration", *Proceedings of SIGGRAPH 97, in Computer Graphics Proceedings, Annual Conference Series, 401-406*, August 1997.
- [15] STRASSMANN S., "Hairy Brushes", *Proceedings of ACM SIGGRAPH 86*, 1986.
- [16] STEVE U., "The Renderman Companion", *Addison Wesley, Reading, MA*, 1989
- [17] WANG B., WANG W., YANG H., SUN J., "Efficient Example-Based Painting and Synthesis of 2D Directional Texture", *IEEE Transactions on Visualization and Computer Graphics Volume 10, Number 3*, 2004
- [18] WEN S. Z., SHIH Z. C., CHIU H. Y., "The Synthesis of Chinese Ink Painting", *National Computing Symposium'99*, page 461-468, 1999.
- [19] WILLATS J., "Art and Representation", *Princeton U. Press*, 1997.
- [20] WINKENBACH G., SALESIN D. H., "Computer-Generated Pen-and-Ink Illustration", *Proceedings of ACM SIGGRAPH 94*, 1994.