國立台灣大學資訊工程研究所碩士論文

指導教授：歐陽明博士

以國畫手法自動描繪三維動物模型

Automatic Rendering of 3D Animal Models in Chinese Painting Style

研究生：葉俊緯 撰

學號：R89922052

中華民國九十一年六月

Abstract

A set of algorithms is proposed in this paper to automatically transform 3D animal models to Chinese painting style. Inspired by the real painting process in Chinese painting of animals, we divide the whole rendering process into two parts: borderline stroke making and interior shading. In borderline stroke making process we first find 3D model silhouettes in real-time depending on the viewing direction of a user. After retrieving silhouette information from all model edges, a stroke linking mechanism is applied to link these independent edges into a long stroke. Finally we grow a plain thin silhouette line to a stylus stroke with various widths at each control point and a 2D brush model is combined with it to simulate a Chinese painting stroke.

In the interior shading pipeline, three stages are used to convert a Gouraud-shaded image to a Chinese painting style image: color quantization, ink diffusion and box filtering. The color quantization stage quantizes all pixels in an image into four color levels and each level represents a color layer in a Chinese painting. The ink diffusion stage is used to transfer ink and water among different levels and to grow areas in an irregular way. The box filtering stage blurs sharp borders among different levels to embellish the appearance of final interior shading image. In addition to automatic rendering, an interactive Chinese painting system which is equipped with friendly input devices can be also combined to generate more artistic Chinese painting images manually.

中文摘要

在這篇論文中我們提出了一套將三維動物模型自動轉換成國畫手法的演算法。受到國畫動物畫法過程的啟發，我們將整個繪製的程序分成兩大部分：邊線筆觸(borderline stroke)的產生以及內部著色(interior shading)。在製作筆觸的過程中我們先依據使用者的視角即時地找出三維動物模型的輪廓，然後使用一個連接輪廓線的機制將之連成一整條的筆觸，最後我們藉由在一條筆觸中的控制點上擴張出不同的寬度使得一個細而平凡的筆觸可以具有手繪風格，並且結合一個二維的筆刷模型以模擬出國畫的筆觸。

而在內部著色的部分，我們共使用三個步驟來轉換一個葛氏著色法(Gouraud- shading)的模型影像成為具有國畫風格的影像：顏色量化法、墨水擴散法、及影像過濾法。顏色量化法將影像中所有的圖素指定成四個色階，而每一個色階代表國畫中不同的色階。墨水擴散法是用來在不同的色階中傳遞墨水並且會使每一塊色階的區域不規則地向外擴張。影像過濾法將不同色階中尖銳的邊界模糊化而其目的是為了修飾最後內部著色繪製的結果。除了自動繪製的方法外，使用者也可結合一個具有著友善輸入介面的互動式國畫繪畫系統來產生更具藝術性的中國水墨動物畫。

TABLE OF CONTENTS

Page

ABSTRACT……2

LIST OF FIGURES……8

CHAPTER 1 INTRODUCTION……10

1.1 Introduction……10

1.1.1 Introduction to Non-photorealistic Rendering……12

1.1.2 Introduction to Chinese Painting……15

1.2 Contributions……18

1.3 Application Areas……18

1.4 Organization……19

CHAPTER 2 FINDING 3D MODEL SILHOUETTE……22

2.1 Introduction……22

2.2 Definition of Silhouette……23

2.3 Algorithms……24

2.4 Data Structures for Polygon Models……25

2.5 Silhouette Culling……26

2.6 Conclusion……29

CHAPTER 3 STROKE MECHANISM……30

3.1 Introduction……30

3.2 Chinese Painting Strokes……30

3.3 Stroke Linking Mechanism……32

3.4 Stroke Making Mechanism……34

3.5 Results……36

CHAPTER 4 BRUSH MODEL……38

4.1 Introduction……38

4.2 Chinese Pen Brush……39

4.3 Simulating a Brush Model……39

4.4 Brush Movement Control Mechanism……41

4.5 From a Brush Model to Strokes……43

4.6 Special Effects……44

4.6.1 Dry Brush Effect……44

4.6.2 Turning Effect……45

4.7 Results……..46
CHAPTER 5 INTERIOR SHADING……...48
5.1 Introduction……..48
5.2 Interior Shading Mechanism……49
5.3 Results……..54
CHAPTER 6 RESULTS……56
CHAPTER 7 CONCLUSION AND FUTURE WORKS……62
7.1 Conclusion……62
7.2 Future Works……..……63
BIBLIOGRAPHY……64

LIST OF FIGURES

Fig 1.1 An example of technical illustration, where the left of figure is the Phong-shaded model and at the right is one that applies the technical illustration lighting model in edge lines. Images are provided by A. Gooch, B. Gooch, P. Shirley and E. Cohen, "A Non- photorealistic Lighting Model for Automatic Technical Illustration", appeared in SIGGRAPH'98……..…….....11

Fig 1.2 Left: Convert a low resolution 2D image into watercolor style. "Computer- generated Watercolor" [4] by CURTIS, C. J. in SIGGRAPH 97. Right: The engraving style using a Venus photo. "Digital Facial Engraving" [15] by OSTROMOUKHOV, V. in SIGGRAPH 99……...13

Fig 1.3 *A* is the source image; *A'* is the filtered version of *A,* and user input a target image *B*; the analogies mechanism will apply the same filter on *B* to get *B'*. "Image Analogies" by HERTZMANN, A. in SIGGRAPH 01……...13

Fig 1.4 Left: A pen-and-ink illustration from a photo portrait. "Interactive pen-and-ink illustration." by SALISBURY, M. P. in SIGGRAPH 94. Right: Applying hatching style on 3D models. "Real-Time Hatching" by PRAUN, E. in SIGGRAPH 01…….14

Fig 1.5 Left: Human figure with expressive outline and shading strokes. "Real-Time Nonphotorealistic Rendering" by MARKOSIAN, L. in SIGGRAPH 97. Right: Ink-wash style applies on silhouettes. "Artistic Silhouettes: A Hybrid Approach" by NORTHRUP, J. in NPAR 2000……15

Fig 1.6 "Poet on a Mountain Top." album leaf mounted as a hand scroll. Shen Chou 1427-1509...16

Fig 1.7 Birds and leaves by Gerald Liu…….17

Fig 1.8 Horses by Liu Qiao Yang……..17

Fig 2.1 Definition of a silhouette: At a point on a surface $\sigma(u,v)$, and given $V(u,v)$ as the eye vector and $n(u,v)$, as the surface normal, a silhouette point is defined as the point on the surface where $V(u,v) \bullet n(u,v) = 0$ or the angle between $V(u,v)$ and $n(u,v)$ is 90 degrees……..23

Fig 2.2 For polygons, an edge between two polygons is a silhouette edge if the edge is shared by a front-facing and a back-facing polygon…….24

Fig 2.3 A code sample. Data structure of our system……...26

Fig 2.4 By the definition of silhouette, we can find two silhouette edges in the example but silhouette edge at the right side is invisible because it is occluded by faces at the left side……..27

Fig 2.5 Three conditions occur in the depth buffer test. (A) Both endpoints pass the depth buffer, (B) Both endpoints don't pass the depth buffer and (C) One endpoint passes the depth buffer filter but the other doesn't……..28

Fig 2.6 The "cutting" mechanism may cause problem because it will increase the distance between two close silhouettes……...28

Fig 3.1 Chinese painting of a cattle and a horse by Li Chi-Mao [20]……...31

Fig 3.2 The pseudo code of our stroke linking algorithm…….32
Fig 3.3 In order to keep the sequence of control points when inserting a new control point, we store data in two directions depending on these two cases. Point A and point B represent the original top and bottom point in the link list……...33
Fig 3.4 A stroke defined by four control points with intervening nodes generated by a cubic spline.…….34
Fig 3.5 A stroke with 9 control points. We set ***width_step*** to 2 and ***MAX_WIDTH*** sets to 7. The following table shows the current values of ***add[i]*** and ***width[i]***……36
Fig 3.6 By passing stroke linking stage and stroke making stage, we automatically transform 3D animal models into silhouette stroke image……...37
Fig 4.1 Brush Model.…….40
Fig 4.2 Brush model moves along the stroke from (x_i, y_i) to (x_{i+1}, y_{i+1}) and rotate by θ.....42
Fig 4.3 Fitting brush model into stroke. We locate the brush model at the control point of stroke, and then we interpolate position and size of the brush model between two control points……...43
Fig 4.4 Pseudo code of dry brush effect……45
Fig 4.5 Apply brush model with dry brush effect and turning effect to silhouette strokes……...47
Fig 5.1 Chinese painting of animals with interior shading. We found that artists always use only two or three levels of colors to paint the interior area……49
Fig 5.2 Interior shading pipeline……...50
Fig 5.3 Two Gouraud-shading images with different lighting parameters and the color distribution histogram related to the image. The right one uses large diffuse and specular lighting value so that the color distribution range is wider than the left one. ……...51
Fig 5.4 We define three thresholds in color distribution histogram: 0.4, 0.6, 0.9 and we assign each pixel color to a threshold value according to their location in color distribution histogram.....51
Fig 5.5 We quantize an original Gouraud-shading image (Left) to a four-color image (Right)…...52
Fig 5.6 Left: A color quantization image. Right: Image after passing ink diffusion stage; the area of each color level has grown in an irregular way. ……53
Fig 5.7 Left: An ink diffusion image, but it still has sharp borders between color levels. Right: We pass the left image to a 5x5 box filter to get a blurred one……54
Fig 5.8 Rendering results with brush silhouettes and interior shading.…….55
Fig 5.9 Final results after using Lien's interactive painting system to add interior strokes and the original 3D model is at the right bottom corner of image……..55
Fig 6.1 Rendering results generated from different 3D animal models by applying algorithms proposed in this thesis and the rendering process is fully automatic. ……...60
Fig 6.2 Images with color interior shading generated by using Lien's interactive painting system…….61

Chapter 1
Introduction

1.1 Introduction

In computer graphics, many researchers put efforts in developing new rendering techniques, because they all have a dream to make the virtual worlds in computer graphics become photo-realistic. Therefore, important algorithms are invented such as ray-tracing, radiosity, etc. and now these techniques are widely used in the movie industry post-production. The extreme case is to use 3D human models as main characters in the movie instead of real human. In 2001, the compute graphics movie "Final Fantasy – The Spirits Within" [1] made a good demonstration of this concept and gave the audience a big shock. No doubt they succeeded in making a breakthrough of computer graphics movie by using the most advanced techniques whether in the modeling stage or in the rendering stage. However, when watching this movie we can still feel something unnatural, for example the facial expression of characters and their facial skin textures. The human perceptions are very sensitive to human hair and skin, since we have much experience in observing them in detail

inside a mirror or through somebody else's face.

On the other hand, in some situations we do not need to render models photo-realistically because a human is easily distracted by other features which are not important. Technical illustrations extract the key points of an object and ignore unimportant parts of it, and yet can provide instant knowledge. If we want to sketch technical illustrations of complicated features in computer graphics such as the building architecture and bone structure, we do not need to apply Phong-shading to these 3D models and just need to sketch few feature lines which we want to emphasis in the model. Similarly, a different lighting model [2] will be applied to technical illustration rendering to give a clear shape of features. Figure 1.1 presents an example of computer-generated traditional technical illustration.

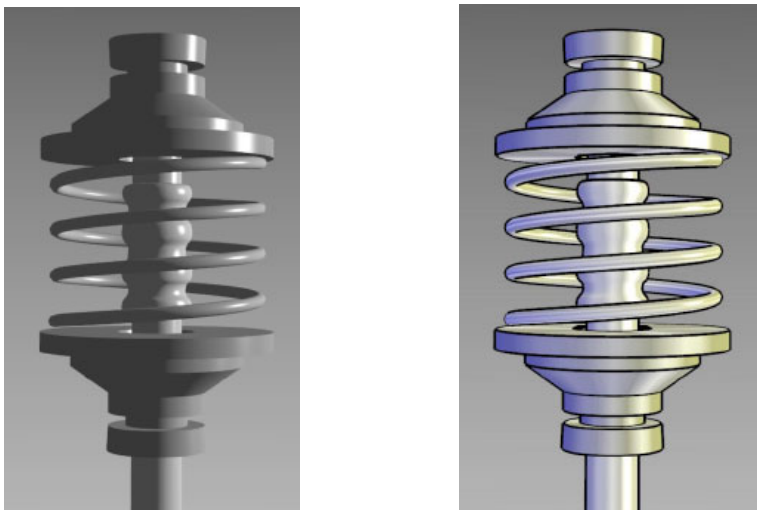


Fig 1.1 An example of technical illustration, where the left of figure is the Phong-shaded model and at the right is one that applies the technical illustration lighting model in edge lines. Images are provided by A. Gooch, B. Gooch, P. Shirley and E. Cohen, "A Non- photorealistic Lighting Model for Automatic Technical Illustration", appeared in SIGGRAPH'98.

In recent years, alternative needs from different cases come out in computer graphics. The goals of these techniques are not rendering objects realistically but more creatively such as in simulating various painting styles on 2D images or 3D models. This kind of techniques is called non-photorealistic rendering (NPR). Due to a different look in rendering results, it can inspire a human in reading a picture. In the next section we will further introduce about non-photorealistic rendering (NPR) techniques.

1.1.1 Introduction to Non-photorealistic Rendering

The main spirit of generating a NPR image is to be the communication tool between a user and the designer [3]. Designers should let a user know clearly what his work want to present. Refer to the book "Non-photorealistic Rendering" [3], researches in NPR can be roughly separated into three categories; artistic media simulation, user-assisted image creation, and automatic image creation.

Designers doing artistic media simulation must analyze the physical property of media in one painting style and then extract proper parameters from these media for simulation. Such as the fluid simulation of media on a paper, transparency factor in different concentration of media on different paper. They have to precisely measure these parameters from real experiments and test their program repeatedly. So this kind of system allows users to choose their painting tools to draw their artwork interactively.

User-assisted image creation system helps users generate artistic image in the painting process. It provides applications which include skills or techniques of human artists to let users apply these effects to a painting. Automatic image creation pre-defined the artistic knowledge of human artists and users can create artistic images automatically. Usually this kind of system needs a specific input source including 3D models or 2D images then users can apply filters such like watercolor, pen-and-ink, hatching and half-toning to input source to generate specific artistic image. Our research has a huge correlation about automatic image creation so we will introduce more related works in this area.

Curtis et al. [4] "Computer-generated Watercolor" simulated the physical property of media and substrate in watercolor as shown in figure 1.2. He also simulated several effects in watercolor painting process such like edge darkening, back runs, granulation, flow effects and glazing. But his work only focuses on 2D image processing and does not use the information of 3D model in 3D scene rendering. Hertzmann et al. [12] "Image Analogies" proposed an "analogous" mechanism that receives a source image and a filtered image which based on the source image then his system can apply the style extracted from above two images to another target image as shown in figure 1.3. Ostromoukhov et al. [15] "Digital Facial

Engraving" introduced an interesting work to convert a facial image to traditional copperplate engraving style as shown in figure 1.4. All these researches we mentioned above only worked on 2D image space but they provided an alternative image rendering method and established a strong basis for further advanced NPR researches.

Fig 1.2 Left: Convert a low resolution 2D image into watercolor style. "Computer- generated Watercolor" [4] by CURTIS, C. J. in SIGGRAPH 97. Right: The engraving style using a Venus photo. "Digital Facial Engraving" [15] by OSTROMOUKHOV, V. in SIGGRAPH'99.

Fig 1.3 *A* is the source image; *A'* is the filtered version of *A,* and user input a target image *B*; the analogies mechanism will apply the same filter on *B* to get *B'*. "Image Analogies" by HERTZMANN, A. in SIGGRAPH'01.

There is another important artistic style in NPR called "pen-and-ink" [5, 6, 7, 8, 9] illustration shown in the following figure 1.4. Most of these researches focus on

rendering 3D models with pen-and-ink style depending on its parametric properties. Another research related to pen-and- ink is pencil rendering [10, 11]. Praun et al. [13] “Real-time Hatching” in Fig 1.4 applies a pencil hatching style to 3D models. The contribution of his work is to avoid discontinuity when changing the tones of textures. The idea “tonal-maps” is inspired from the paper “Non-photorealistic Virtual Environments” by Klein et al. [14]. The original idea of “tonal-maps” is “art-maps” which is proposed by Klein et al. and it will not change the brush size of a texture while scaling.

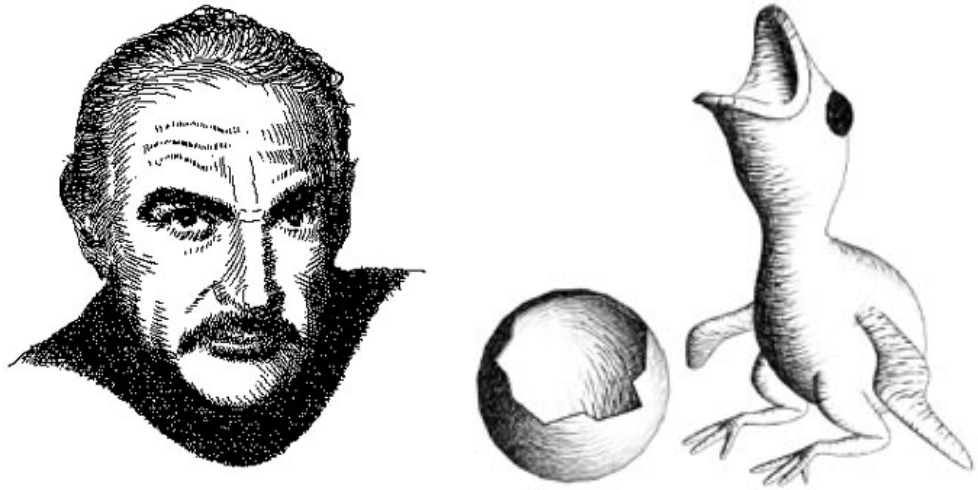


Fig 1.4 Left: A pen-and-ink illustration from a photo portrait. “Interactive pen-and-ink illustration.” by SALISBURY, M. P. in SIGGRAPH 94. Right: Applying hatching style on 3D models. “Real-Time Hatching” by PRAUN, E. in SIGGRAPH’01.

Other researches in NPR are about finding silhouettes in 3D models quickly and applying various line styles to silhouettes. Markosian et al. [17] “Real-Time Nonphotorealistic Rendering” proposed a new algorithm that can detect visible lines and surfaces in real-time, and it needs not travel all edges in the model to find the silhouette lines by a special walking algorithm. As the result of his paper, he applied several styles of strokes to the silhouette edges on 3D models. Northrup et al. [16] “Artistic Silhouettes: A hybrid Approach” used a hybrid method to draw silhouettes. First they used image-based approach to find visible line paths and these paths have the resolution of object-space edges. Their algorithm can also solve the zigzag line problem. The final result can render at interactive rates. Kalnins et al. [28] provided a direct interface for content creators to easily draw NPR strokes on an existed 3D model. Four efficient algorithms were proposed in this paper for detecting and rendering silhouettes, synthesizing stroke styles by example, view-dependent hatching

under artistic control and simulating various types of natural media.

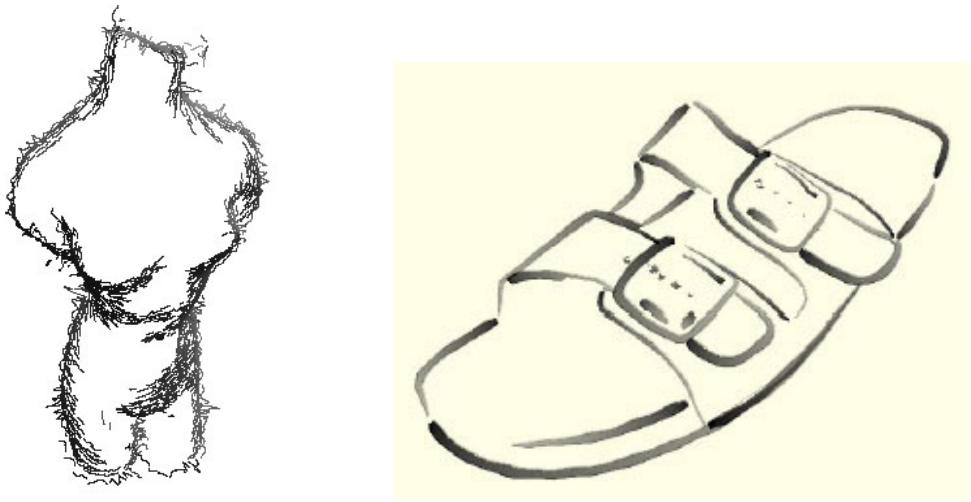


Fig 1.5 Left: Human figure with expressive outline and shading strokes. “Real-Time Nonphotorealistic Rendering” by MARKOSIAN, L. in SIGGRAPH 97. Right: Ink-wash style applies on silhouettes. “Artistic Silhouettes: A Hybrid Approach” by NORTHRUP, J. in NPAR’00.

Algorithms and mechanisms of these NPR researches are inspired from the real painting process. Depending on different lighting environment, different viewing direction and different model surface, every painting style has its own rules to represent the image. Programmers do the same thing as artists, but the only difference is to convert these painting rules into several routines and try to simulate these artistic processes in algorithms.

We can easily find that in previous researches, almost all NPR systems simulated the western painting arts, such like watercolor, pen-and-ink, hatching and oil painting, etc. It is a pity that few researches in NPR field are about Chinese painting except these two years. So we decide to develop a Chinese painting rendering system in this master thesis.

1.1.2 Introduction to Chinese painting

When we start drawing a Chinese painting, four essential items should be well-prepared for painting. They are Chinese pen brush, Chinese ink stick, Xuan paper(宣紙) and Chinese ink stone. Sometimes we combine color mineral pigment

with original Chinese ink to colorize the whole painting. Xuan paper is made of a special kind of tree skin and its texture has a large difference with other paper used in western painting arts. Xuan paper has good absorbency of water and ink and because of this characteristic of Xuan paper; an artist must conceive a well-composed draft in his mind before painting. During a painting process, an artist should carefully complete the artwork on a paper without any wrong strokes at on go. The Chinese pen brush is composed of a shaft and brush bristles. Bristles using different animal's hair can change the usage of a pen brush. An artist dips few inks on Chinese pen brush to produce dry brush effect, or dips more inks to produce ink diffusion effect depending on style of strokes. Ink is generated from Chinese ink stick and Chinese ink stone by adding little water on an ink stone and an artist rubs ink stick on it. Concentration of ink can be controlled in the process of rubbing an ink stick on the ink stone and the quantity of water.

There are many genres in Chinese painting, for example the Chinese landscape painting, the Chinese flower painting, the Chinese figure painting and so on. The Chinese landscape painting is constructed of waters, mountains, trees, and rocks. However, they were originally intended to be the contrast of other figures but due to the attraction of natural landscape in philosophy and literature, Chinese landscape painting became a popular interest of artists. The Chinese flower and bird painting are originated from a remote past. First discovered on the decoration of colored potteries but the design of painting is very simple and plain. However, this genre of art did not become an entirely separate one on its own merit until mid- and late- Tang Dynasty. Such artists as Bian Luan, Xue Ji and Diao Guangyin had played an important role in contributing to the independence of this genre.

Fig 1.6 "Poet on a Mountain Top." album leaf mounted as a hand scroll. Shen Chou 1427-1509.

Fig 1.7 Birds and leaves by Gerald Liu.

Fig 1.8 Horses by Liu Qiao Yang.

Early-stage painting works nearly all took figures as the main theme such as horses and buffalos. Nevertheless, two opposite painting style aroused in the officialdom and the populace, and there was a big difference between them; sophisticated vs. coarse. The former performed an accurate depiction in lines and controlling of colors whereas the latter showed a bold and unconstrained style in displaying the vivacity of animals.

The difference between the Chinese painting of animals and the other painting styles is that animals are vivid and have their own characteristic, but the landscape painting or the flower painting is not. We can observe that horses in Chinese painting of animals are elegant and unconstrained; the buffalos are steady and harmony. Depending on the creativity of artists, one can add his personal emotion on the painting to convey to the viewer.

By this point of view, we choose 3D animal models to be our source models and render in Chinese painting style automatically. In computer graphics, it is hard to automatically generate expressive image only by few attributes provided from a model, for example a free hand brushwork which is characterized by simple and bold

strokes intended to represent the exaggerated likenesses of objects. Therefore, a sophisticated Chinese painting style is preferred in order to decrease the complexity in the rendering process.

1.2 Contributions

Our automatic rendering in Chinese painting style aims on drawing 3D animal models, and it is a new topic in NPR. In recent years, there are similar researches on the Chinese painting system. Lee [18] developed an oriental black-ink painting system and this system only provided an interactive painting environment to help user paint more easily with various brush tools and ink effects. The system focuses on a user's imagination and creativity. Way and Shih [19] presented a method of synthesizing rock textures in Chinese landscape painting, but users have to specify the contour of mountain and then complete the painting process, or users can paint manually. Way et al. [20] developed another application to simulate the Chinese painting of trees by using silhouette and texture strokes. Unlike their previous system, they began to use 3D tree models to be their source models, and established four reference maps to analyze the information for the bark texture. They can draw various styles of bark texture by defining the texture patterns.

A complete rendering process to transform 3D animal models to Chinese painting style is proposed in this thesis, and does not need to specify any parameters during this process. First, a simplified software implementation is used to extract the silhouettes of a 3D model and a culling algorithm is proposed to eliminate invisible silhouette edges from a specific viewpoint in 3D scene. Stylus strokes with various widths will be generated automatically by applying our stroke mechanisms to silhouette edges. 2D brush model with special effects is then combined with each stroke to generate a Chinese painting style stroke, while interior shading by diffusion is done automatically.

1.3 Application Areas

Almost all cartoon movie films in recent years have applied computer graphics techniques to ease human efforts, such as computer-generated shadows and texture

mapping for cel animation [21, 22]. It is more popular to make pure computer graphics films in these years and artists want to show different artistic styles in a film by rendering characters in various ways. The kernel of our rendering process can be a render engine of 3D models by simply specifying the file format of models and the motions of characters in each frame. Unfortunately, there are few self-production CG movies in Taiwan because of lacking shading techniques for models and it is the most important component in the production of CG movies. It is a big pity because we have lots good stories about Chinese history and many famous Chinese fictions that everybody knows to be the scenario of movies. By applying our Chinese painting render engine to the characters and scenes, we can make a movie that has its own characteristic and I believe it will be successful if we can reach that goal.

In all kinds of computer games, the RPG (Role-Playing Game) is the only one that should be localized. A game with Chinese style means it has Chinese story and Chinese scenes and best rendered in Chinese painting style. Therefore, 3D models of game characters can be rendered by our system to transform in Chinese painting style. However, only few 3D RPG games are developed by local Taiwan companies. We still hope that a pure Chinese style RPG games can become available in the market one day.

In 2D image processing softwares such as Photoshop or 3D animation software like MAYA, they have many extra filters or plug-ins to transform images into different styles. Some of these functions are developed by the manufacturer but others are designed by software end-users or researchers. All of them put their new ideas into the plug-ins for the sake of enhancing the originally software and end-users can use these plug-ins without any professional knowledge in advance. Also we can transform our rendering engine into the format of plug-in of MAYA, for example. End-users who install our plug-in can easily apply Chinese painting style to any 3D models or scenes. Because of this kind of extension in software engineering, everybody contributes his research works and finally end-users will get profit by using more and more resources.

1.4 Organization

In chapter 2, a simplified silhouette finding algorithm with its culling mechanism will be discussed. In chapter 3, we link these silhouette segments and grow a width at each control point by applying our stroke mechanisms. In chapter 4, a simulated brush

model will be introduced to combine with strokes and become a Chinese painting stroke along the silhouette of a 3D model. An automatic interior shading pipeline will be introduced in chapter 5 used to paint the interior area of 3D animal models. Chapter 6 will show rendering results transformed from 3D animal models to Chinese painting style. Finally, the conclusion and future works of this thesis are available in chapter 7.

Chapter 2
Finding 3D Model Silhouette

2.1 Introduction

In Chinese painting theory, artists always add silhouette strokes along features such as mountains, trees and so forth. This characteristic is most obvious in painting animals because in a Chinese painting artists put much stress on displaying the "implicit meaning" so that the whole painting tends to be abstracted. This is why artists must paint the silhouette strokes before or after they fill the interior area of an object.

In the following sections we will introduce our algorithms which can dynamically calculate the silhouettes of 3D models as well as data structures used in this algorithm.

2.2 Definition of Silhouette

Silhouette is the most economy way to convey the shape of an object in non-photorealistic rendering. Unlike creases or boundaries, silhouettes can not be pre-calculated before the program running. Because silhouettes are inherently view-dependent, we have to find them every time when the scene or camera changes. First we consider the silhouette on a NURB surface.

At a point on a surface $\sigma(u,v)$, given $V(u,v)$ as the eye vector and $n(u,v)$ as the surface normal, a silhouette point is defined as the point on the surface where $V(u,v) \bullet \sigma(u,v) = 0$ or the angle between $V(u,v)$ and $\sigma(u,v)$ is 90 degrees, as shown in Fig 2.1.

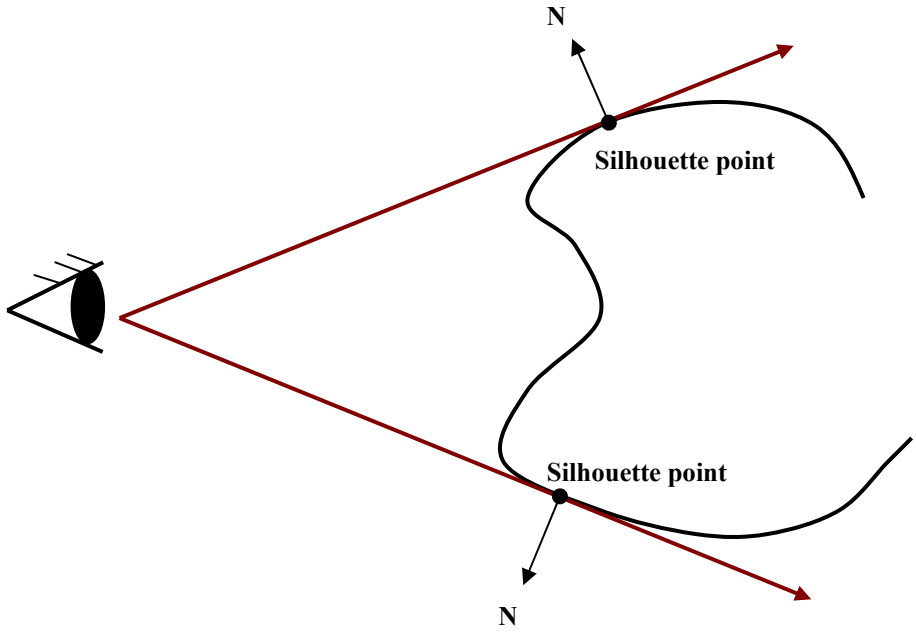


Fig 2.1 Definition of a silhouette: At a point on a surface $\sigma(u,v)$, and given $V(u,v)$ as the eye vector and $n(u,v)$, as the surface normal, a silhouette point is defined as the point on the surface where $V(u,v) \bullet n(u,v) = 0$ or the angle between $V(u,v)$ and $n(u,v)$ is 90 degrees.

But this definition is not useful because all input models in our system are polygon meshes. Markosian and Kowalski [17] generalized the definition for polygon models. But we have to make some assumptions in advance. First, we assume the

model to be rendered is represented by a non-self-intersecting polygon mesh. Second, no edge has more than two adjacent faces at the same time. A silhouette edge is an edge adjacent to one front-facing and one back-facing polygon where a polygon is front-facing if the dot product of its outward normal and a vector from the camera position to a point on the polygon is negative. We can rewrite as $N \bullet V < 0$ where N is the outward normal of this polygon, V is the viewer direction. Otherwise the polygon is back-facing. Or we can write as $N \bullet V > 0$ where N and V are the same as the former. Fig 2.2 illustrates a silhouette edge is shared between a front-facing polygon and a back-facing polygon.

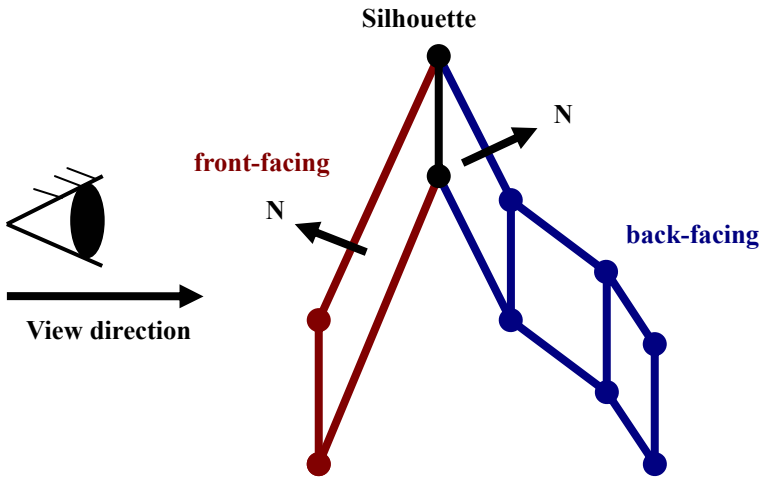


Fig 2.2 For polygons, an edge between two polygons is a silhouette edge if the edge is shared by a front-facing and a back-facing polygon.

2.3 Algorithms

Raskar and Cohen [23] proposed three hardware-supported methods to find silhouette edges. All silhouette edges need to be rendered in black and background should be white. “Hardware-supported” means they use OpenGL depth function for speeding up their algorithms. However, depending on hardware resources will lose much power of controlling these silhouettes. For example we can not apply various width or textures on a long linked silhouette line.

Considering the situation above, we chose the original software implementation

that is to test every edge explicitly in the polygon model while rendering. We found it does not cause any system delay by implementing the original algorithm on a Pentium III 500Hz machine. As a result, we can know which edge in the whole model is silhouette edge now at specified viewpoint. By receiving this edge information in polygon model we can apply our stroke mechanism to these edges. The relative mechanism will be introduced in chapter 3.

Next we will show the data structure for polygon models and it is well-defined for our special purpose.

2.4 Data Structures for Polygon Models

Because we have the requirement of finding 3D model silhouettes which use edge as their basis, we have to extract edge information besides reading vertex, face and vertex normal information of original polygon model. Therefore, when reading face information from a model we additionally add its three edges into our edge structure simultaneously. But we have to carefully remove edges which have the same endpoints with other edges. It costs lots of time in repeating this process at pre-computation section and the length of time is positive-correlated to the number of edges in a model.

In a polygon model, we store several lists such as vertex list, face list, edge list and vertex normal list which are represented in arrays of their own structure. In face structure we record index number of its three vertices and vertex normals; they are stored in the vertex structure and vertex normal structure respectively. We calculate face normal from its three vertex normals for the purpose of doing inner product with viewing vector. The result is used to test whether a face is front-facing or back-facing. Besides, we also add a flag ***bIsFrontFace*** to record it. The flag is dynamic changing every time when a model rotates or camera moves.

Edge structure includes useful data which is convenient for finding silhouette edges. Information of which faces are adjacent to this edge is needed because a silhouette edge is adjacent to one front-facing polygon and one back-facing polygon. Therefore, we can test every edge to see whether its adjacent faces are precisely one

front-facing and one back-facing to make it become a silhouette edge. In addition we need to know 2D position of its endpoints and group number of silhouette. The former is used to check whether endpoints of two different edges are close enough in 2D position during the silhouette linking stage. The latter is used to link silhouette edges which have the same group number. These methods will be introduced in chapter 3 – Stroke mechanism. ***PassCounter*** decides whether we remove this silhouette edge or cut the edge into half.

```
typedef struct ModelObj {
    Vertex** vlist;      // vertex list
    Vertex** vnlist;     // vertex normal list
    Face** flist;        // face list
    Edge** elist;        // edge list
    int e_num;           // edge number
    int vnum;            // vertex number
    int fnum;            // face number
    int vn_num;          // vertex normal number
} ModelObj;

typedef struct Face {
    float x,y,z;         // face normal
    int nverts;          // number of vertex indices in list
    int verts[3];        // vertex index list
    int vnindex[3];      // vertex normal index list
    BOOL bIsFrontFace;   // is this face front face?
    int face[3];         // index of adjacent 3 faces
} Face;

typedef struct Edge {
    int verts[2];        // 2 endpoints of a edge
    Vertex Map2D[2];     // 2D position of 2 endpoints
    bool bShowSil;       // Is silhouette or not?
    int GroupNum;        // group number of a edge
    int face[2];         // index of adjacent 2 faces
    int PassCounter;     // counter of 2 endpoints which pass depth
                         // buffer
} Edge;
```

Fig 2.3 A code sample. Data structure of our system.

2.5 Silhouette Culling

After we implement this software algorithm mentioned above, we can find all edges which are satisfied by the definition of silhouette. However it is not the final results we need. Consider Fig 2.4, silhouette edge at the right side is also adjacent to a

front-facing and a back-facing polygon, but unfortunately it is occluded by faces at left side. Therefore, at our view direction the silhouette edge at right side should be invisible. The OpenGL culling function will not automatically eliminate this edge because we draw silhouettes by our edge structure, not in the triangle mode. We have to implement culling mechanism by ourselves.

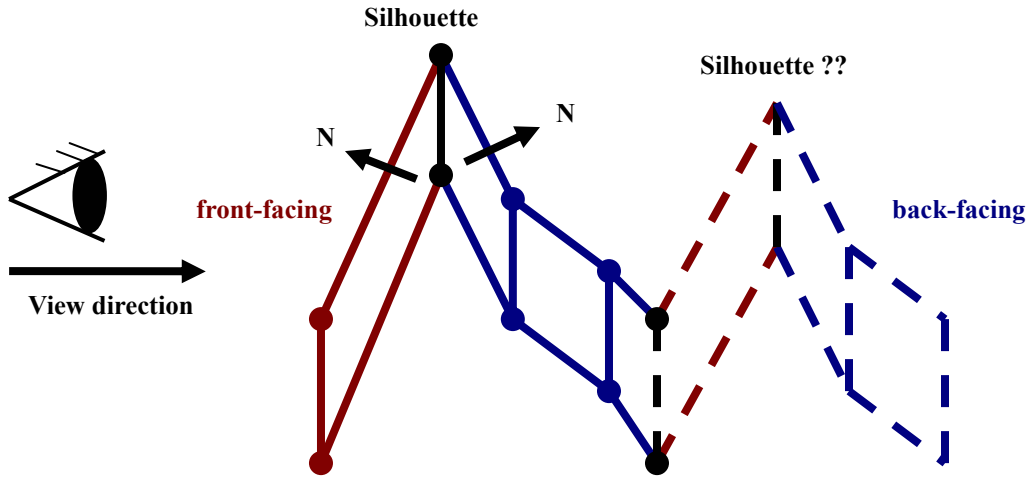


Fig 2.4 By the definition of silhouette, we can find two silhouette edges in the example but silhouette edge at the right side is invisible because it is occluded by faces at the left side.

Fortunately, OpenGL will maintain a depth buffer when we draw the model in triangle mode. We can check depth value by index number of pixel location in current rendering frame, and make it easier to implement our culling function. First we convert 3D-coodinate of both endpoints in all silhouette edges into 2D window coordinate and a depth value. We store both endpoints' depth value in ***Map2D*** of edge structure. Since we know exact pixel location of both endpoints in silhouette edge, next we compare their depth value to the depth buffer on current rendering frame, and three cases will meet during this comparing process as shown in Fig 2.5:

1. Both endpoints pass the depth buffer.
2. Both endpoints do not pass the depth buffer.
3. One endpoint passes the depth buffer but the other doesn't.

It is simple to handle case 1 and case 2; we keep and delete both two endpoints respectively. But in case 3 we want to preserve its visible part of whole edge. Thus we

cut the edge into two halves and it will generate a new endpoint at middle; again we check this new endpoint and its visible endpoint to see whether they can both pass the depth buffer. If not, repeat this operation until it is satisfied with case 1.

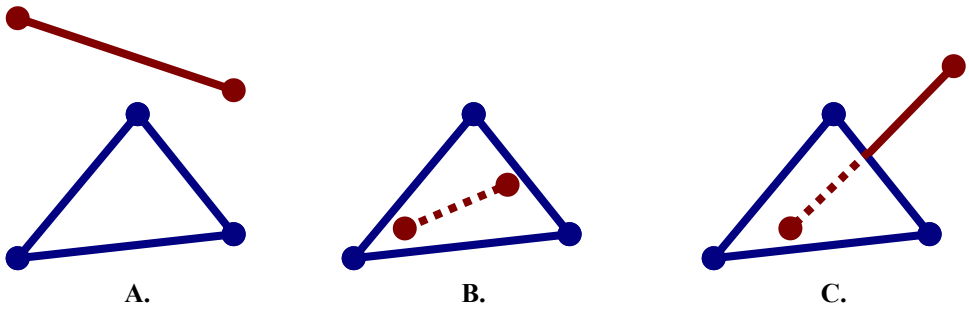


Fig 2.5 Three conditions occur in the depth buffer test. (A) Both endpoints pass the depth buffer, (B) Both endpoints don't pass the depth buffer and (C) One endpoint passes the depth buffer filter but the other doesn't.

One may question about this cutting operation because it seems to be dangerous because it will increase distance between two endpoints which are originally close. This situation is shown in the following Fig 2.6.

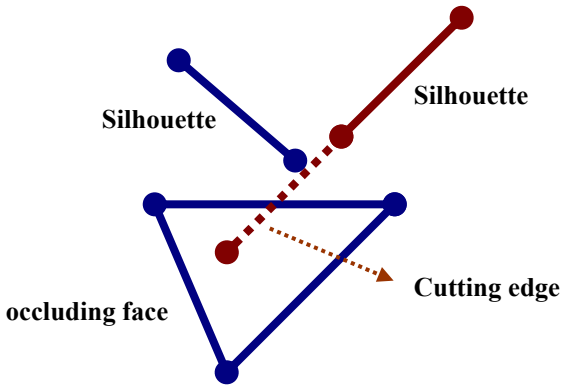


Fig 2.6 The "cutting" mechanism may cause problem because it will increase the distance between two close silhouettes.

The fact is, by testing various models in this cutting operation we found that each silhouette edge is short in our models and the distance it increases may be only one or two pixels. Therefore it won't cause any serious problem like fig 2.6 as shown above because we also set some thresholds in checking the connectivity between two silhouettes.

2.6 Conclusion

Finally we can only find visible silhouette edges in 3D model dynamically when model rotates or camera moves. In next chapter we will connect these silhouette edges into a long linked stroke and grow various stroke widths at each control point of a stroke. By our stroke making mechanism we can make a plain stroke become stylus.

Chapter 3
Stroke Mechanism

3.1 Introduction

In last chapter we can automatically find 3D silhouette edges and eliminate invisible edges in a model at runtime of program. However, these edges are represented in edge segments. Our goal in this chapter is to draw a continuous stroke along silhouette lines and simulate Chinese painting strokes. In section 3.2 we will introduce Chinese painting strokes and in section 3.3 a stroke linking mechanism is proposed to connect these edge segments to a linked stroke. In section 3.4 we present a stroke making mechanism that assigns various stroke widths at each control point in a stroke in order to simulate the shape of Chinese painting strokes and final rendering results will be shown in section 3.5.

3.2 Chinese Painting Strokes

Different kinds of stroke in Chinese painting can represent different characteristics of animals. As shown in following figures, when painting a cattle we use heavy strokes to display feelings of honest and steady, but when painting a horse we use light strokes to represent dynamics of horse running. A Chinese painter Li [24] said "The main point of ink painting lies on the gradation of the ink and the manipulation of the brushwork. To paint is not to copy the outer form of the subject, but to represent the mental conception of the subject in the painter's mind." Therefore, strokes can convey spirit of a painting in an effective way.

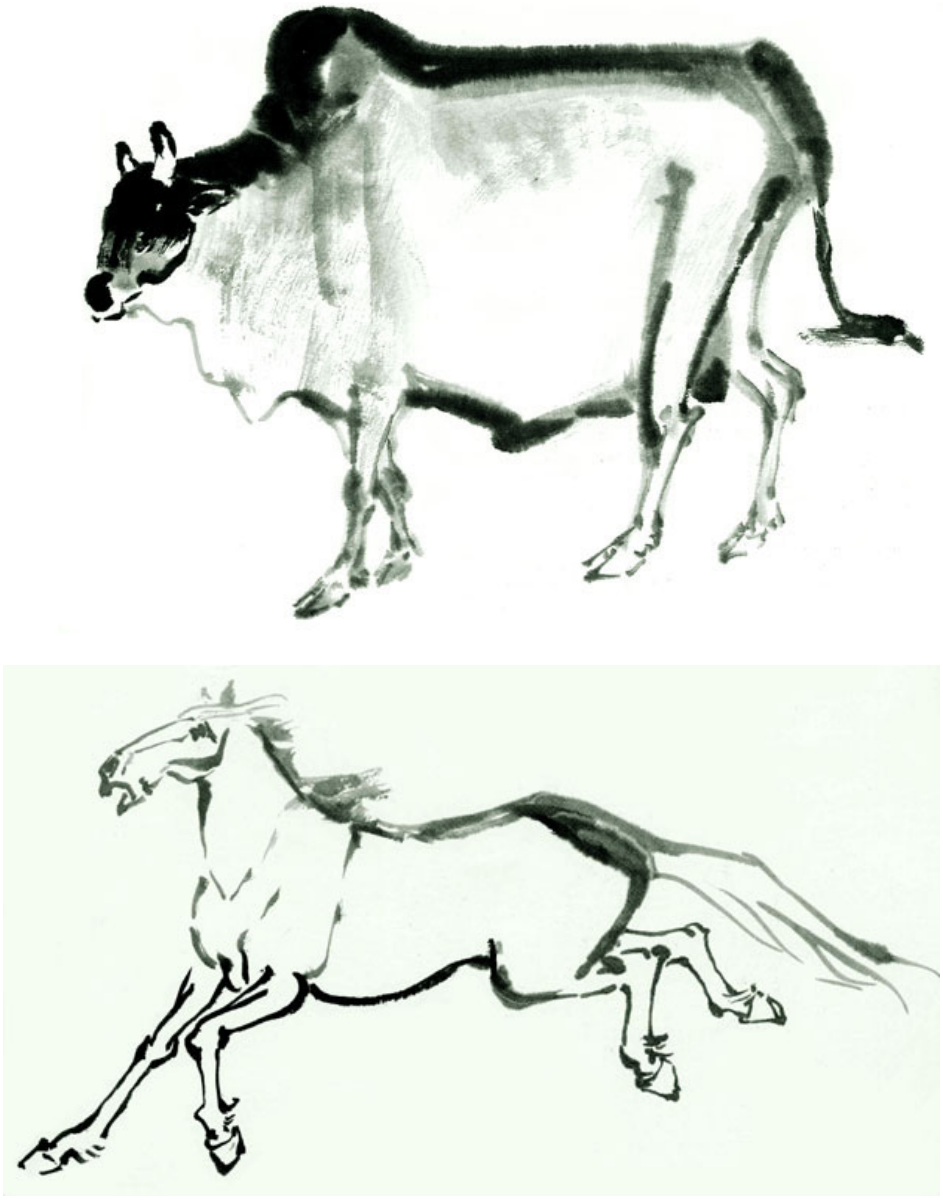


Fig 3.1 Chinese painting of a cattle and a horse by Li Chi-Mao [20].

3.3 Stroke Linking Mechanism

In chapter 2, we have already extracted silhouette edges from a model. However, these silhouette edges are stored as segments in our edge structure and there is no any correlation between them. Therefore we must link these silhouette edges which are connected in 2D-coordinate to a stroke. In our definition, a stroke is composed of a group of edge segments which are connected and both endpoints of an edge are control points in a stroke. 2D-coordinate of both endpoints in each visible silhouette edge have been calculated in chapter 2 and are also stored in our edge structure. In order to store a stroke, we should record its control points in sequence. As a result, we make a link list for storing control points conveniently.

Following fig 3.2 shows a complete procedure for the stroke linking stage. We store all silhouette edges in a waiting pool and take one edge out which has not been classified to any group number to be an initial edge, and push its 2D-coordinate of both endpoints into top and bottom of link list in the meantime. Next we explicitly search silhouette edges in the waiting pool by ***SearchIndex*** to check whether this edge is connected to the impermanent stroke in link list. We compare its 2D-coordinate of both endpoints with top and bottom element in link list to check their connectivity. It is also important to set an appropriate threshold to decide whether these two points are connected together. As we mentioned in section 2.4, the value of threshold should tolerate error generated from silhouette cutting and be careful not to link two silhouette edges that are not connected originally. In our case, we take a threshold of two pixels around a point to be an acceptable range for checking connectivity.

```
For(all silhouette edges){
     if(group number == 0){
          [Insert both two endpoints of this edge into the list]
          while(SearchIndex < EndOfEdgeIndex){
               [Check whether it is connected to stroke]
               if(match){
                    [Insert point into list by two cases and give this edge the same group
                     number. Set SearchIndex to be the first index]
               }
               else
                    [Increase SearchIndex by 1]
          }
     }
}
```

Fig 3.2 The pseudo code of our stroke linking algorithm.

Once we found an endpoint of current edge that is connected with the impermanent stroke in link list, we push another endpoint into this control point list. When pushing an endpoint into the link list, two cases will arise due to the directional property in link list. As shown in following fig 3.3, there are only two control points inserted into link list at the beginning, called point A and point B which are on the top and bottom of link list respectively. In case 1, we found a silhouette edge whose endpoint has the same 2D-coodinate with point B that is already existed in link list, we called this point B'. Now we know this silhouette edge is connected with the stroke in the link list, therefore we have to push its control points into the list. However, we only push another point C of this silhouette edge into the link list in a bottom direction. By following this insertion rule, we can guarantee that the following control points are inserted in a proper sequence. It is the same in case 2; we only push point C into link list in a top direction. After this insertion, point C becomes the new bottom in case 1 or the new top in case 2.

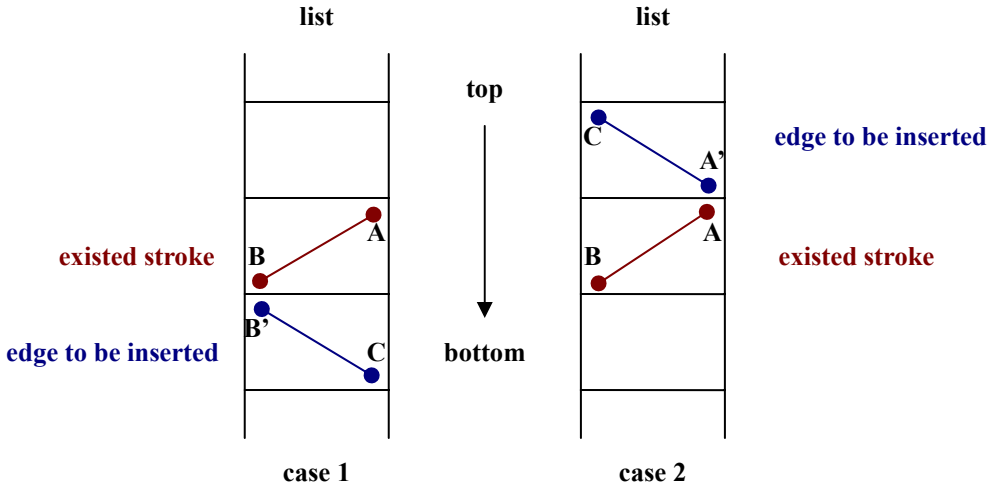


Fig 3.3 In order to keep the sequence of control points when inserting a new control point, we store data in two directions depending on these two cases. Point A and point B represent the original top and bottom point in the link list.

When a new control point is inserted into link list, we search for all silhouette edges again from first index in waiting pool because the top or bottom element of link list has been changed. While every silhouette edge in waiting pool has been searched, it means we have inserted all control points that are connected together, and all edges in link list will have the same group number to identify that they belong to the same stroke. Then we repeat this process until all silhouette edges have been classified to a

group number.

3.4 Stroke Making Mechanism

After the stroke linking stage, we have the information about 2D-coodinate of control points in a stroke. But without any post-processing mechanism, it is only a long linked stroke line and has no style and width. Therefore, a stroke making mechanism is needed to automatically generate width at each control point in a stroke. Strassmann [25] simulated several effects of brush deposited ink onto paper and produced Japanese painting images called “sumi-e”. However, his method must work by having a user input lists of positions and pressure via keyboard. Two cubic B-splines are then computed, one for the positions and one for the pressure values and distance along strokes. The position spline values are used to find smoothly varying positions along a stroke, while the pressure and distance spline is used to find width values for each derived position. At each position point along a stroke, unit normals are found, and the corresponding width value is used to find coordinates of the borderline of the stroke. An example of a stroke is shown in Fig 3.4.

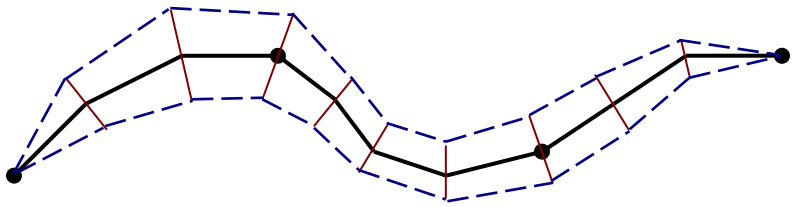


Fig 3.4 A stroke defined by four control points with intervening nodes generated by a cubic spline.

In order to grow various widths at every control point, we make some modifications to Strassmann’s method because we already have positions of control points in a stroke. Observing Chinese painting strokes, a stroke always has its both ends thin but fat in the middle. Depending on this principle of Chinese painting we can automatically assign pressure values at each control point. We also make an assumption that when user put more pressure on a Chinese pen brush, a bigger stroke width will be generated on the paper.

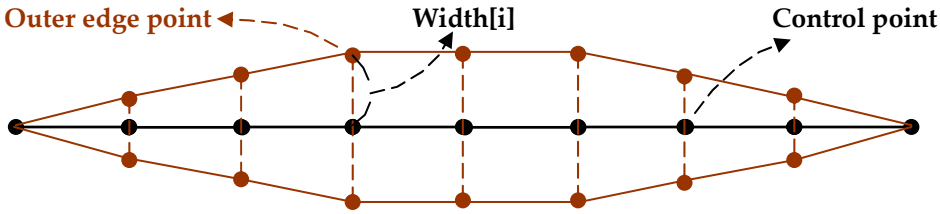
We set two parameters for initialization and limitation of stroke width calculation: ***width_step*** and ***MAX_WIDTH***. The former defines an increment of stroke width from the last control point and the later defines the maximum stroke width value of this stroke. We calculate stroke widths according to following equations:

$$add[i] = \begin{cases} +width\_step, & if \quad i < \frac{1}{2} \times ncp \\ -width\_step, & if \quad i \geq \frac{1}{2} \times ncp \\ 0, & if \quad width[i] \geq MAX\_WIDTH \end{cases} \qquad \text{..... (1)}$$

$$width[i] = \begin{cases} 0, & if \quad i = 0, i = ncp \\ add[i] + width[i-1], & else \end{cases} \qquad \text{..... (2)}$$

where ***i*** is the index of control point of this stroke, ***ncp*** is the number of control points, ***add[i]*** is an 1D array which defines an increment of stroke width in this control point, and ***width[i]*** is another 1D array which defines stroke width in this control point.

Equation (1) shows that our mechanism continuously adds on a constant width ***width_step*** point by point in sequence until the index is moving beyond the middle point of this stroke, or the stroke width at this control point ***width[i]*** exceeds the limitation ***MAX_WIDTH***. In other words, stroke width at this control point is more than the last one by an increment ***width_step*** in the first half, and it is reverse in the second half of control point list. Equation (2) simply indicates that we add an increment ***add[i]*** and the stroke width at last control point ***width[i-1]*** together to get the current stroke width of this control point ***width[i]***. Fig 3.4 illustrates a whole process of our stroke making mechanism by a simple example.



i	0	1	2	3	4	5	6	7	8
Add[i]	0	2	2	2	0	0	-2	-2	-2
Width[i]	0	2	4	6	6	6	4	2	0

Fig 3.5 A stroke with 9 control points. We set ***width_step*** to 2 and ***MAX_WIDTH*** sets to 7. The following table shows the current values of ***add[i]*** and ***width[i]***.

2D-coordinate of the outer edge point can be calculated from a control point and ***width[i]*** related to it. We calculate a vector which is perpendicular to the direction of an edge segment and increase a distance ***width[i]*** on both side of an edge from the control point to get outer edge points. These outer edge points will form a stroke outline which is shown in fig 3.5. After we calculate all outer edge points of a stroke, the fastest way to fill the cover area inside a stroke is to draw a series of quadrilaterals representing the main body of a stroke, and two triangles representing the beginning and end of a stroke and using OpenGL drawing primitive functions will speedup this filling process. But in next chapter we must apply brush model to strokes, therefore strokes can not be rendered in 3D primitives. We still have to put pixels into the frame buffer and it will increase the load of our system.

During the runtime of program users can change two default parameters: ***width_step*** and ***MAX_WIDTH*** to get different styles of strokes such as the heavy one and the light one. Users can also change the color of strokes through our user interface.

3.5 Results

We show some rendering results after applying our stroke mechanism discussed in this chapter. Until now, this system can be run at interactive rates on PC with Pentium 933MHz and GeForce 256 display card.

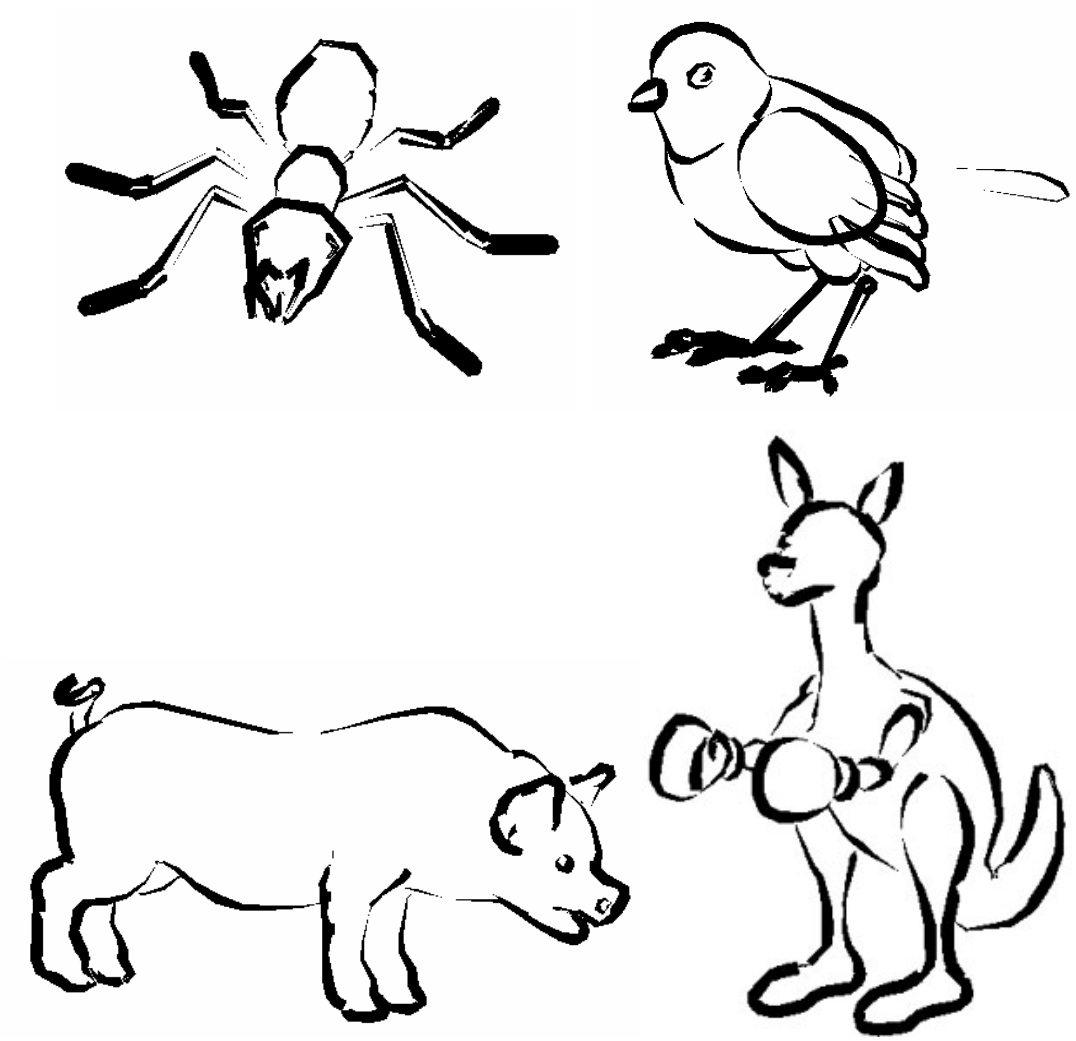


Fig 3.6 By passing stroke linking stage and stroke making stage, we automatically transform 3D animal models into silhouette stroke image.

Chapter 4
Brush Model

4.1 Introduction

The Chinese pen brush is an essential element in Chinese painting and can not be replaced by other brushes because by using Chinese pen brush artists can apply calligraphic techniques in paintings. The most difference between Chinese painting and other paintings is the usage of pen brush and its representation when touching the paper. Thus we have to add a model of Chinese pen brush in our rendering system to simulate the phenomenon of a real Chinese pen brush moving along the path of a stroke and deposit ink on the canvas. In section 4.2 we will introduce the physical structure of Chinese pen brush. In section 4.3, 4.4, a simulated model of Chinese pen brush with its movement mechanism is proposed to use in our system. In section 4.5 we will combine brush model with strokes to make Chinese painting strokes and two special effects in brush model called dry brush effect and turning effects will be introduced in section 4.6.

4.2 Chinese Pen Brush

The Chinese pen brush is composed of shaft and brush bristles. Brush bristles are made from the soft hair of animals such as wolf hair, rabbit hair or horse hair. The length of brush bristles is about 5 or 6 centimeters in usual. The brush bristles will form a pointed end after user dips bristles into black ink. While painting, a user can easily control the size of contact region on a paper from a little point to a whole area by bringing different pressures on a Chinese pen brush. A user must carefully control the quantity and the proportion of water and ink in brush bristles. When drawing a line, a user dips few black inks into brush bristles. When drawing an area, a user should dip more inks into brush bristles. Or a user can change the traditional usage of Chinese pen brush to his own method and it will cause different appearance in his painting.

4.3 Simulating a Brush Model

Lee, J. [18] proposed a very precise 3D brush model with bristles to simulate Chinese pen brush. He also simulated physical property of bristles such as deformation and ink diffusion on paper in his model. These physical properties make his interactive painting system more realistic in painting strokes. However, in our rendering system we need not simulate physical properties of brush bristles and calculate bristle deformation because our system does not focus on the interactive painting environment such as showing brush on the screen or using force feedback joystick. We only need to simulate the contact region where a brush touches a paper and combine it with strokes to make Chinese painting strokes.

Here we refer Way's and Shih's [19] method to simulate the contact region of Chinese pen brush and its movement mechanism. We also use the same 2D ellipse model with Way's to simulate contact region and we distribute brush bristles in this 2D ellipse. Consider the following figure 4.1, an ellipse $\boldsymbol{C}$, where center is $\boldsymbol{O}$, $\boldsymbol{A}$ and $\boldsymbol{B}$ are short axis and long axis of $\boldsymbol{C}$ respectively.

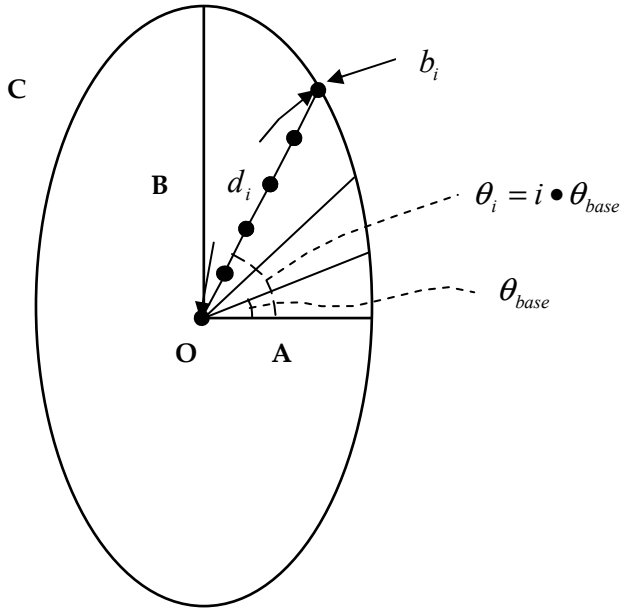


Fig 4.1 Brush Model

We distribute the Chinese pen brush bristles inside ellipse ***C*** and denote as b_i. Each b_i can be represented uniquely in polar coordinate with respect to the center of ellipse ***C***.

$$b_i = (d_i, \theta_i) \quad \text{..... (1)}$$

where *i* is the index of bristles, d_i is the distance from center ***O*** to b_i, and θ_i is the angle between x-axis and $\overline{ob_i}$. We divide the central angle of ellipse ***C*** equally into *i* parts and each part has base angle θ_{base}, so θ_i can be taken as $i \bullet \theta_{base}$. For each θ_i we put numbers of b_i along the radius as shown in figure 4.1. Here we can modify relative parameters to change the appearance of a brush model such as numbers of b_i on θ_i, angle of θ_{base} and length of short axis and long axis. If we need to lighten the color of strokes, a large base angle θ_{base} is set and it will cause less number of b_i distributed on brush model. On the other hand, we take a small base angle θ_{base} to make the color of strokes darker in the above example.

After setting these parameters, we have to calculate the position of each b_i inside ellipse ***C*** respect to center ***O***. Consider Fig 4.1, assume the 2D-coordinate of center ***O*** in contact region is (o_x, o_y), the position of each bristle b_i can be

calculated by the following equation:

$$\begin{cases} b_{i,x} = o_x + b'_{i,x} = o_x + [d_i \bullet \cos\theta_i - d_i \bullet \sin\theta_i] \\ b_{i,y} = o_y + b'_{i,y} = o_y + [d_i \bullet \sin\theta_i + d_i \bullet \cos\theta_i] \end{cases} \quad \cdots\cdots\cdots\cdots\cdots\cdots (2)$$

where $b_{i,x}$ and $b_{i,y}$ are the x-position and y-position of b_i respectively. In equation (2), a temporary result $(b'_{i,x}, b'_{i,y})$ will be first calculated to use local coordinate with its center ***O*** being the origin (0, 0). As long as we decide the position of ***O***, we can calculate the exact position of each b_i by equation (2) and we draw these b_i with black pixels then a simulated brush contact region is painted at its center ***O***. Of course, it is not enough to only simulate the brush contact region. We have to fit it into strokes and do some effects during painting in order to make this simulation more realistic.

4.4 Brush Movement Control Mechanism

During a painting process, center ***O*** of the brush model will move depending on the current position of stroke. We can simply achieve this goal by changing (o_x, o_y) and calculate new b_i by equation (2). Besides translation, when traveling along the stroke trajectory the brush model may also change its orientation as shown in following Fig 4.2.

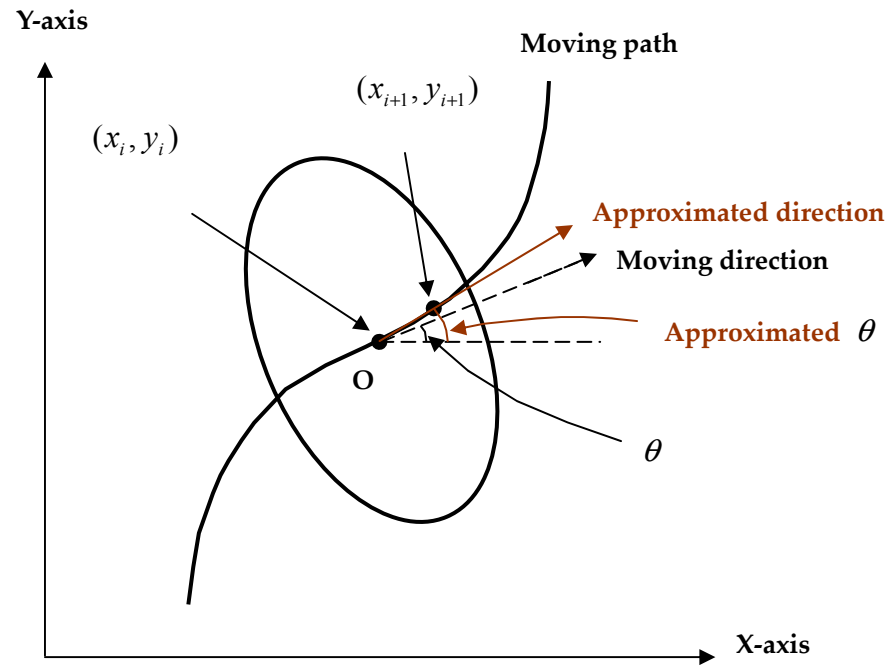


Fig 4.2 Brush model moves along the stroke from (x_i, y_i) to (x_{i+1}, y_{i+1}) and rotate by θ.

Assume (x_i, y_i) and (x_{i+1}, y_{i+1}) in Fig 4.2 are two control points on a stroke, and we want to move the brush model from (x_i, y_i) to (x_{i+1}, y_{i+1}). First we should initiate center ***O*** of brush model at point (x_i, y_i). The rotate angle θ of brush model is determined by tangent line of stroke moving path at (x_i, y_i). Unfortunately we can not know precise value of rotate angle θ at (x_i, y_i), but we can approximate the moving direction by an extended line from (x_i, y_i) to (x_{i+1}, y_{i+1}). Therefore, we can make an approximation of $\tan\theta$ at point (x_i, y_i) by:

$$\mathrm{T}(x_i, y_i) = \frac{(y_{i+1} - y_i)}{(x_{i+1} - x_i)} \quad \ldots\ldots\ldots\ldots\ldots\ldots (3)$$

The rotate angle θ can be approximated by:

$$\theta = \tan^{-1}(T(x_i, y_i)) \quad \ldots\ldots\ldots\ldots\ldots\ldots (4)$$

Once we know the approximated value of θ, the new position of each b_i can be calculated by the following equation (5):

$$\begin{cases} b_{i,x} = o_x + [b'_{i,x} \bullet \cos\theta - b'_{i,y} \bullet \sin\theta] \\ b_{i,y} = o_x + [b'_{i,x} \bullet \sin\theta + b'_{i,y} \bullet \cos\theta] \end{cases} \quad \ldots\ldots\ldots\ldots\ldots\ldots (5)$$

where $(b'_{i,x}, b'_{i,y})$ is the position in local coordination with center $\boldsymbol{O}$ being the origin (0, 0). By applying these equations, we have the ability to locate the brush model in any stroke positions with translation and rotation. We can also scale the long axis of the brush model in order to fit in different stroke widths.

4.5 From a Brush Model to Strokes

As we mentioned in chapter 3, a stroke is composed of several control points and each of them has different stroke widths. Since we have the ability to rotate, translate and scale the brush model, it also means we can fit the brush model into any stroke even they have different size and different rotate angle θ. Consider the following sample stroke in figure 4.3, point A, B, C, E are four control points in the last half of a stroke and the brush model will move along trajectory $\overrightarrow{AB}$, $\overrightarrow{BC}$, $\overrightarrow{CE}$. Assume that the brush model are located at control point A now, we should change its attributes at A by scaling its size and rotating an angle θ so that the short axis of brush model can match the direction of $\overrightarrow{AB}$. Only as the brush model moves to a new control point, it will change its orientation. When moving the brush model between two control points, it goes one pixel at a time and the size and position of the brush model will be interpolated in the traveling process, for example point D in figure 4.3.

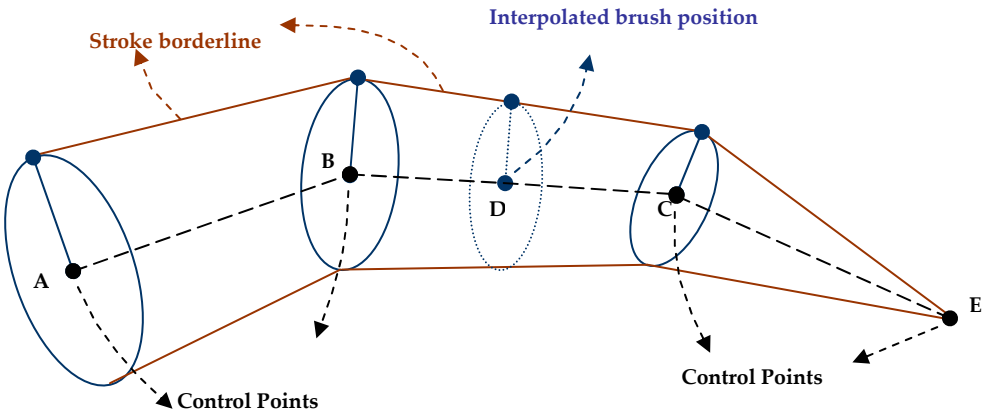


Fig 4.3 Fitting brush model into stroke. We locate the brush model at the control point of stroke, and then we interpolate position and size of the brush model between two control points.

The brush model keep moving, scaling and rotating when it travels along the moving trajectory. Each time it scales its long axis to fit the stroke width, we have to re-calculate ellipse equation to retrieve the position of each bristle b_i inside brush model. Smaller size of brush model will not decrease the number of bristles in brush model because we scale the long axis will only shorten the distance d_i between each bristle on the same radius. It is the same situation when we use the Chinese pen brush, the same number of bristles touch the paper even if a smaller area is painted. Thus the amount of calculation is the same and independent to the size of brush model. We have to do this calculation every few steps because the stroke width of each control point is different. Unfortunately it causes a large overloading in our system. By designing several patterns of brush model for dynamic brush size may be a good solution to ease this overloading.

4.6 Special effects

Until now we have simulated the brush model and implemented 2D affine transformation to fit into strokes, but it is still a long distance from simulating realistic Chinese painting strokes. Due to special characteristics of Chinese pen brush such like dry brush effect and brush turning effect, we have to add these characteristics in our model to increase the quality of simulation. They will be discussed separately in next two subsections.

4.6.1 Dry Brush Effect

In Chinese painting there is an important technique called “dry brush effect” and it makes the whole painting look roughly. We found that in many Chinese painting of animals, this painting technique is widely used in drawing silhouette strokes. Artists usually dip few inks onto brush so that ink deposited from bristles is discontinuously. This effect can also attract a viewer to especially appreciate strokes in a Chinese painting.

Referring to Hsu's master thesis [26], here in Fig 4.4 is our implementation of dry brush effect. We set a parameter ***gap_size*** for every bristle in the brush model. The parameter ***gap_size*** indicates whether a bristle deposits inks on paper or not. When ***gap_size*** equals to zero, it means this bristle has the chance to be drawn. On the other hand, ***gap_size*** is non-zero indicates this bristle will not be painted and when moving brush model pixel by pixel, ***gap_size*** will be decreased by one at each step.

```
for(each bristle in brush model)
{
     // get gap_size for every bristle
     if(gap_size == 0)
     {
          rand1 = rand();
          if(rand1 > 20000)
               gap_size = rand1 % 5;
     }
     else
          gap_size -= 1;

     // when gap_size equals to zero means this bristle touches on paper
     if(gap_size == 0)
     {
          DrawPixel()
     }
}
```

Fig 4.4 Pseudo code of dry brush effect.

4.6.2 Turning Effect

This technique "turning effect" is widely used in painting strokes. While painting turning strokes, artists always like to slow down the speed of brush and put a little more pressure on the brush then more ink will be deposited on the canvas. We consider brush pressure ***P*** to be parameter and it varies from 0 to 1. It determines whether this bristle touches the paper in the contact region and the amount of ink deposited on it. For each bristle b_i we calculate the pressure weight W_p by its relative position of contact region. W_p also varies from 0 to 1 and then it is multiplied with the original color of bristles, if the value of W_p is bigger, the color of this bristle is deeper. Otherwise, bristles with zero W_p do not touch the paper. We

can calculate W_p by the following equation:

$$W_p = \begin{cases} 0 & ,if \quad \dfrac{|b'_{i,y} - B|}{2 \bullet B} \geq P \\ \left(\dfrac{|b'_{i,y} - B|}{2 \bullet B \bullet P} \right) \bullet (P-1) + 1 & ,if \quad \dfrac{|b'_{i,y} - B|}{2 \bullet B} < P \end{cases} \quad \cdots\cdots\cdots\cdots\cdots\cdots (6)$$

where $b'_{i,y}$ is the y-coordinate of b_i in the contact region, and ***B*** is the length of long axis.

Before we implement turning effect, we have to determine the value ***P*** (brush pressure) at each step when brush model moving on a stroke. Larger ***P*** means the pressure is bigger on that point and the color will be darker also. We decide to assign larger ***P*** at control point whose included angle of adjacent two edges is smaller. We also interpolate this pressure value between two control points to get a smooth stroke line.

4.7 Results

The following result images are rendered by applying our simulated brush model on strokes with dry brush effect and turning effects. In the result image, we divide ellipse brush model into 28 equal parts and there are 7 bristles distributed on each radius.

Fig 4.5 Apply brush model with dry brush effect and turning effect to silhouette strokes.

Chapter 5
Interior Shading

5.1 Introduction

Until this chapter, our system has the ability to make Chinese painting strokes along silhouettes and apply a simulated brush model to strokes. Although these mechanisms applied to 3D animal models have already provided good visual effects for representing Chinese painting style, a large area inside the animal model is still kept blank. From observing real Chinese paintings of animals, we found that artists prefer filling the interior area of an animal to only drawing outside silhouettes because good interior shading can convey the appearance of an animal better, for example the color of an animal and the lighting condition. During a painting process, artists first paint the interior area with a light color to make a bottom layer on a paper, and use different levels of dark colors to emphasis the lighting or important features of an animal. At last, artists draw silhouette lines around the outside shape of an animal.

In this chapter we try to render the interior area in a fair way and will not draw

too many strokes in detail. We believe while everything in an artwork is generated by computer automatically, this artwork will become valueless. Therefore the main target of our system is to automatically complete routines which are artists always do in a painting process such as drawing silhouette lines and painting bottom colors in the interior area. We prefer a user to add strokes on the output image generated by our system and develop interesting artworks.

5.2 Interior Shading Mechanism

Fig 5.1 Chinese painting of animals with interior shading. We found that artists always use only two or three levels of colors to paint the interior area.

Considering figure 5.1, we found that most of the area inside an animal body is filled, and filled with only two or three tones of colors. During a real painting process, artists always take a lighter color to be the bottom color for the sake of declaring the range of an interior area and add some darker strokes upon the bottom layer to emphasis special features or dark areas. Therefore we decide to use only 2D pixel color information to simulate interior shading instead of retrieving 3D model properties. Three stages are proposed in our interior shading pipeline: color

quantization, ink diffusion and box filtering as shown in figure 5.2 and they will be introduced in following paragraphs.

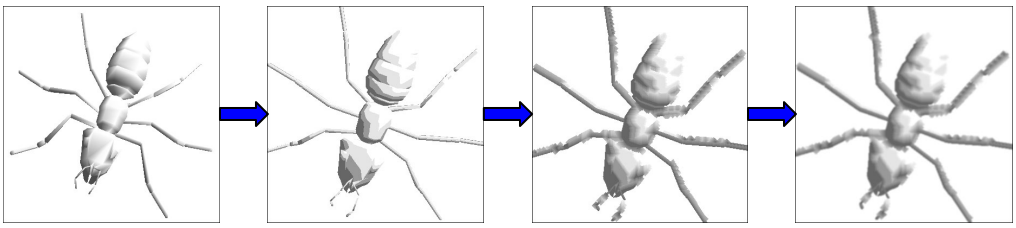


Fig 5.2 Interior shading pipeline

In order to get pixel color information in the interior area, we render 3D animal model with general Gouraud-shading. However, we increase parameters of diffuse lighting and specular lighting whereas decreasing the parameter of ambient lighting. The purpose is to create a large contrast in the whole render image so that the color distribution range is enlarged and the color distribution histogram is equalized, as shown in figure 5.3.

We have mentioned that artists only use few tones of colors when painting the interior area in a Chinese painting process. Therefore we quantize all pixels in the interior area into four color levels. Color ranging from pure black to pure white is distributed from 0.0 to 1.0. From the color distribution histogram as shown in the top of figure 5.4, we set three thresholds for color quantization: 0.4, 0.6, 0.9. A pixel located at any place in color distribution histogram will be quantized to a relative threshold color value. The color-mapping table is also shown in the bottom of figure 5.4. After color quantization, an image that is originally rendered in Gouraud-shading will be converted into a four-color image as shown in figure 5.5.

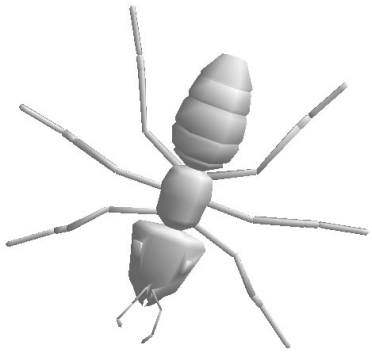
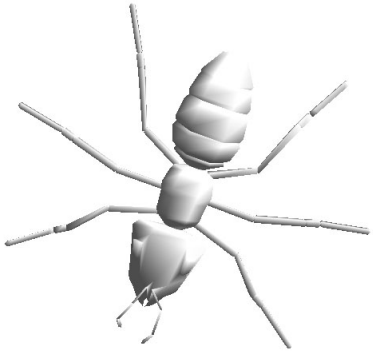
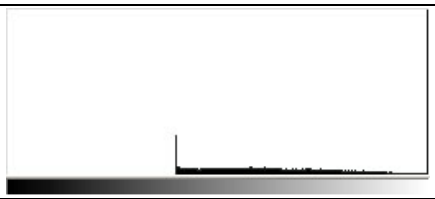
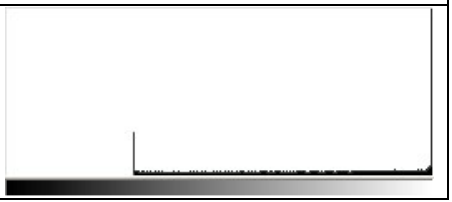
	3D model with normal lighting	3D model with large diffuse and specular lighting
Gouraud-shading image		
Color Distribution Histogram. Left to right: pure black to pure white		

Fig 5.3 Two Gouraud-shading images with different lighting parameters and the color distribution histogram related to the image. The right one uses large diffuse and specular lighting value so that the color distribution range is wider than the left one.

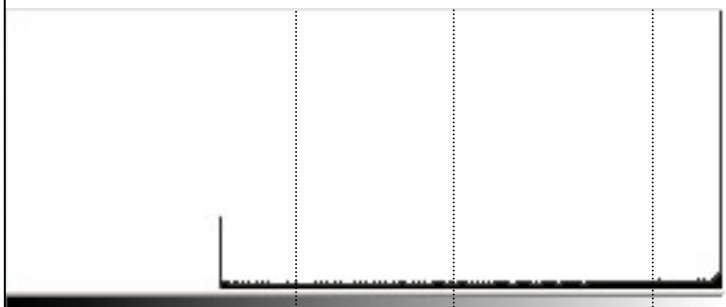
				
Color Range:	0.0~0.4	0.4~0.6	0.6~0.9	0.9~1.0
Quantization:	0.4	0.6	0.9	1.0

Fig 5.4 We define three thresholds in color distribution histogram: 0.4, 0.6, 0.9 and we assign each pixel color to a threshold value according to their location in color distribution histogram.

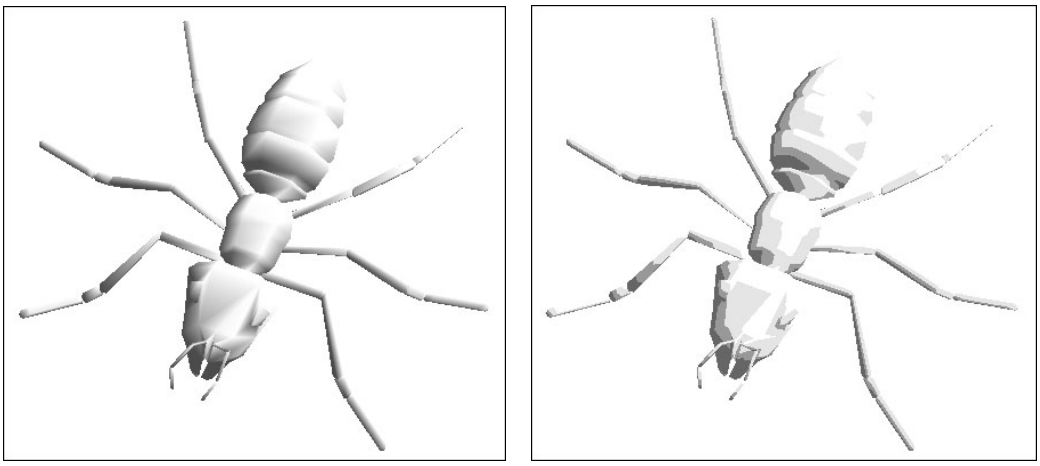


Fig 5.5 We quantize an original Gouraud-shading image (Left) to a four-color image (Right).

Consider the right image in figure 5.5, we found that borders between different color levels are obvious. From the interior shading result of a real Chinese painting, we found that every time when artists add a darker color on an existing color level, the darker level will blend with others at borders and its ink will diffuse to other levels in an outward direction. We need to simulate this phenomenon in our interior shading pipeline because it is also an important characteristic in Chinese painting.

The basic diffusion equation is represented as the following equation:

$$\frac{\partial q}{\partial t} = D\nabla^2 q$$

where ***q*** is the ink quantity, ***D*** > 0 is constant and referred as the diffusion coefficient. The operator ∇^2 is called the *Laplacian*.

In order to simulate ink diffusion, we first define two parameters: ink quantity and water quantity. Ink quantity is used to transfer inks between different levels whereas water quantity determines whether this pixel should transfer ink to its neighbors. Neighbor is defined as four adjacent pixels related to the current pixel. Ink quantity contained in one pixel can be calculated according to its color value:

$$ink = 255 \times (1 - color)$$

where color value is ranging from 0.0 to 1.0. If the color value of one pixel is 1.0, it means pure white and its ink quantity is zero. We also define four levels according to

the quantization levels mentioned above; they are dark level, medium level, light level and paper level; the ink quantities is 153, 102, 25.5 and 0, respectively. For ink diffusion process, we define three rules when transferring inks between different levels:

1. Only when the water quantity of one pixel exceeds a threshold, it will transfer inks to neighbors.
2. Inks will only transfer from a darker level to a lighter level.
3. Only when a difference of ink quantity between current pixel and its neighbors is bigger enough, it will transfer inks to its neighbors.

In order to maintain stability in ink diffusion stage, one pixel will not decrease its ink and water quantity when transferring inks to neighbors. Instead, it will assign an ink quantity that is a little less than it to neighbors. This constraint avoids transferring inks and water back and forth between pixels. When transferring inks, if the difference of ink quantity between one pixel and its neighbors is bigger, the ink diffusion effect is weaker; on the other hand, this pixel can diffuse longer distance from itself. Considering the real painting situation, we set strong ink diffusion for transferring inks from light level to paper level and no ink diffusion from other levels to paper level. Figure 5.6 shows an ink diffusion process converted from a color quantization image. Borderlines between paper level and other levels are irregular and the interior area has grown in an outward direction.

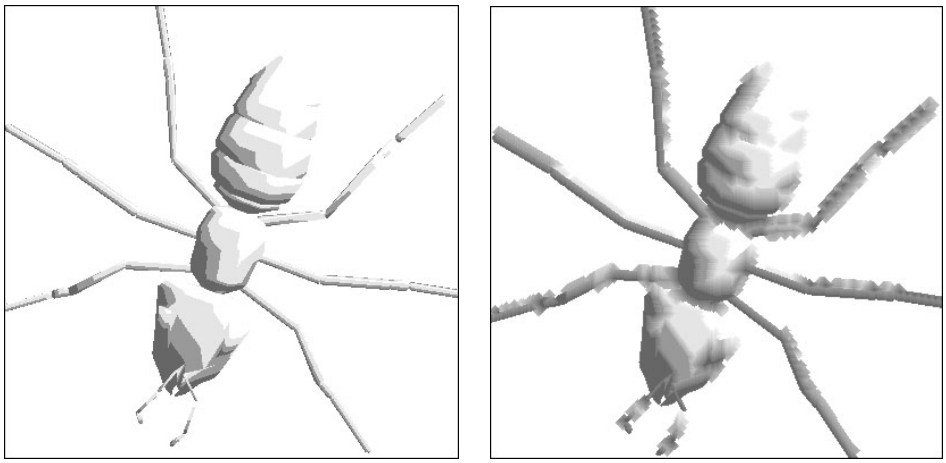


Fig 5.6 Left: A color quantization image. Right: Image after passing ink diffusion stage; the area of each color level has grown in an irregular way.

By observing the ink diffusion result from the right image of figure 5.6, we can still find sharp borders between different color levels. Therefore we apply a 5x5 box filter to each pixel of whole image to blur sharp borders. Figure 5.7 shows the box filtering image from ink diffusion stage.

Fig 5.7 Left: An ink diffusion image, but it still has sharp borders between color levels. Right: We pass the left image to a 5x5 box filter to get a blurred one.

We have tried to apply box filter in a different order, for example to apply it before ink diffusion stage, however, different levels will mix together and weaken the ink diffusion effect; or we apply box filter twice before and after ink diffusion stage separately, but the result image is too blurred and causes an inadequate effect.

Instead of using our interior shading algorithms, Lien [27] developed another interactive Chinese painting system that takes a tablet and a pen as its input devices. The tablet can detect current position and pressure from pen during the painting process. Bringing different pressure on a pen will change different hues of brush on a paper. By using this interactive painting system, a user can easily complete the interior shading process and create an interesting artwork.

5.3 Results

As a result, we combine our interior shading pipeline with 2D brush silhouettes and generate interesting images about Chinese painting of animals. The interior shading of figure 5.9 is painted manually by Lien's interactive Chinese painting system. We can find a difference between the following two figures which are computer-generated and hand-drawn.

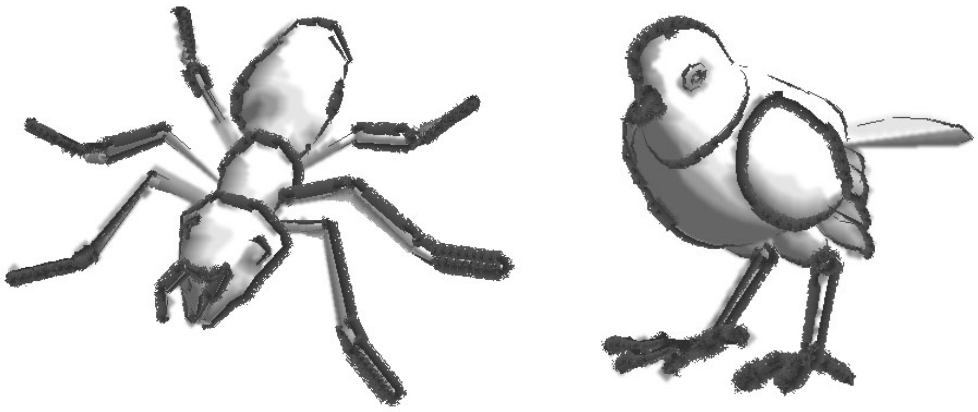


Fig 5.8 Rendering results with brush silhouettes and interior shading.

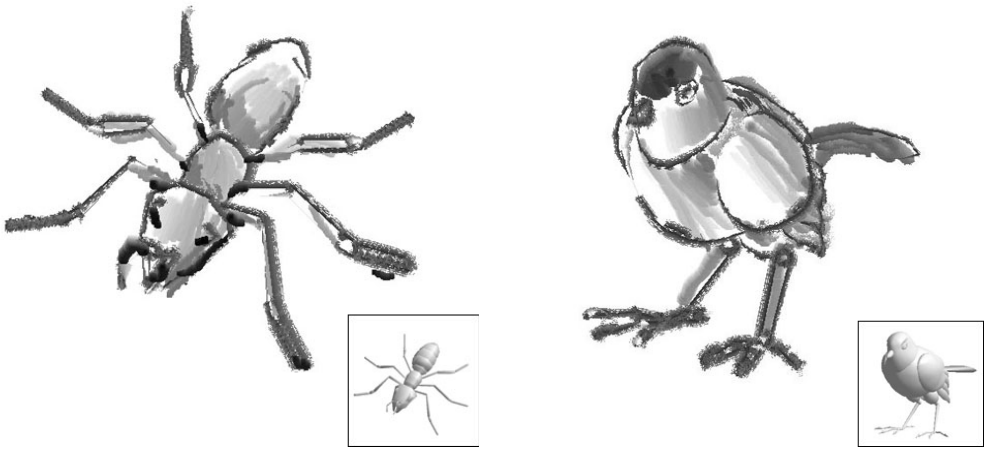


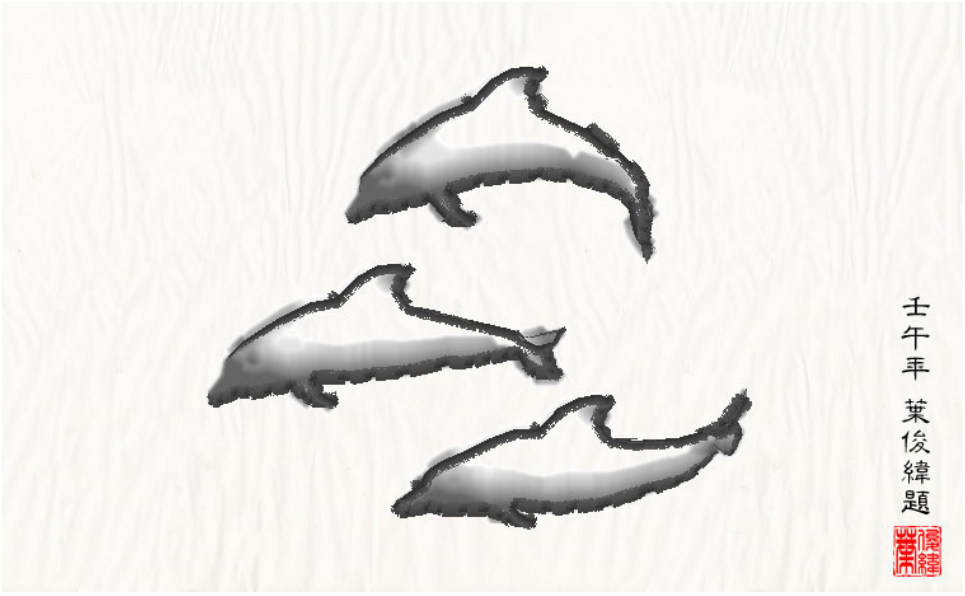
Fig 5.9 Final results after using Lien's interactive painting system to add interior strokes and the original 3D model is at the right bottom corner of image.

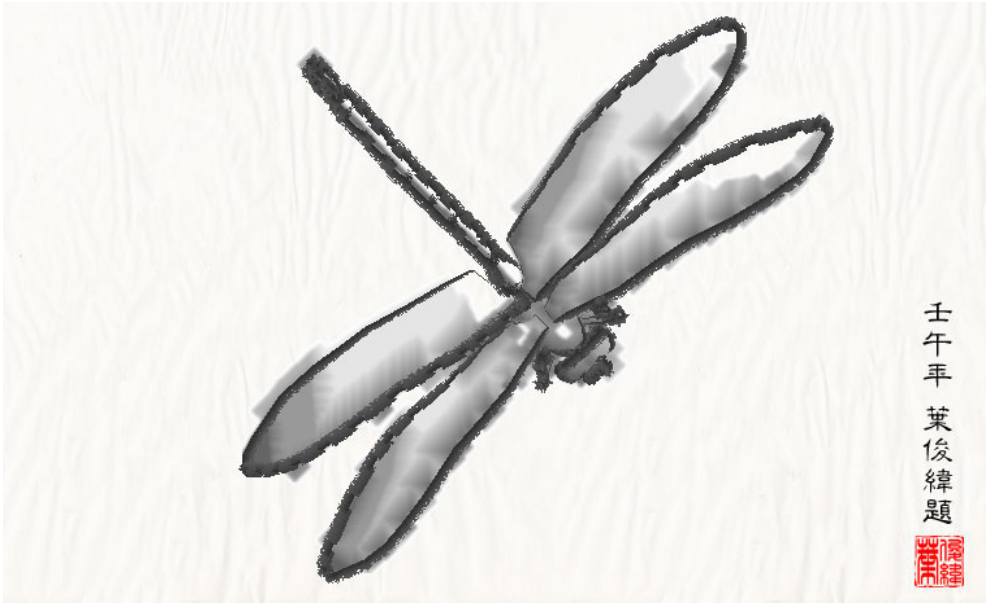
Chapter 6
Results

In this chapter we present result images that take different 3D animal models as input models and the rendering process is fully automatic. Three intensity levels of ink diffusion effect can be chosen to generate different interior shading images: large diffusion, medium diffusion and small diffusion. Besides, a Xuan (宣紙) paper image with my signature is put on the bottom layer to embellish the rendering results.


壬午年 萊俊緯題

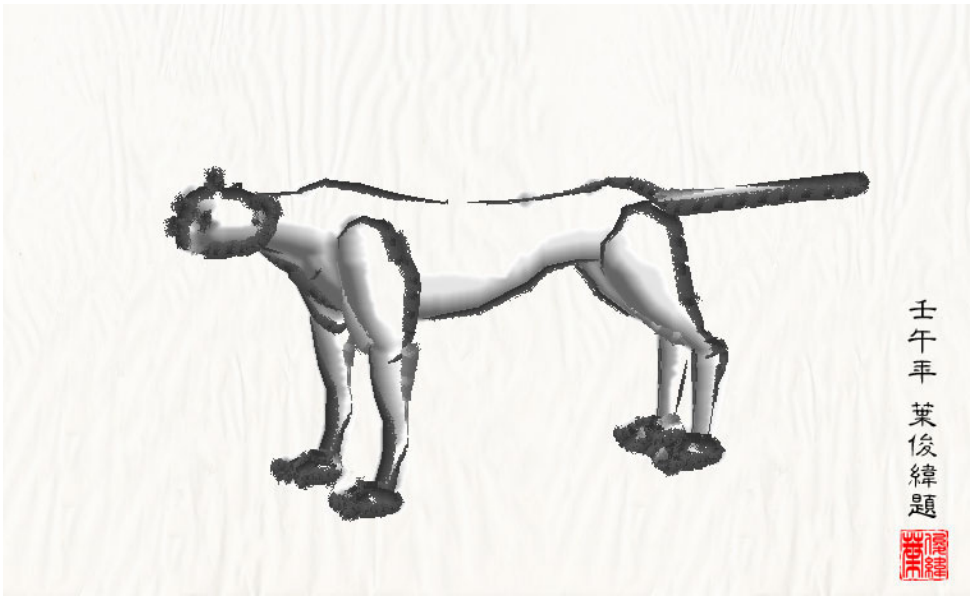

壬午年 萊俊緯題


壬午年 萊俊緯題


壬午年 萊俊緯題


壬午年 萊俊緯題


壬午年 萊俊緯題


壬午年 萊俊緯題


壬午年 萊俊緯題

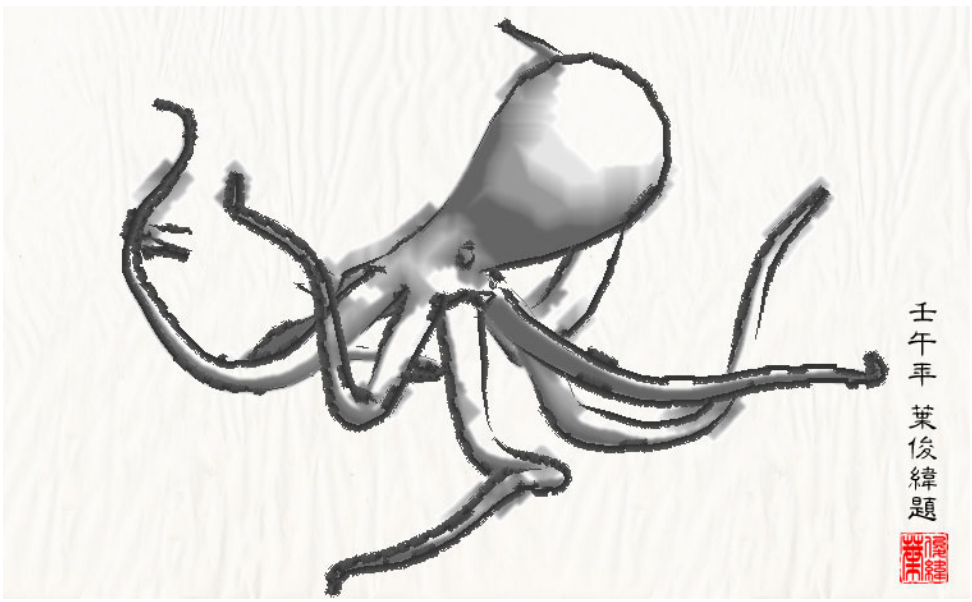

壬午年 萊俊緯題

Fig 6.1 Rendering results generated from different 3D animal models by applying algorithms proposed in this thesis and the rendering process is fully automatic.

Interior shadings of next two color images are painted by Lien's interactive painting system and the brush silhouettes are generated from our system automatically.

Fig 6.2 Images with color interior shading generated by using Lien's interactive painting system.

Chapter 7
Conclusion and Future Works

7.1 Conclusion

A complete set of algorithms to make silhouette strokes automatically from 3D models is proposed in this thesis, and the rendering system is enhanced with the capability of automatic interior shading. 3D model silhouettes can be found by a simplified software implementation and silhouette culling is also done after silhouette finding stage. Silhouette segments are then linked into a long stroke and various stroke widths are assigned at control points of a stroke by our stroke making mechanism. Simulated brush model with special effects used for making Chinese painting strokes and interior shading pipeline are used to render 3D animal models in Chinese painting style. Therefore, a user can automatically transform 3D scenes with animals into a Chinese painting by using our rendering system.

As we mentioned in chapter 5, an artwork is valuable because it can represent an artist's intelligence and creativity during the painting process. On the other hand, our

system can only imitate a real painting process and generate similar results. All 3D scenes will be rendered in an expected way without any surprises. Therefore we encourage a user to paint manually on the result image generated automatically by our system and create unique artworks.

7.2 Future Works

This thesis is only a beginning for non-photorealistic rendering in Chinese painting of animals and there are still many parts to be improved. First, the silhouette finding algorithm can be replaced by new algorithms such as probabilistic testing or new data structures such as edge buffer data structure. These new mechanisms can speed up our system. More special effects on brush model are needed such as the ink decreasing effect, ink soaking effect and back-run effect.

We can also add several brush model patterns in our system to let a user choose his favorite one, and by applying different brush model will greatly change the style of rendering result. The concept of patterns can also be used in the interior shading pipeline. We can develop few algorithms that represent different interior shading methods and integrate with real Chinese painting methods to let users choose one also. Such as retrieving 3D curvature information of a model to find its directionality and draw interior strokes in the interior area. Finally, our system is better extended to have both rendering and painting capabilities so that users can directly use the painting tools provided by us to paint strokes on rendering result. All we need to do is to facilitate users to make a Chinese painting more easily.

Bibliography

[1] “Final Fantasy – The Spirit Within “, presented by Columbia Picture and Square Picture, 2001.

[2] GOOCH, A., GOOCH, B., SHIRLEY, P., AND COHEN, E. “A non-photorealistic lighting model for automatic technical illustration.” Proceedings of SIGGRAPH 98, Computer Graphics Proceedings pp.447- 452.

[3] GOOCH, B., GOOCH, A. “Non-Photorealistic Rendering” A K Peters, Ltd, 2001. pp.2-3.

[4] CURTIS, C. J., ANDERSON, S. E., SEIMS, J. E., FLEISCHER, K. W., AND SALESIN, D.H. “Computer–generated watercolor.” In Proceedings of SIGGRAPH 97, Computer Graphics Proceedings, pp.421-430, 1997.

[5] SALISBURY, M. P., ANDERSON, S. E., BARZEL, R., AND SALESIN, D. H. “Interactive pen-and-ink illustration.” In Proceedings of SIGGRAPH 94, Computer Graphics Proceedings, pp.101-108, 1994.

[6] WINKENBACH, G., AND SALESIN, D. H. “Computer-generated pen- and-ink illustration.” In Proceedings of SIGGRAPH 94, Computer Graphics Proceedings pp.91-100.

[7] WINKENBACH, G., AND SALESIN, D. H. “Rendering Parametric Surfaces in Pen and Ink.” In Proceeding of SIGGRAPH 96, Computer Graphics Proceedings pp.469-476.

[8] SALISBURY, M. P., WONG, M.T., HUGHES, J. F., AND SALESIN, D.H. “Orientable textures for image-based pen-and-ink illustration.” In Proceedings of SIGGRAPH 97, Computer Graphics Proceedings, pp.401-406, 1997.

[9] SALISBURY, M., ANDERSON, C., LISCHINSKI, D., AND SALESIN, D. H. “Scale-dependent reproduction of pen-and-ink illustrations.” In Proceedings of SIGGRAPH 96, Computer Graphics Proceedings, pp.461-468, 1996.

[10] SOUSA, M. C., AND BUCHANAN, J. W. “Observational model of blenders and erasers in computer-generated pencil rendering.” In Proceeding of Graphics Interface 99, pp.157-166, 1999.

[11] SOUSA, M. C., AND BUCHANAN, J. W. “Computer-generated graphite pencil rendering of 3D polygonal models.” Computer Graphics Forum 18(3): pp.195-208, September 1999.

[12] HERTZMANN, A., JACOBS, C. E., OLIVER, N., CURLESS, B., SALESIN, D. H. “Image Analogies” In Proceedings of SIGGRAPH 01, Computer Graphics Proceedings, pp.327-340, 2001.

[13] PRAUN, E., HOPPE, H., WEBB, M., FINKELSTEIN, A. “Real-Time Hatching” In Proceedings of SIGGRAPH 01, Computer Graphics Proceedings, pp.581-586, 2001.

[14] KLEIN, A. W., LI, W., KAZHDAN, M. M., CORREA, W. T., FINKELSTEIN, A., AND FUNKHOUSER, T. A. “Non-photorealistic Virtual Environments” In Proceedings of SIGGRAPH 00, Computer Graphics Proceedings, pp.527-534.

[15] OSTROMOUKHOV, V. “Digital Facial Engraving” In Proceedings of

SIGGRAPH 99, Computer Graphics Proceedings, pp.417-424, 1999.

[16] NORTHRUP, J. D., MARKOSIAN, L. “Artistic Silhouettes: A Hybrid Approach” In Proceedings of NPAR 2000, pp.31-38.

[17] MARKOSIAN, L., KOWALSKI, M, A., TRYCHIN, S. J., BOURDEV, L. D.,GOLDSTEIN, D., HUGHES, J. F. “Real-Time Nonphotorealistic Rendering” In Proceedings of SIGGRAPH 97, Computer Graphics Proceedings, pp.415-420, 1997.

[18] LEE, J. “Simulating Oriental Black-Ink Painting.” IEEE Computer Graphics and Applications, May/June 1999.

[19] WAY, D. L., AND SHIH, Z. C. “The Synthesis of Rock Texture in Chinese Landscape Painting.” COMPUTER GRAPHICS Forum Volume 20, Number 3, pp.C123-C131, 2001.

[20] WAY, D. L., LIN, Y. R. AND SHIH, Z. C. “The Synthesis of Trees in Chinese Landscape Painting Using Silhouette and Texture Strokes.” Journal of WSCG Volume 10, Number 2, pp.499-506, 2002.

[21] PETROVIC, L., FUJITO, B., WILLIAMS, L. AND FINKELSTEIN, A. “Shadows for Cel Animation.” In Proceedings of SIGGRAPH 00, Computer Graphics Proceedings, pp.511-516, 2000.

[22] CORREA, W. T., JENSEN, R. J., THAYER, S. E., AND FINKELSTEIN, A. “Texture mapping for a cel animation.” In Proceeding of SIGGRAPH 98, Computer Graphics Proceedings, pp.435-446, 1998.

[23] RASKAR, R. AND COHEN, M. “Image Precision Silhouette Edges.” 1999 ACM Symposium on Interactive 3D Graphics, pp.135-140, April 1999.

[24] LI, C, M. “The Method of Painting Animals.” ART Book Co., Ltd, 1990.

[25] STRASSMANN, S. “Hairy Brushes.” Computer Graphics (Proc. SIGGRAPH 86) 20(4): 225-232 (August 1986).

[26] Hsu, Chih. Wei. Master thesis “The Synthesis of Rock Textures in Chinese Landscape Painting” pp.47-48. Dept of CIS, National Chiao Tung University,

2000.

[27] Lien, Ting. Yu. Master thesis “A Chinese Painting System with real-time Brush Dynamics Simulation.” Dept of CSIE, National Taiwan University, 2002.

[28] KALNINS, R. D., MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., LEE, J. C., DAVIDSON, P. L., WEBB, M., HUGHES, J. F. AND FINKELSTEIN, A. “WYSIWYG NPR: Drawing Strokes Directly on 3D Models” appears on ACM SIGGRAPH ‘2002.