Non-Photorealistic Rendering in Sketchy Style

by

Chung-Chun Chiang

Submitted to the Department of Computer Science and

Information Engineering

in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Information Engineering

at the

National Chung Cheng University

July 2007

Non-Photorealistic Rendering in Sketchy Style

by

Chung-Chun Chiang

Submitted to the Department of Computer Science and
Information Engineering
on July 20th, 2007, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Information Engineering

Abstract

Sketch is one of the important aesthetic styles in domain of arts. The sketchy drawing is essential to communicate visual ideas and can be used to illustrate drafts and concepts in architecture and product design. In this thesis, we present an effective rendering algorithm for sketchy drawings that include contour drawing and interior shading to simulate the sketchy drawing. In contour drawing, we utilize an image-based approach that extracts and simplifies contour edges using depth and normal information of models. Furthermore, contour maps use perturbed mapping and multiple-drawing to imitate characteristic of hand-drawing. In interior shading, we exploit texture multi-rendering and direction field to depict the interior strokes of model. Our system uses a small number of textures to map smooth results using texture multi-rendering, and variation of curvature can be implemented using the direction field. At last, we show several rendering examples that demonstrate this system can render images in impressively good NPR styles.

素描風格之非真實感繪製

學生：江中竣　指導教授：劉興民

國 立 中 正 大 學 資 訊 工 程 研 究 所

摘要

在藝術領域中，素描是相當重要的一個主題。它利用簡單並清楚的線條來描繪主題，所以被廣泛的使用在概念建築與產品設計等應用上面。因此，我們在本論文中提出了一系列的繪圖演算法，可以藉由輸入的3D模型來模擬出素描風格的圖像。我們把素描風格概略分成輪廓繪製和表面繪製兩大方向來實作：輪廓繪製方面，我們利用3D物件法向量和深度資訊來擷取輪廓線在2D平面中的位置，並同時消去多餘的線條讓成果更加精簡;接著利用雜訊抖動和多重描繪等演算法讓輪廓圖更接近手繪效果。而表面繪製方面，我們使用多重材質貼圖演算法，只需要少量的材質貼圖仍舊可以模擬出生動的光影變化; 並且加上了方向力場的概念,模擬出繪畫中筆刷線條的變化。綜合上面的演算法，我們利用電腦模擬出生動的素描繪圖,並且在最後展示出由我們的系統所繪製素描繪畫和其他風格的成果。

Acknowledgments

I appreciate the many contributions that people have made for this thesis. First of all, I especially want to thank my advisor and mentor, Damon Shing-Min Liu for his helpful guidance and insightful comments on this research. If not for his advice, commitment and suggestion, this thesis would not have been completed.

Many thanks to the officemates who were involved in the survey. Thanks goes to Yu-Hung Hsueh, Chih-Ming Chiu, Chih-Tsang Chen, Chih-Chieh Chang, Ching-Yij Cheng, Hsing-Jen Chen, Chun-Lin Hou, Shao-Shin Hung, Shih-Yu Li, Ching-Sheng Tsai, and Tai-Chun Wei, who kindly offered me their suggestions for conducting the survey and supported me in these three years.

Finally, deeply thanks to my thesis committee members, Prof. Cheng-Chin Chiang, Prof. Wen-Kai Tai, Prof. Chun-Fa Chang, and Prof. Lieu-Hen Chen , who provided many constructive comments in the oral exam for this thesis.

Contents

List of Figures

Chapter 1

Introduction

In this chapter, we present a rough description of this research works. First of all, we narrate the motivation and objective, and then we describe the contributions of our research. After that, we briefly introduce the functionality and the framework of our system. At the end is organization of the thesis.

1.1 Research Motivation and Objectives

Non-photorealistic rendering (NPR) has been a hot topic in computer graphics for last few decades. In contrast with traditional computer graphics, which has focused on photorealism, people utilize software algorithms or hardware accelerations to simulate natural phenomena of real world. But NPR holds an interest on the esthetic sensibility to enable a wide variety of expressive styles for digital art. In order to imitate artistic styles, NPR is inspired by atrworks such as painting, drawing, technical illustration, and animated cartoons. Sketch is one of the popular painting skills, which uses some simple strokes to give a brief accout of outline and shadow. In this research work, we attempt to design an automatic method for generating sketch style which is drawn by artist.

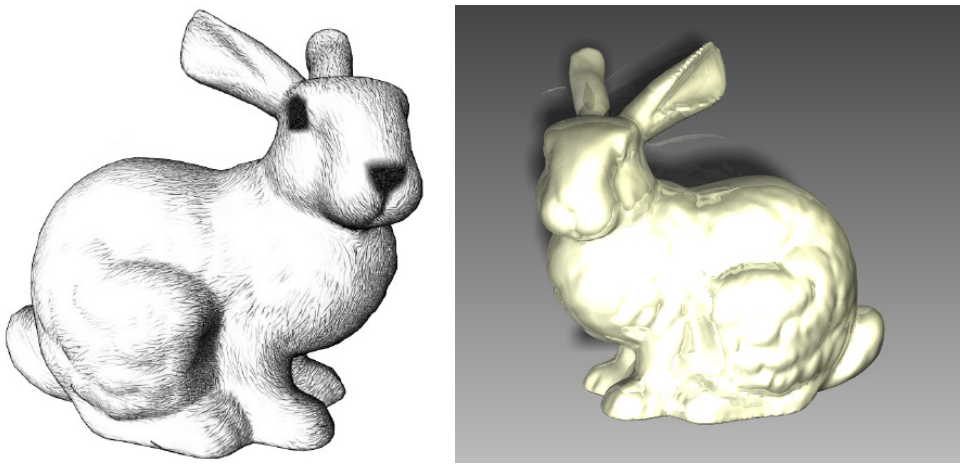


Figure 1-1: The contrast of non-photorealistic rendering and photorealistic rendering.

Generally, artists often provide a sketch by emphasizing lighting, drawing suggest material properties, and outlining reveal shape. Sketch generally refers to groups of strokes with spatially-coherent direction and quality. The local density of strokes controls tone for shading. Their character and aggregate arrangement suggest surface texture. Lastly, their direction on a surface often follows principal curvatures or other natural parametrization, thereby revealing bulk or form. The goal of the system is to develop an approach to simulate these techniques.

However, NPR has an obvious drawback. It costs much time to calculate. Many recent NPR researches focus on how to speedup the NPR algorithm. Despite these good results, rendering of 3D meshes in the NPR drawing style has not been fully addressed in the graphics literature. Although it can generate very natural and realistic effects, the simulation-based approach for NPR drawing has a limitation in the rendering speed. Therefore, we attempt to use a 2D image-based approach which binds stroke textures to replace the simulation ones.

To supply a NPR drawings, many research works have been implemented in

these few years. Pen-and-ink illustration can be considered similar to pencil drawing in that the tones, materials, and shapes of objects are mostly represented by strokes. Excellent techniques were developed to generate pen-and-ink illustrations from 3D objects [47]. An image-based approach was presented to obtain pencil drawings which transforms an input image into the pencil drawing style [25]. Real-time hatching techniques [33] can also be applied to pen-and-ink illustration of 3D meshes.

1.2 Contributions

Sketching is important to express the preliminary state of a draft, concept, and idea, and it describes a drawing that is not a final, perfect result. In this thesis, we propose a NPR sketch rendering technique, which produces directional hatching drawing style images of an input 3D mesh in real time. Specifically, this thesis presents contributions in the following aspects of sketch rendering. Our system implements the sketchy drawing by two techniques, which are contour drawing and interior shading. Contour drawing describes the outline drawing and sketchy skill on contour edges. Interior shading exploits the texture multi-rendering to shade the complex stroke variation on sketchy drawing.

Implementing an effective sketch style rendering system is desirable. The implementation of our sketch drawing is described as follows.

1.2.1 Real-Time Sketchy Contour Drawing

Usually the most important component of an illustration is represented by contour lines. Our contributions are a technique for effective silhouette detection and stylization, and rendering contour lines by sketching style.

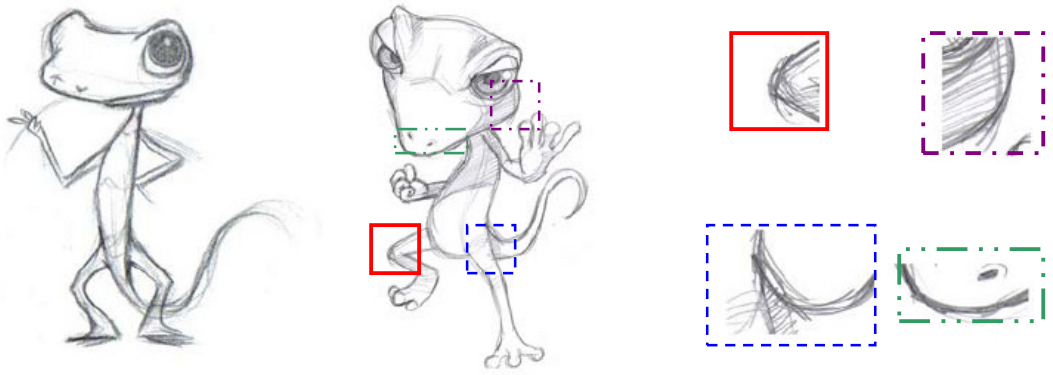


Figure 1-2: Example of real sketches, and zoom-in of details.

A contour drawing includes images of the most visually important curves on the surface: silhouette edges and crease edges. Silhouette edges are the lines that outline the adumbration of object. Crease edges are the curves between two surfaces whose dihedral angle is above a threshold. We implement an image-based contour detection technique to find the contour in seconds. The contour detection task is therefore performed in constant time based on the screen-resolution.

When an artist sketches a scene, the contours are usually drawn a few times with slightly difference. Contour is usually composed of overlapped multiple lines, we can see an example of real sketches in Figure 1-2. Therefore, we attempt to use a simple noise function to perturb the contour to simulate the characteristic of artist technique. Our system exploits different noise functions to modify the same edge map, and the final edge map is composed of these perturbed edges.

1.2.2 Texture Multi-Rendering Using Color-Buffer

Drawing strokes as individual primitives is very expensive. It is ineffective to generate the stroke one by one in computer drawing. In photorealistic rendering, the traditional approach for rendering complex detail is to capture it in the form of a

texture map. We apply this same approach to the rendering of strokes for NPR. We prepare the stroke material where one texture image includes a wisp of strokes.

Our system prepares a small number of stroke textures, and at run-time the surface is rendered by appropriately blending these textures. We implement the shading process using a six-way blending algorithm. This algorithm uses six stroke textures and then blends them in color-buffer to simulate the artist style. The time complexity of our shading algorithm fully depends on the number of triangle of the input model.

1.2.3 Orienting Stroke Drawing

The characteristic of sketch drawing is the alterable stroke brushes. Strokes of hatching are all parallel and straight, but strokes of sketching are varied and curvy. In order to express variation of handwriting, orienting strokes is one of the most important properties for sketch drawing. The computation of the stroke directions depends on the 3D mesh model. The directions are represented as a direction field where for each vertex a value is stored. Our system will rotate stroke textures using direction field to bend the strokes, which imitates the stroke variation on 2D image drawing.

Chapter 2

Related Works

In this chapter we introduce many related woks in different topics. We review several popular styles of NPR techniques in the First Section. The Second Section describes the most recent edge detection methods. At last, the Third Section discusses stroke textures that are utilized in our approach.

2.1 Non-Photorealistic Arts

The number of literatures on NPR drawing approaches increases quickly in these years. Many researches are inspired by artistic styles which are implemented using a variety of NPR algorithms. In this section, we introduce several popular stylistic drawings, such as cartoon style, watercolor style and pen-and-ink style, and we also illustrate the related NPR works of them. We expect to get some inspirations of paining skill from these related algorithms.

2.1.1 Toon and Cel Shading

Cartoon style has often been known as toon shading which is one of the most popular approaches in NPR. The shading is a method of simulating the effect of cel animation (i.e., "2D" or "traditional" animation) using 3D-animation techniques - common

Figure 2-1: Toon shading of human face, Barla, 2006 [1].

characteristics of cel animation, such as contour lines, outlines, and solid-color shading are all replicated (see Figure 2-1). 3D models and animation may be rendered so that the results are more-or-less indistinguishable from images created using traditional cel techniques. This style uses minimalist, high-contrast visual cues to convey 3D shape information. Cartoon styles also are increasingly popular in video games, where interaction makes it appear to be hand-drawn.

极简抽象派

Many previous techniques [10] [12] [26] [28] [51] exist for fast, automated outlines and two-tone shading. And multiple-toon shading had been implemented with good result in recent years [1]. Most of these require a mesh and extend well to amorphous shapes represented by particle systems or grids.

2.1.2 Watercolor Painting

The creation of watercolor illustrations has a long tradition in various arts, fashion, and botanical figures (see Figure 2-2). Artist colorizes a sketchy or detailed drawing and exploits the very complex nature of watercolors in order to achieve a wide variety of artistic effects. Numerous natural effects such as diffusion, pigment granulation,

Figure 2-2: Illustration of watercolor style, Luft and Deussen, 2006 [22].

and color graduation are applied to the color layers of watercolor. Therefore, production of watercolor illustrations is one of the most intricate and difficult problems in the field of NPR.

There are several approaches that explicitly treat methods to generate watercolor paintings. A physically based approach was the work by Curtis et al. [5], which was directly related to the cellular automaton approach by Small [41]. Curtis et al. offered a detailed description of methods to automatically simulate the major effects that occurred during the watercolor painting and drying process. The key idea is to use a simulation model that consists of three physically based layers. While the simulation of these layers produce very convincing results, this method is computationally expensive, and thus could not be considered as real-time graphics. Van Laerhoven et al. [49] provided a real-time simulation of watercolor painting on the basis of a virtual canvas. The work of Lum and Ma [24] that inspired by traditional watercolor paintings described a ray tracing based rendering pipeline that creates procedural textures based on line integral convolution and produces a watercolor like appearance. The combination of several semitransparent color layers forms the final result. Their work allows the rendering of 3D scenes, but does not provide real-time techniques.

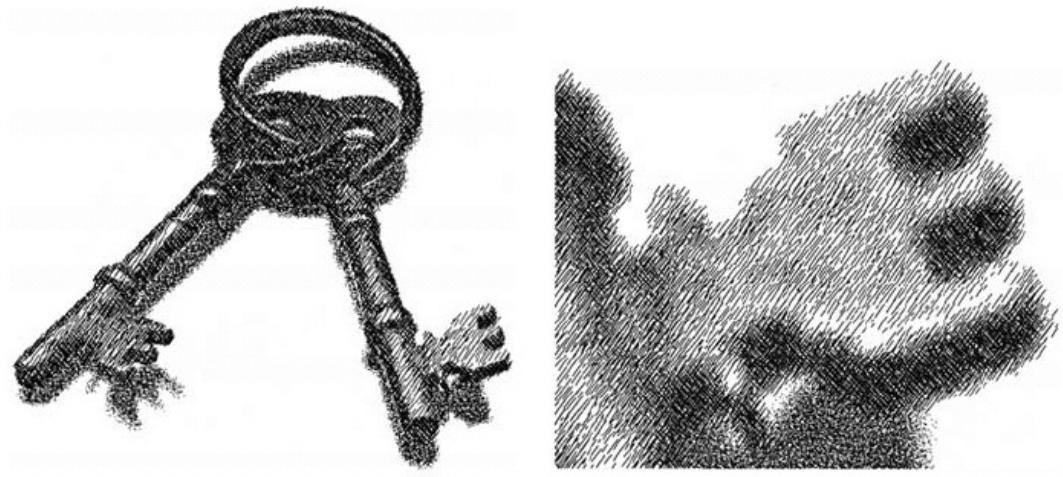


Figure 2-3: Illustrations of pen-and-ink style, *Non-Photorealistic Rendering* [12].

2.1.3 Pen-and-Ink Drawing

Pen-and-ink style refers to a technique of drawing or writing in ink by a pen or other stylus. Pen-and-ink style focuses on the calligraphy of stroke brushes. A number of systems have been built to produce illustrations in a pen-and-ink style. These systems can be classified into two broad categories, depending on their input. One is geometry-based system that takes 3D scene descriptions as input. The other is image-based system that produces its illustration directly from grayscale images. In general, sketchy drawing and pen-and-ink drawing are similar artistic painting skill with distinct stroke materials. We can get some interesting inspirations from these previous works.

Sousa and Buchanan [44] analyzed and simulated the properties of various materials for pencil drawing. They modeled pencils, paper, erasers, and blenders, and their simulation results show nice effects comparable to real pencil drawings. However, this approach needs heavy computation for simulation, which may prohibit

real-time rendering.

Mao et al. [25] proposed an image transformation technique that changes an input image to have the pencil drawing style. They constructed a 2D vector field on an image and applied the line integral convolution to the vector field to generate pencil strokes. Winkenbach and Salesin [48] developed a rendering technique that generates pen-and-ink illustrations from 3D meshes and showed impressive results. Lake et al. [3] presented real-time rendering techniques in various styles, including pencil sketch shading, with projected textures. Saito and Takahashi [38] introduced line drawing on an object surface based on a curvature field for comprehensible rendering of 3D shapes.

2.2 Edge Detection

Edge drawing is one of the important characteristics in NPR algorithm. Artists often draw the outlines to enhance modeling of object in drawing. Therefore, edge detection reduces significantly the amount of data, and filters out information that may be regarded as less relevant, preserving the important structural properties of NPR. To design an effective edge detection algorithm is important for our work. There are many efficient algorithms for silhouette detection in recent years. Generally, edge detection algorithms can be grouped into three major categories: image-based, object-based, and hybrid-based [51].

2.2.1 Image-Based Edge Detection

Image-based algorithms [7] [8] [31] [32] use image-space buffers to calculate the contours. Edges may be viewpoint dependent - these are edges that may change as the viewpoint changes, and typically reflect the geometry of the scene, objects occluding.

Saito and Takahashi [38] introduced the G-buffer as a two-dimensional data structure that stores geometric properties of 3D shapes. G-buffer includes Id-buffer, normal-buffer, and z-buffer. Furthermore, Saito and Takahashi also described image-processing operations used to analyze G-buffer contents to produce "comprehensible images", for example, edge-enhanced or hatched renderings of 3D scene objects.

Nvidia used a screen-aligned quad technique to render edges of 3D shapes in image-space [29]. It samples adjacent texture values to process encoded normals and then uses normal buffer to detect the discontinuities to find silhouette edges. ATI used normal buffer, z-buffer, and ID-buffer to improve Nvidia's original algorithm [30]. Using this way, edges of 3D shapes, regions in shadow, and texture boundaries can be outlined.

2.2.2 Object-Based Edge Detection

Object-based algorithms [2] [3] [16] [32] perform the edge detection computations in object space. Edge may be viewpoint independent - these generally reflect properties of the viewed objects such as surface markings and surface shape. These algorithms have to solve the partial visibility problem in a 3D scene. Only the visible parts of edge lines which are not occluded by the interior of any front-facing polygons have to be rendered. The complexity of object based algorithms is dependent on the size of the input model. If the input model is very large, object-based algorithms are not be feasible.

The candidate edges are identified using the connectivity information, which requires preprocessing of data. Raskar [35] presented a non-obvious but simple to

implement render such features without connectivity information or preprocessing. At the hardware level, based only on the vertices of a given flat polygon, system generates new polygons, with appropriate color, shape and orientation, so that they eventually appear as silhouette edge.

2.2.3 Other Algorithms

Hybrid image/object-based algorithms render objects in certain special way, with the result being the silhouette in image space. Raskar and Cohen [35] introduced an algorithm that computes the intersection of adjacent front-facing and back-facing surfaces in image space. Rossignac and Emmerik [36] presented a two-pass algorithm to render both silhouette edges and feature lines. Rustagi [37] described a stencil-based algorithm which is simple and fast to detect the external silhouette of an object. Gooch et al. [11] discussed a similar method using the stencil buffer. Hybrid algorithms are as efficient as the image-based algorithms. Hybrid algorithms also can be hardware-accelerated. The primary problem with published results to date is, as with image-space approach, that of fitting lines to the resulting silhouette lines for further stylization.

Object-based edge detection algorithms filter the contour edges accurately, but they cost much calculating time on searching edges. In our system, we desire for an effective algorithm to detect edges. Therefore, image-based or hybrid-based algorithms, which can find the contours in constant time, are adaptable to our sketchy drawing.

2.3 NPR Stroke Texture

Strokes in hand-drawn illustrations convey both material and shading. Artist shades

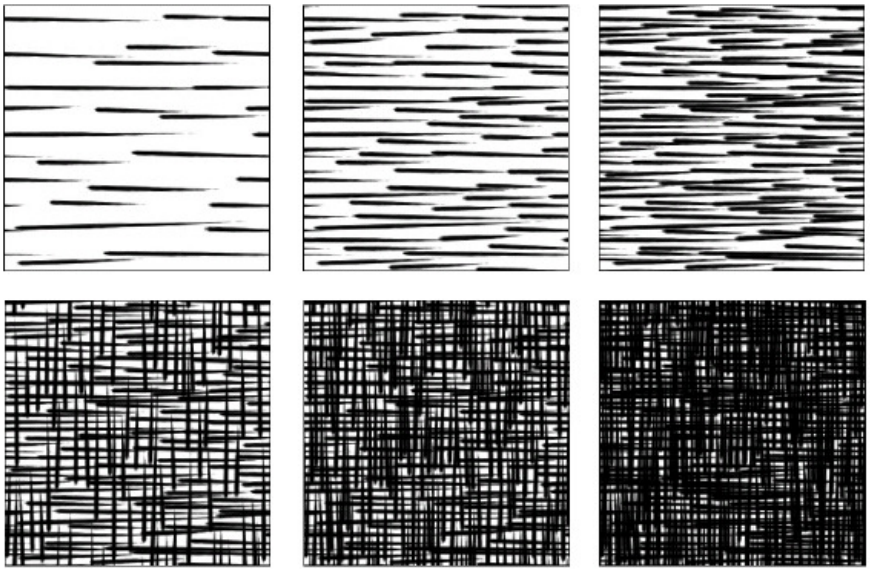


Figure 2-4: Example of tonal art maps, Praun, 2001 [13].

the appearance or shadow regions using different drawing techniques. But drawing strokes as individual primitives is very expensive. In non-photorealistic rendering, the traditional approach for rendering the complex detail is to map the stroke texture images.

In order to draw a sketchy drawing, Praun [20] [33] turned the shading of the model into a selection of a Tonal Art Maps (TAM). A tonal art map is a series of hatched images that depict how an artist would shade an area consisting of 2D grid of images. The tonal art maps ultimately becomes texture maps that the hardware uses to shade the triangles and images. Praun did the linear blending between the 2D textures corresponding to the extreme tone values to be represented. A white texture in TAM represents a small tone value. The grid of the TAM images can help to illustrate temporal coherence in animation. For continuity across textures, Praun generated TAM using a nesting property. Figure 2-4 shows the darker texture is inherited from the lighter one. Darker texture is composed of lighter texture and additional strokes.

This approach could make sure that strokes going from light to dark texture will be seen as spatial continuous.

Webb and Praun [31] then designed an improved algorithm of TAM. Their rendering scheme uses the third dimension information to encode tone value. TAM images are simply stacked up along the tone axis of texture volume. The volume of TAM represents the brightness of texture by the density of strokes. And the high resolution of TAM texture includes more levels of tone value.

Chapter 3

Approaches

We introduce the concepts of sketching rendering technique in this chapter. In a NPR drawing, objects are usually depicted by contours and interior shading. First, we trace silhouette edges and crease edges using 2D image-based method. Our method generates an edge map according to the depth and normal information of object. However, edge map detecting by the approach is too regular and pallid to achieve the hand-drawn effect. Adding error to contours can help to imitate hand drawing. Then, our system exploits stroke textures to simulate the sketching strokes in interior shading. Direction field decides the direction of oriented texture. The texture is applied to these models using the natural parameterization of the shape and rendered using the blending scheme.

3.1 System Overview

In this section, we introduce the organization of our system and the core algorithms we use. Figure 3-1 illustrates the process flow of our system for generating sketchy drawing.

In a drawing, objects are usually depicted by contour and interior shading. In our

edge map:crease edges and silhouette egdes

system for sketchy rendering of a 3D mesh. We first render the input mesh to obtain normal and depth image from the shape. The normal and depth images are used to detect contours in image space. Abrupt changes in normal buffer usually occur at crease edges, and changes in the depth buffer often occur at silhouette edges. We thus were able to create a complete edge map via combining crease edges and silhouette edges. Then we draw the contour several times using different perturbed maps with distinct noises to make the extracted contours look natural, similar to the sketchy drawing by human.

法线buffer突然的变换表明crease edges depth buffer突然的变化表明silhouette edges.

The interior shading is another important topic to be considered. System starts with intensity computation for the input mesh to determine the tone value of rendering vertexes. And we exploit Phone Shading that we can interpolate each pixel of triangle by three vertexes. According to the tone value we calculated, system generates an adaptive texture from the prepared material at run-time. Then, to express sketchy stroke, we use texture rotation and multi-rendering for mapping oriented stroke textures. After contour drawing and interior shading, the resulting images are composed and enhanced to produce the final image.

In the preprocessing step, we prepare several data that will be used in the run-time process. This step only needs to be performed once for each input model. For contour drawing, we prepare some perturbed maps to be used to perturb the contour edges. The distorted map is calculated by Perlin-noise function. For interior shading, we also construct direction field to be used for modifying the orientation of stroke texture. The direction field is computed for the mesh vertexes to determine the orientations in interior texture mapping.

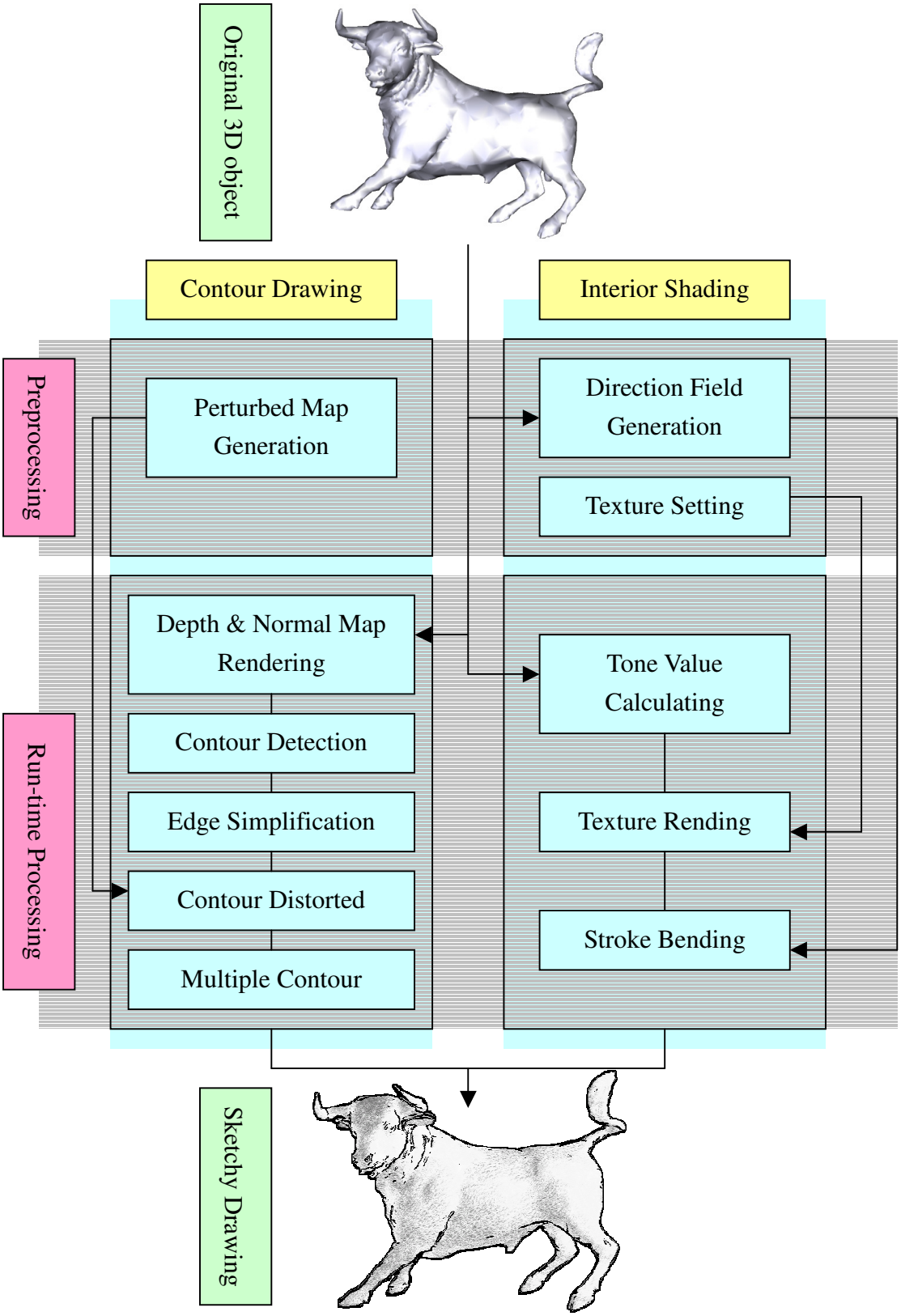


Figure 3-1: System overflow.

3.2 Image-Based Edge Detection

A contour drawing includes only images of the most visually important curves on the surface: silhouette edges and crease edges. We assume that 3D scene objects are represented by polygonal meshes and usually consider edges as of two kinds.

照相机照出来的轮廓线

Silhouette edges: Silhouette edges are the lines that outline the adumbration of object. In 3D space, they represent edges adjacent to a polygon which faces towards the camera, and another polygon which faces backward the camera.

Crease edges: Crease edges are the arc curves between two surfaces whose dihedral angle is above an angle. In 3D space, they represent edges between two front-facing or back-facing polygons whose dihedral angle is above a threshold. The threshold value therefore determines the number of crease edges.

An example illustrates different types of edges in Figure 3-2. For a given 3D scene object, silhouette edges outline the profile of the object, while crease edges outline inner forms of the object. Object-based edge detection can accurately classifies the contour edges. But it is unsuited for real-time algorithm because the computing time grows up with model complexity. Therefore, our system chooses image-based algorithm to implement the NPR rendering. We conducted both silhouette and crease edges detecting using geometric buffers which preserve geometric properties of scene objects in image space. Derived image-space information such as normal and depth is evaluated and combined the different edge images to the final result. The edge detection task was therefore performed in constant time, and the evaluation time depends on resolution of scene.

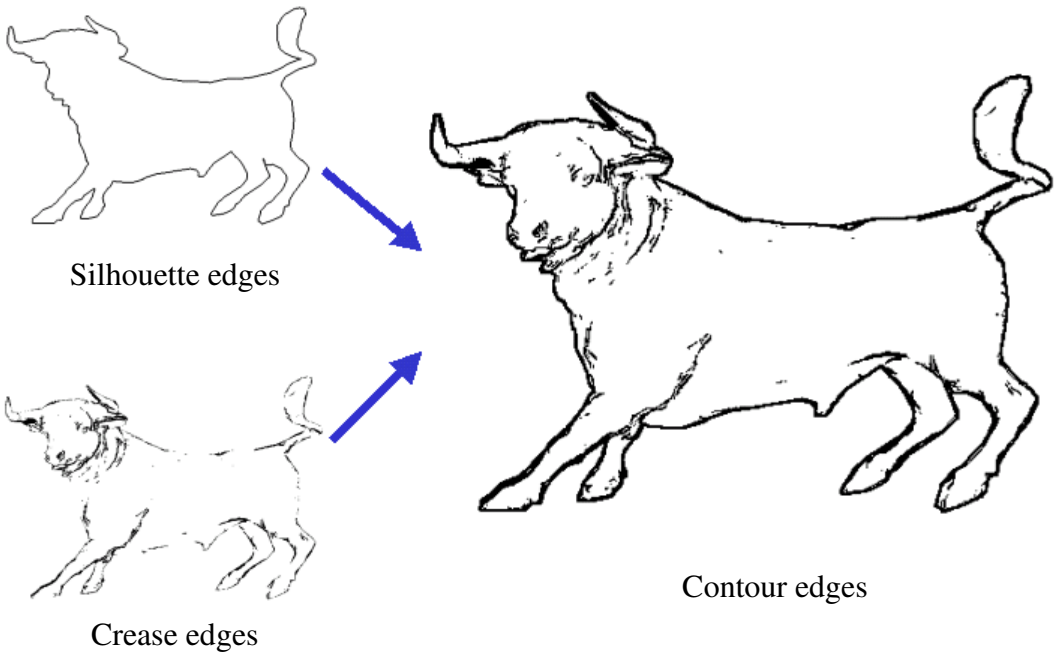


Figure 3-2: Contour edges are composed of silhouette edges and crease edges.

3.2.1 Domain Transfer

Silhouette edges in image plane are the profile of object. In image-space, silhouette edges occur at boundary between two objects or object and background. Crease edges are the curves which profile the variation of surface. They occur at boundary between two surfaces whose dihedral angle is above a threshold in space-domain. Therefore, we can note that edges usually occur at abrupt changes in direction of surface. For object-based algorithms, we analyze the variation of each triangle edge to find the contour edges. Our algorithm transfers the object information to image-space, which exploits color information to represent the space information, to analyze where the edges occurs.

For the silhouette edges, we have to find out the boundary of object and background. Z-buffer records the depth information of the screen, so we can simply

Z-buffer

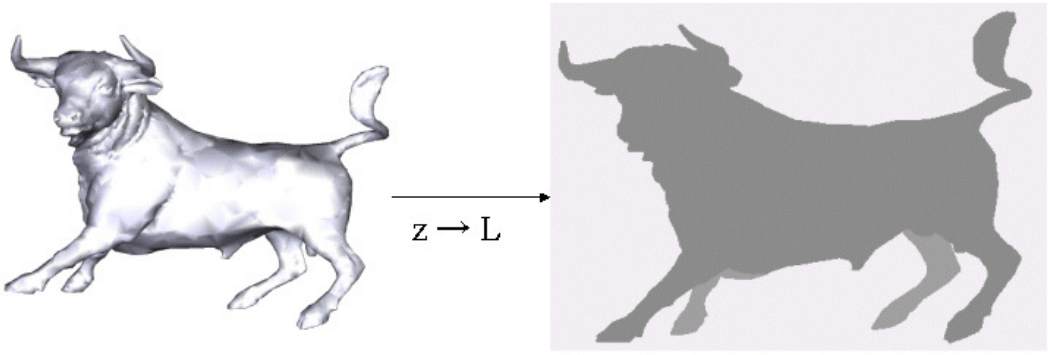


Figure 3-3 Visualize the depth-value information to grayscale color domain.

利用z-values去识别物体和背景

exploit the z-values to classify the objects and background. We captured z-values of the 3D scene into a high precision depth texture which is called a depth map. In OpenGL, for example, the depth map can be extracted by calling OpenGL function "glReadPixels()" with the GL_DEPTH_COMPONENT argument. Different gray values in the depth map represent positions of different depth. The depth value of background is highest, so that we can distinguish silhouette edges using abrupt changes in the depth map. An example is shown at Figure 3-3. We utilize the depth values to draw a grayscale color image, whose grayscale color represents the variation of depth buffer. We set a high threshold value to ensure that it extracts high-quarity silhouette edges.

Depth map

Normal-buffer

Normal vector is a kind of direction information to represents the direction of surface, so we exploit variation of normal to detect crease edges. But it is difficult to compare the variation of direction immediately. Our method uses surface normals to construct a normal map, which is an image that represents the surface normal at each point on an object. Values in each R, G, and B color component of a point on the normal map correspond to the X, Y, and Z surface normal at that point, respectively.

Normal map

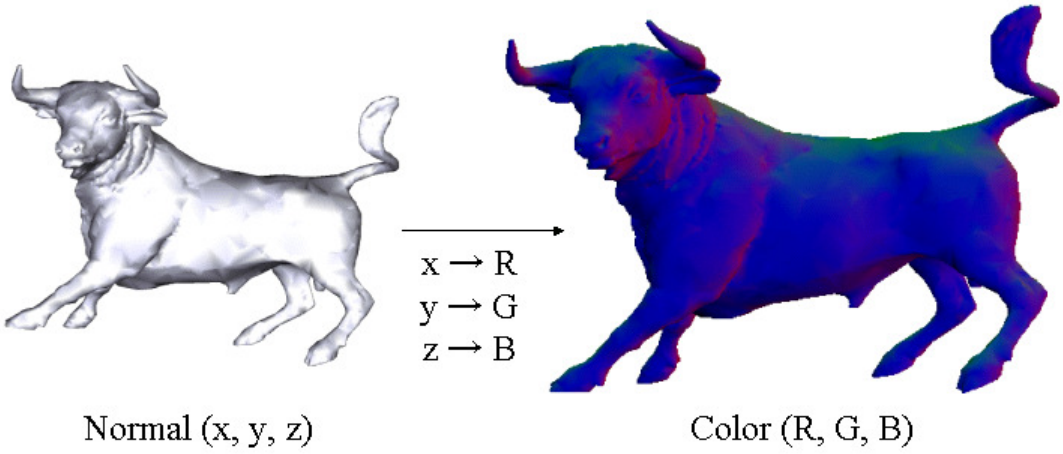


Figure 3-4: Visualize the normal information to RGB color domain.

We render the full object on the screen, and each color of vertex is replaced using the normal information. We replace value of X-axis to red channel, value of Y-axis to green channel, and value of Z-axis to blue channel. In OpenGL, we call OpenGL function "glReadPixels()" with the GL_BGR_EXT argument to record the scene as normal map. As a result, the distribution of color represents the normal vector variation, and we can detect the edge by looking for abrupt variation of color. An example is shown at the Figure 3-4. We utilize the normal vector on each vertex, which replaces the color information of vertex, to draw a full color image. We can exploit this color map to analyze variation of normal buffer.

通过查找颜色的突变来识别edge

After the domain transfer, we could therefore detect edges using the depth map and normal map. In the next section, we exploit a simple filter to obtain a reasonably good contour result from these images.

3.2.2 Contour Filter

Our system translates the information of depth or normal value on space-domain to

image-domain. In the first rendering pass, all 3D scene objects that are declared in the scene are rendered into two 2D textures with depth fragment and normal fragment, which are called depth map and normal map. We analyze data to find the crease edges and silhouette edges, respectively.

To detect discontinuities in image data, linear filter can be used to filter variation of image. For each pixel to be filtered, linear filtering takes into account its neighboring pixels. We need a high-pass filter to find out the parts of high frequency on the normal and depth maps. For NPR sketching, edge detection is only the first step, there are more NPR processing in following step. We have to choose a simple and fast algorithm to detect the variation of maps. Sobel mask calculates the gradient of the image intensity at each point by giving the direction of the largest possible increase from light to dark and the threshold of change in that direction. The result of performing Sobel mask shows how abruptly the image changes at that point, and therefore how likely the image represents an edge. In our experiment, the magnitude of an edge calculation is reliable and fast.

选择一个线性过滤器
Sobel kernels

The Sobel kernels are:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

S_x will detect the vertical variation and filter S_y will find the horizontal variation on the image of depth map and normal map. Vertical and horizontal edge images are both computed using discrete 2D convolution. Summing of S_x and S_y can make sure variation of all directions can be detected [42]. We exploit the Sobel mask on the maps to find out the silhouette and crease edges, and use a threshold T to filter the

unexpected edges which variation is too small to ignore. At last, we combine the two kinds of edge, silhouette edge and crease edge to yield a final contour map.

3.2.3 Edges Simplification

Depth and normal map detect the edges expeditiously, but contour map may include too many unnecessary redundant contour edges. Controlling edge threshold dynamically to conform to each 3D mesh is difficult to implement, we exploit an extra redundant edge filter to remove the shatter line strips. Although we could get a complete contour map, we observed that there are too many line segments existing in it. User becomes more and more difficult to concentrate on contours of the scene due to those small fragments. Regulating the threshold of filter may reduce redundant edges, but it is ineffective to control the variable in each case. Therefore, our system exploits two other filters to decrease the unexpected crease edges.

Crease edges include abrupt variations of normal and depth values. Only these two conditions occur in the same time then we determine this edge is a crease edge. But we only classify normal information in single normal map. Therefore, we exploit the normal and depth map in the same time to remove the redundant edges. We add an extra depth filter to remove these redundant lines after getting *Edge(x, y)*. Objective of this filtering is that we want to remove the edges which are crease edges with depth values equal or close to those of their neighbors. This filtering traces the depth map in a way assuming that a pixel be a component of an edge. The filter calculates difference of the depth value between that pixel and its neighbors. If the difference is negligible, we remove this pixel from the edge map, implying this pixel is no longer a component of the edge. At each pixel Z_0 in depth map, we assume that Z_1 to Z_8 are the

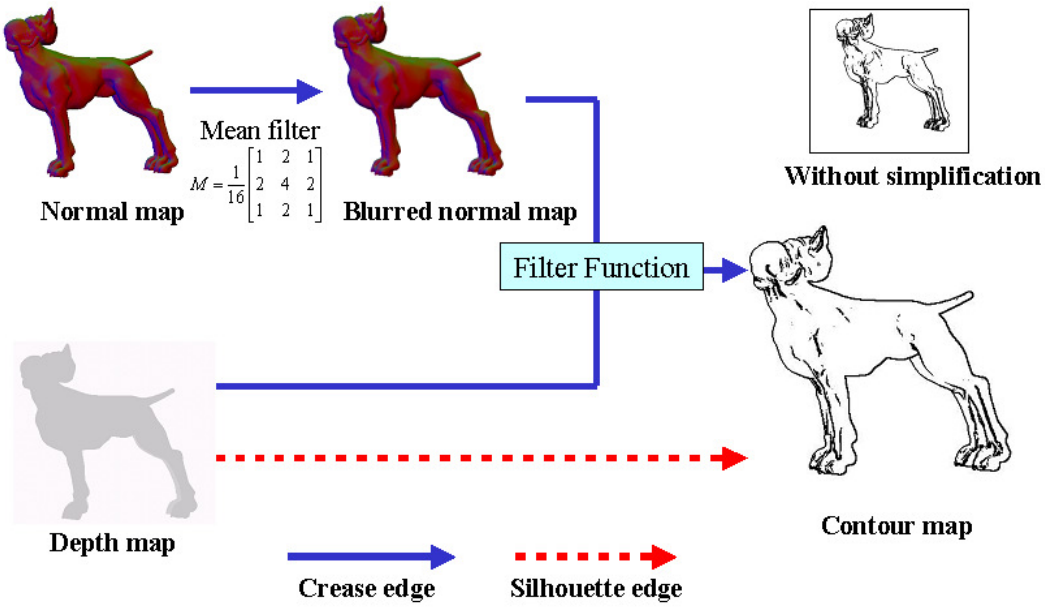


Figure 3-5: Workflow of edge simplification.

neighboring depth values of Z_0.

Z_1	Z_2	Z_3
Z_4	Z_0	Z_5
Z_6	Z_7	Z_8

Entries in filter function are:

$$I_z = (|Z_1 - Z_0| + 2|Z_2 - Z_0| + |Z_3 - Z_0| + 2|Z_4 - Z_0| + 2|Z_5 - Z_0| + |Z_6 - Z_0| + 2|Z_7 - Z_0| + |Z_8 - Z_0|)$$

For each pixel of edge, we will calculate the value of I_z. I_z is the sum of the depth distances between current Z_0 and neighboring pixels. Because the horizontal and vertical neighboring pixels are considered closer to Z_0 than the oblique neighboring pixels are, we therefore double the weight of horizontal and vertical neighboring depth values in our redundant edges removing filter. If the sum I_z is lower than threshold we setup, it implies that this edge point falls on the flat surface. System will ignore this pixel for a crease edge. Most of redundant line segments can be removed

using this filter.

Depth test filters out most of redundant edges, but some tiny noise edges still remain on the map. In contour map, we noticed that most of these noise edges often occur on arc surfaces if normal of face in surface changes abruptly in a small region. In order to reduce the noise edges, we can blur the normal map in advance before filtering edges. Then we use a simple mean filter to smooth the normal map, it can smooth the region of abrupt variation. The smoothing operator is a 2-D convolution operator that is used to blur normal map that can remove detail and noise of crease edges. In this sense it is similar to the mean filter:

$$M = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Filter M will process on each pixel and calculate the mean with neighbors to smooth the normal map. This filter can reduce the abrupt variation of normal between neighbor pixels, so that the noise edges can be cleaned easily. Figure 3-5 shows the workflow of edge simplification in our system.

Our system detects contour edges and removes redundant edges using the same depth and normal map. Moreover, our method can simplify contour edges effectively, and we consider that is good for user to reduce the trivial edges. Therefore, users can focus on the important contours after applying the extra filter.

3.3 Contour Drawing in NPR

Edges in contour map acquired by the method in Section 3.2 may differ from those drawn by hand. There are some disadvantages on edge detection using image-based

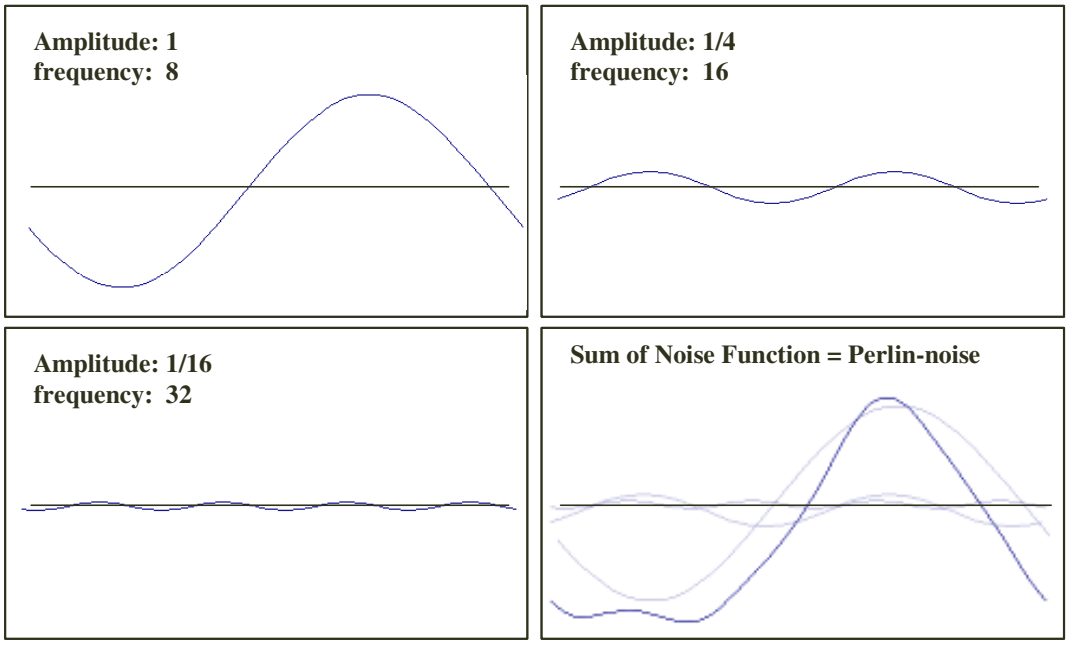


Figure 3-6: Illustration of one-dimensional Perlin-noise. Perlin-noise is composed of the same periodic function with distinct amplitudes and frequencies. In this example, we use the cosine function for the basic function.

approach. In NPR drawing, we can observe that a contour is irregular and usually composed of overlapped lines. We attempt to add artistic effects of human attributes to formal contour map to make the result more vivid. Our system utilizes the Perlin-noise to simulate the habit of human drawing. Perlin-noise distorts the contour using perturbed maps. Then our system merges contour maps with distinct Perlin-noise to imitate the line thickness and wiggliness of NPR drawing.

3.3.1 Contour Perturbation

In human drawing, lines drawn by a human artist are never completely straight and regular. Adding errors to contour map can help to imitate hand drawing. Although a person cannot draw lines without errors, these errors are usually bounded in limited range. When drawing error increases, people will try to fix the shape of target edge to

reduce such error.

Random number is certainly used in error noise, but the randomness may be too harsh to appear natural at times. Therefore, we used a periodic Perlin-noise function serving as the perturbing patterns to simulate human drawing. The Perlin-noise function recreates this by simply adding up functions at a range of different scales. We select a simple periodic function which mixes with Perlin-noise to approximate a perturb function.

With this idea, we use a cosine function to imitate the basic function in Perlin-noise

$$f(x) = a \cdot \cos(b \cdot x + c) \qquad \textbf{(3.1)}$$

where a and b are the range and period of the error, respectively. c is randomness shift of the periodic function.

The definition of Perlin-noise is a function which is sum of the same periodic functions with decreasing amplitude and frequency. We define the one-dimensional noise function:

$$P - Noise(x, amp, freq) = amp \cdot f(\frac{x}{freq}) + amp^2 \cdot f(\frac{x}{freq^2}) + \ldots \qquad \textbf{(3.2)}$$

where *f(x)* is a cosine function computed using Eq. (3.1). amp is a variable of amplitude rate, and freq controls the frequency of periodic function. The progression of amplitude is limited in [0, 1], and progression of frequency is over than 1. We can perturb contours by adding the noise computed using Eq. (3.2).

In Figure 3-6, we can see that this function with distinct variations using different amplitudes and frequencies. The final Perlin-noise function is composed of these periodic functions. Our system will add this noise on the contour map to make it

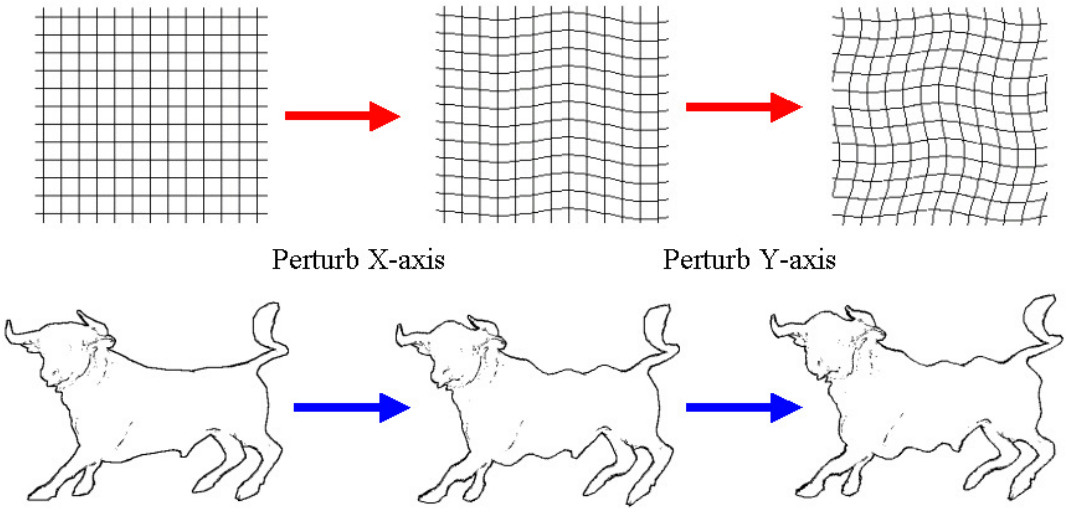


Figure 3-7: Example of perturbed mapping in rectangles and contour maps.

look more natural.

3.3.2 Perturbed Mapping

Our method exploits a one-dimensional Perlin-noise function described in previous section to distort the contour map. But it is difficult to implement the perturbation process directly. We perturb the contour map using the noise function, which shakes contour edges linearly. But these edges are composed of independent color pixels, because contour map is generated using an image-based approach. Another reason is due to the hardware limitation. Fragment shader cannot write a value to another fragment happening to be the current one. We therefore shifts color of pixel to the neighbor one linearly to shake contour map using the noise function. We have to design an indirect approach to wiggle the contour map in single pass. Therefore, our system resolve the problem by rendering the contour map using perturbed map, instead of shaking the edges directly.

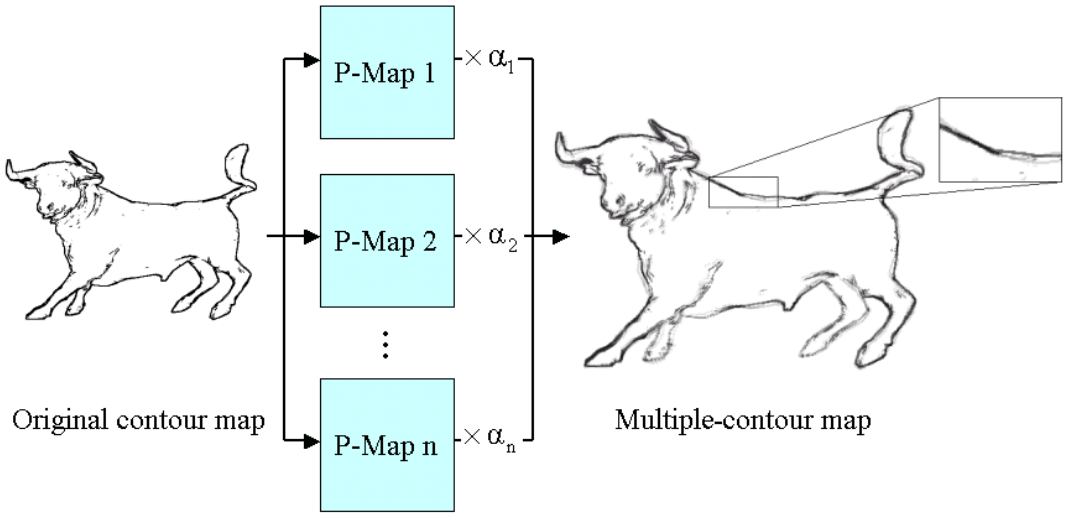


Figure 3-8: Multiple-contour drawing with distinct perturbed maps.

In the preprocessing step, system divides the screen space into regularly sized rectangles. We assign the coordinates on the contour map to the perturbed map coordinates of rectangles. Each coordinate on distorted map corresponds to the rendering coordinate of screen. Noise function wiggles perturbed map in X-axis and Y-axis respectively. Figure 3-7 shows the example of perturbed map using Perlin-noise function. Top right shows the results of perturbed planes with shaken rectangles. And bottom right shows the shaken results of contour maps.

In the run-time rendering process, contours will be redrawn by perturbed map and original contour map. Value on perturbed map records the real coordinate of rendering pixel. Fragment shader receives the color value from assigned coordinate. The redrawn contour map will achieve the hand-drawn effect.

3.3.3 Multiple-Contour Drawing

In sketch drawing, we observe that a line drawn by human is usually composed of

overlapped multiple lines, not a single one. Human may draw a single line several times using similar but different segments repetitively. In our contour drawing, our system composes several contour maps which are distorted by distinct perturbed maps to achieve human drawing effect.

When combining multiple-contour maps in screen, we put edge contributions with much overlapping of contours. This strategy imitates a darkening effect induced by overlapping stroke. Each contour map possesses its own alpha density which determines upon the contribution of color in the final contour map. Higher density imitates a darkening contour stroke on the image. In general, we control the highest density on main contour map with slight perturbed map, and other contour maps where each map has a heavy distortion with minor differences.

Figure 3-8 shows an example of multiple-contour drawing. The original image on the left is the created contour map. System will exploit perturbed maps with distinct Perlin-noises to shake original contour map, and multiply these contour maps based on their contributions of densities. The right picture is the merging result of the left images. From the zoom-in image, we can see that overlapping contours are successfully implemented. Our system can imitate the wiggleness of hand-drawing using the preprocessing perturbed maps and a simple blending function.

3.4 Interior Shading

In sketch drawing, various effects are considered for the interior of an object. Drawing strokes as individual primitives is very expensive, and it is difficult to render each stroke individually. Therefore, we render the complex detail using texture map which includes a group of strokes in one texture. Our system prepares a small number of

stroke textures, and the surface is rendered by appropriately blending these textures at run-time. We implement the shading process using a multi-pass blending algorithm. This algorithm uses six stroke textures and blends them in color-buffer to imitate smooth rendering. Furthermore, in order to express variation of hand-drawing, orienting strokes is one of the most important properties for sketch drawing. The directions of strokes are represented as a direction field where for each vertex a value is stored. Our system will rotate texturesusing direction field to bend the strokes. The time complexity of our shading algorithm fully depends on the number of triangle of the input model.

3.4.1 Stroke Material

Strokes in hand-drawn illustrations convey both material and shading. Artist shades the appearance or shadow regions using different drawing techniques. In order to draw a sketchy drawing, we convert shading of the model into a selection of a stroke texture. A stroke texture is a series of oriented hatched images that depict how an artist would shade an area. Stroke texture can be used to represent a variety of aesthetics (e.g., pencil, crayon, stippling, and charcoal). The material will ultimately become texture maps that the hardware uses to shade our triangles and images. There are many algorithm can generate the material of strokes, such as Tonal Art Maps (TAM). These textures provide the various lightings of stroke materials. Choosing an appropriate texture from the group of textures to shade the object is important.

The images of stroke material capture stroke shading at various tones. We describe the range of tone values, and construct a sequence of hatch images representing these discrete tones. We can render the sketch-like surface using such

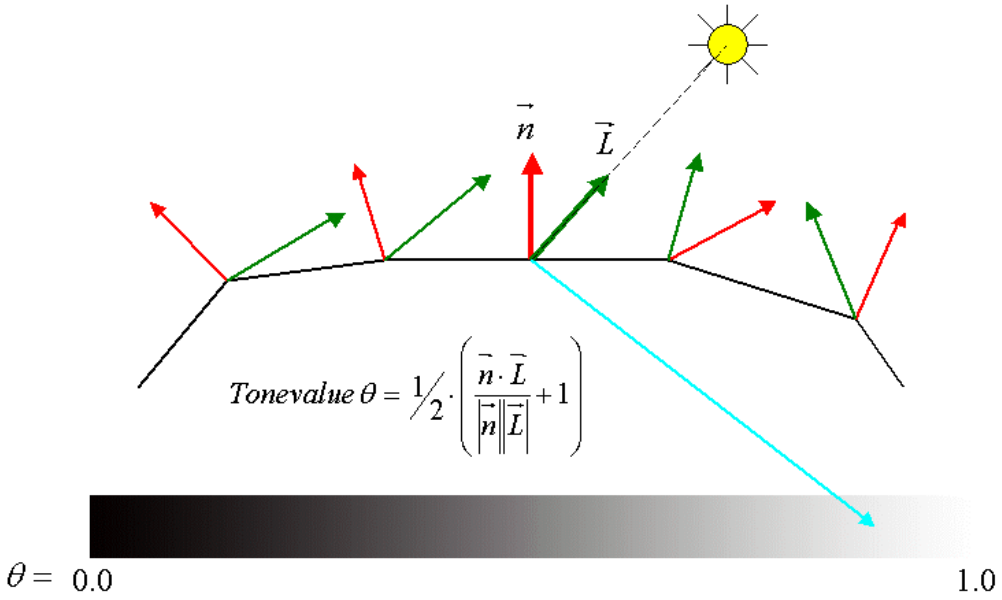


Figure 3-9: Illustration of tone value calculating.

textures. Therefore, we decide a simple tone function to assign an adaptive texture to corresponding triangle. We calculate the tone value of each vertex of triangle and interpolate the value of pixels. Each tone value of vertex uses the angel between vertex normal and light vector at each vertex, and calculates the cosine value to record the tone level of the vertex. The range of tone function is {-1, 1}, and we shift the value into {0, 1} interval as a lighting value. Therefore, the lightest area is located at tone value = 1, and the darkest area is at tone value = 0, which is shown in Figure 3-9. The lighting value represents the lighting level of a vertex. The lighting value is a floating point, and the possible numbers of them are infinite. System will use the tone level to choose the corresponding texture to shade the surface.

3.4.2 Texture Rendering Using Color Buffer

Shadow and lighting are the most essential characteristics in NPR sketchy. Artist

Strokes for interior shading
brightness, thickness

usually draws different density of parallel hatching line to express the material of object. In our system, we exploit stroke textures mapping in the interior surface to replace the complex drawing works. Strokes for interior shading should express essential properties of strokes, such as brightness and thickness, and should be much effective than drawing them one by one. Expressing a smooth lighting effect may need a large number of stroke textures, because the variation of stroke is unlimited. But we only prepare a small number of critical textures in our system, and system blends two texture images dynamically at run-time to blend the suitable stroke materials.

There are some problems in the interior shading when generating the stroke texture at run-time. 3D model is composed of triangle surface, to choose an adaptive texture applied on is important. Based on the tone value we calculate, we choose the most adaptive texture images to represent the tone value of a vertex. Tone value is a floating-point number in {0, 1} and it is difficult to pre-generated enough texture to match any possible value. However, our approach picks a unique texture for each triangle, continuity across textures is a critical problem to improve. Therefore, blending stroke materials to maintain constant tone by adding various strokes to fill the enlarged gaps between few stroke textures. We blend textures across faces by choosing a tonal art map at each vertex to achieve this property. The next issue of discontinuity occurs on the some triangle. Tone value is calculated using the normal of vertex, and these three vertices are all involved with the interior shading of triangle. We linearly interpolate tone value of each vertex and pick the adaptable texture, by blending between the ceiling and floor tonal art maps of vertex. Nevertheless, we exploit Phone Shading to decide contribution of three vertices on each fragment. This

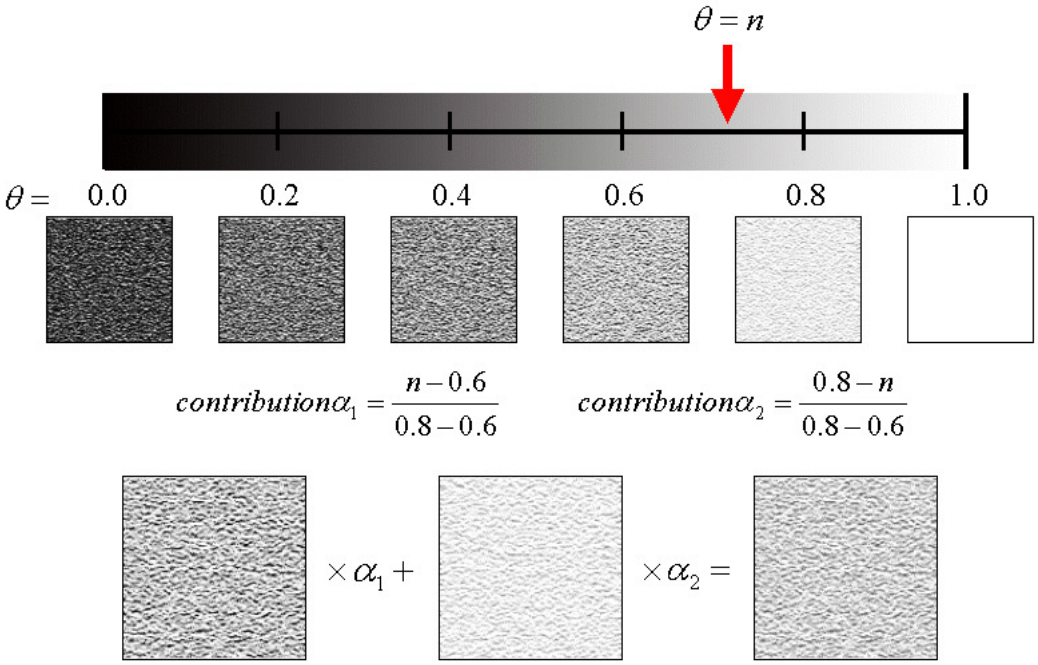


Figure 3-10: Illustration of texture multi-rendering using color buffer.

approach can enhance the continuity of strokes in the same triangle.

To implement the approach we described in previous section, we exploit a color-buffer blending function using OpenGL language. First, we sort the stroke images according to the brightness of texture or density of strokes. System will assign a constant tone value to each stroke texture which is sorted by the brightness. Texture with tone value = 0 represents the texture of brightest region, and tone value = 1 expresses the darkest stroke texture. Tone value is a floating-point number in {0, 1} and must fall on the region of any two destined stroke textures. To render the corresponding tone texture, we linearly interpolate the contribution of tone value between the upper and lower bounds. Figure 3-10 shows an example of texture multi-blending. When our system calculates a tone value, a corresponding tone texture is rendered using the two tonal textures whose contributions are determined by

the weights of linear interpolation. OpenGL's blending function generates the stroke material by blending the corresponding stroke images at run-time. Therefore, we can exploit a small number of textures to implement the interior shading.

After solving the issue of variation of strokes, our system uses a multi-rendering algorithm to smooth the interior shading. In the first rendering pass, we render each triangle by selecting a stroke image based on the lower bound value of the first vertex. In the second rendering pass, we render each triangle by selecting a stroke image based on the upper bound value of the first vertex. Since a triangle has three vertices and each vertex has two extreme values, totally we need to perform six rendering passes. For each triangle, we select six textures for upper and lower bound of vertices and calculate the contribution of these textures. System calls the blending function to merge these textures and shade the final result on the screen.

3.4.3 Direction Field

Stroke rendering implements the NPR stroke hatching on the 3D object. But it is still not good enough to realize an artist-like drawing. In sketchy drawing, variation of curve direction is usually used to describe the modeling of object. Orienting strokes is one of the keys for illustrators to provide the viewers with a sense of shape of the portrayed objects. Our system exploits direction field to enhance the interior shading.

Computing the stroke directions depends on the way in which the images are produced. Before shading surfaces of the model, we first calculated the direction field of the model. Our input model has a three-dimensional information, and the directions are represented as a vector field where for each vertex a value is stored. System calculates the direction information in the preprocessing step. In most cases, strokes

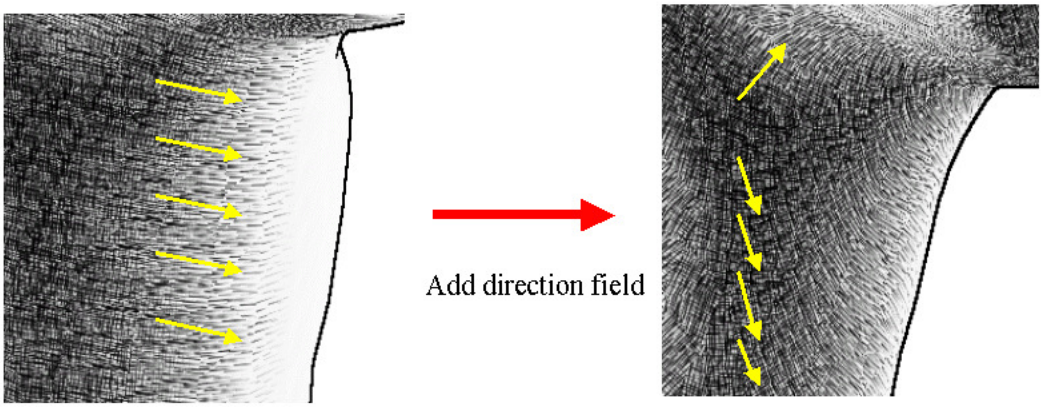


Figure 3-11: Stroke rendering with the aid of direction field.

are aligned to the curvature of the object to be drawn, and these curvature values are relatively easy to derive. For stroke textures that cover a certain area, this direction field is the basis to align the strokes properly. The approach can be used to align a single stroke along the given direction vectors. To start, the algorithm randomly selects a vertex P_i and the next vertex point P_{i+1} from the object model. We assume θi is an included angle between vector $P_{i+1} - P_i$ and the vertical direction of P_i. For each vertex on the model, system will find a direction vector p on tangent plane of vertex whose angle between p and vertical vector is the same with θi. This direction vector p is the value of direction field recorded.

In the run-time rendering, stroke textures are mapped onto the object's interior in the image space. In the mapping, we rotate a stroke texture of triangle using the average direction of three vertices via direction field. The rotation angle of texture is calculated using the direction vector and X-axis. Our system uses the direction field to perform the smooth changes of stroke direction to simulate human drawing. The arrows in Figure 3-1 are the stroke directions which are affected by the direction field. We can compare the stroke rendering using direction field with the one not using it.

Chapter 4

Analysis and Results

In this chapter, we present several experiments of sketchy style approach described in previous chapter. In order to evaluate our system, we compared the results with other methods.

We implemented the proposed sketchy rendering technique with OpenGL and OpenGL Shading Language (GLSL). The sketchy rendering examples in this thesis were obtained on a Windows XP PC using AMD Athlon64 3000+ with 512Mb memory. The graphics card is Nvidia GeForce 7900GT with 256Mb video memory.

Our system is an image-based rendering system. In other words, our sketchy drawing is a view-dependent algorithm. In the test environment, we fix the viewport size in 512*512 resolution and set the initial camera position in world center (0, 0, 0). There are three virtual light sources pre-setup in our system. One is at the front left of the object as key-light, and the other one is at the rear right of the object in opposite direction as backlight. The last one is at the bottom of the object as fill-light. These virtual lights only support the system to calculate the tone values, and they are invisible in visual space. All resulting figures shown in this chapter are performed using the same configuration setup. We exploit these three lights to render the objects.

4.1 Edge Simplification

Our system exploits a filter of image-based approach to detect the contours. But this method may cause too may redundant edges of contours. Therefore, our system uses another filter to simplify the contour map, and this technique can help users to focus on the real important contours.

In this section, we show the results of performing our edge simplification algorithm. Our algorithm has an extra advantage that is we can change a threshold value to control the number of edges to remove. Figure 4-1 shows the resultant images of performing edge simplification. In Figure 4-1, there are two sets of input models, first set is the *Pitbull* model, and the second set is the *Bull* model. Each model set we show the results in different thresholds. We marked the conspicuous areas with quadrangles to highlight the effect of removing redundant edges.

In Figure 4-1 *(a)* and Figure 4-1 *(b)*, we illustrate the original contour map without edge simplification. There are too many redundant edges on the legs or head of *Pitbull* model and neck of *bull* model, so that it is difficult to determine the real contour edges. Figure 4-1 *(c)* and Figure 4-1 *(d)* are the results of simplification using a low threshold. In Figure 4-1 *(e)* and Figure 4-1 *(f)*, we can see the result of edge simplification with a middle threshold. In contrast, both mages are much clear and well-shaped. In Figure 4-1 *(g)* and Figure 4-1 *(h)*, we set a high threshold for the edge simplification. We observed the crease edges are almost removed, because of this misfit threshold.

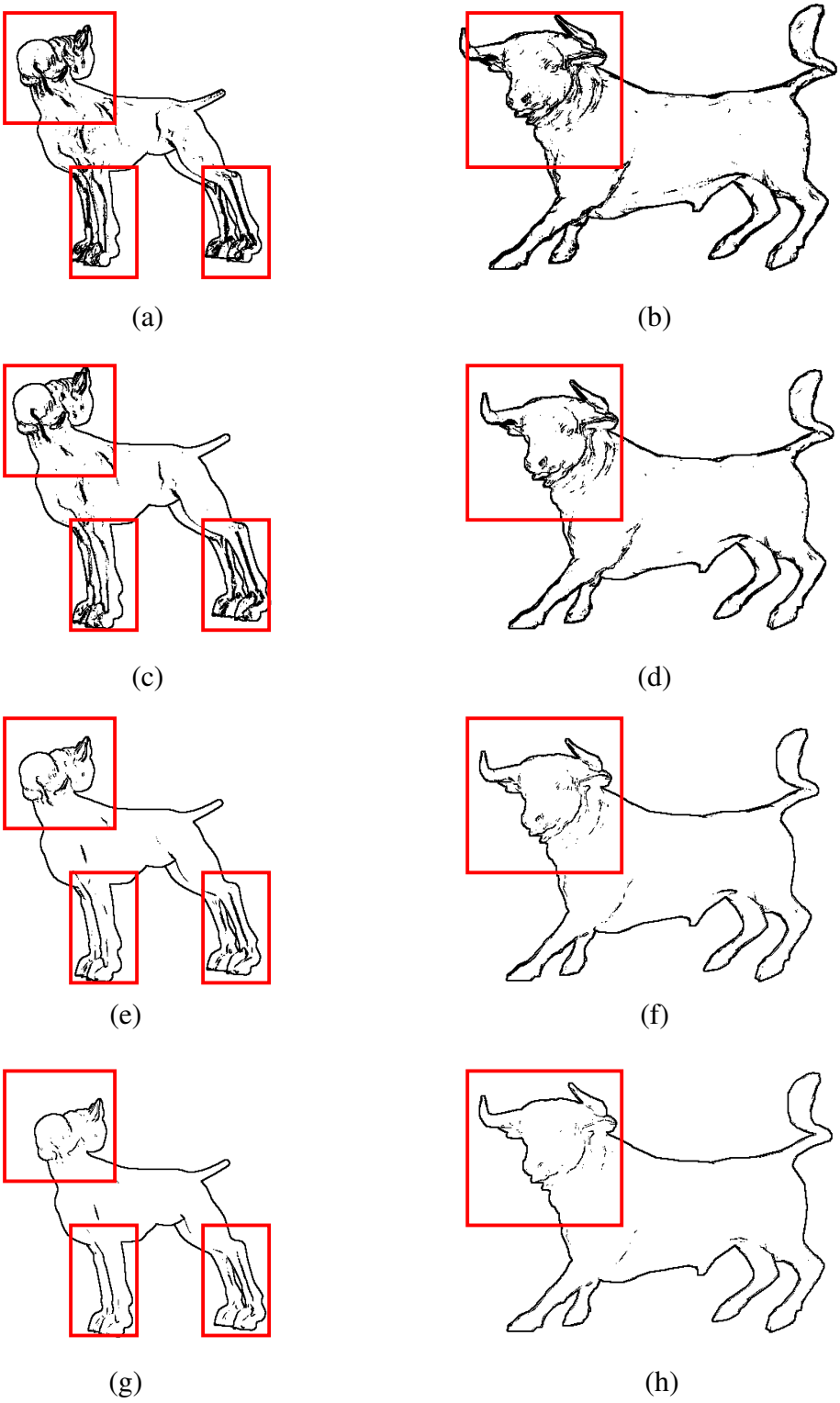


Figure 4-1: Result of edge simplification. (a) and (b) show the original contour without simplification. Set (c)(d), (e)(f), and (g)(h) show the simplification results with low, middle, and high thresholds.

We can see the contour edges have been simplified into distinct levels using different thresholds. More edges can be removed using a high threshold value, and a lower threshold can keep many crease edges on the final contour map. The threshold value of edge simplification is difficult to be determined due to the different characteristics of models. Moreover, each user may prefer different degrees of simplification for the same model. We can only try and test to decide a threshold which can provide the most desired results. In our sketchy style rendering, we suggest setting the threshold between 10 and 15, because it generally generates visually acceptable contour maps.

4.2 NPR Contour Drawing

Edges in contour map acquired by our edge detection method may differ from those drawn by human. In NPR drawing, we attempt to add artistic effects of human attributes to formal contour map to make the result look more natural. The perturbed maps generated using Perlin-noise will distort the original contour map. Then each contour map with different perturbed maps multiplies its own contribution of density. We combine these contours for artist style contour drawing.

In this section, we show the results of performing our multiple-contour algorithm. Figure 4-2 shows the illustrative images of perturbed contour. We show the shaken *Bull* model with distinct perturbed maps in Figure 4-2 *(a)*, *(b)*, and *(c)*. We set the highest density on main contour map of 4-2 *(a)* with slight perturbed map, and other maps 4-2 *(b)* and *(c)* contain heavy distortions with minor differences. In order to approximate the handwriting effect, variation of Perlin-noise has to be limited in one or two pixels. Figure 4-2 *(d)* is the result of combining the previous contour maps. From the zoom-in image, we can see that overlapping contours are successfully

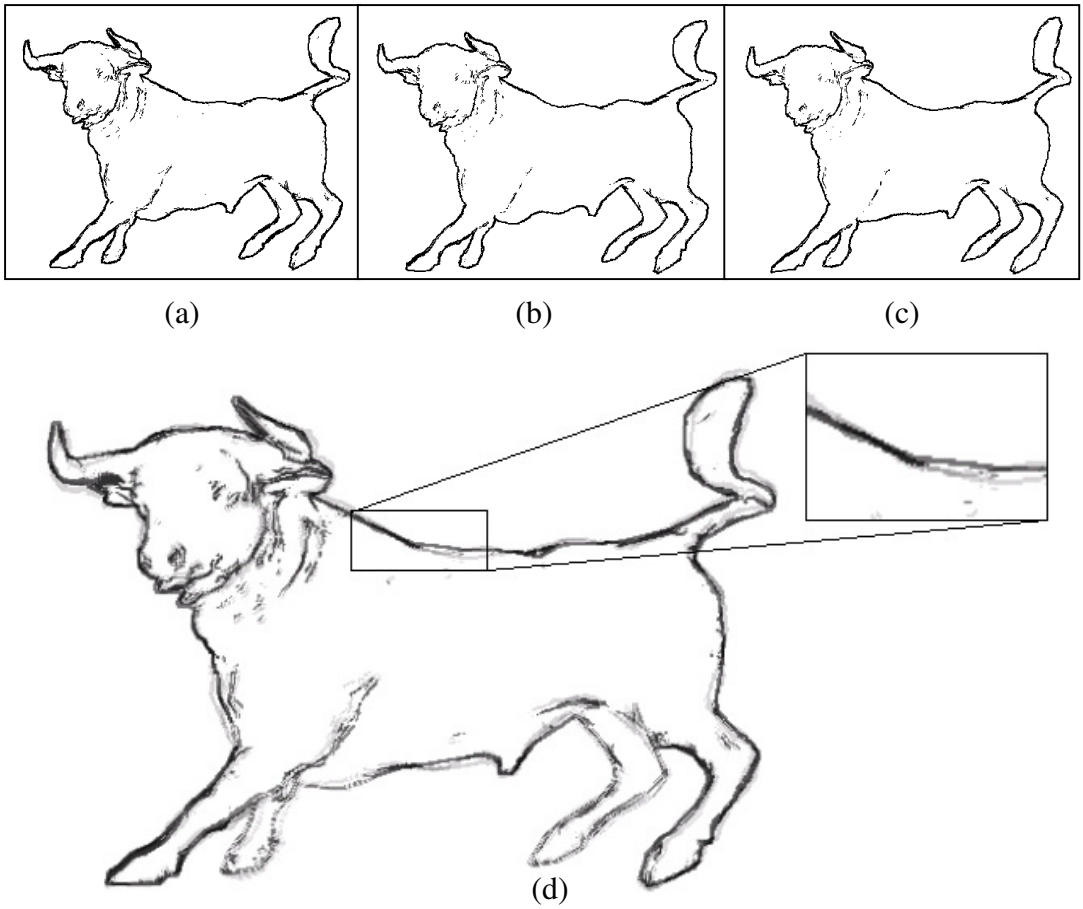


Figure 4-2: Result of multiple-contour drawing. Figure (a), (b), and (c) are shaken contours with distinct perturbed maps. Figure (d) shows the result of multiple contours that is composed of these three contours.

implemented. Figure 4-3 shows another examples of multiple-contour drawing.

We also compare our perturbed result with others similar works. Figure 4-4(left) is the results from J.P. Lewis et al. [21]. In 2005, their published paper describes that the system takes from a few seconds to a few tens of seconds for the drawing of Figure 4-4 with several thousand silhouette edges. In our system, it takes about 0.048 seconds to render the *Coffee-Cup* (see Figure 4-4(right)) with 1696 polygons. Due to the advantage of the image-based approach, the numbers of polygons would not affect the calculating time. The NPR contour drawing is performed in constant time as long as screen resolution in the same.

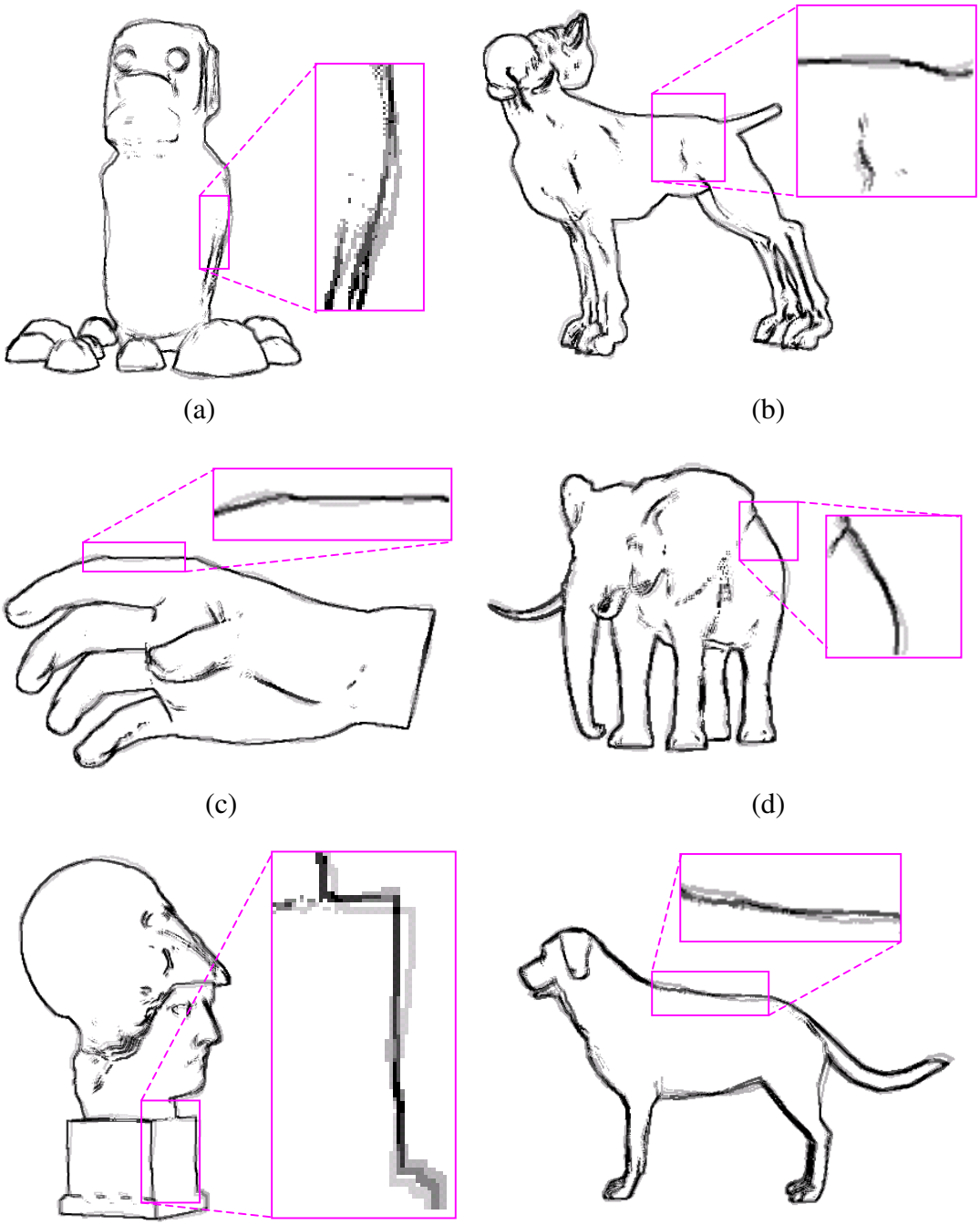


Figure 4-3: Demonstration of multiple-contour drawing.

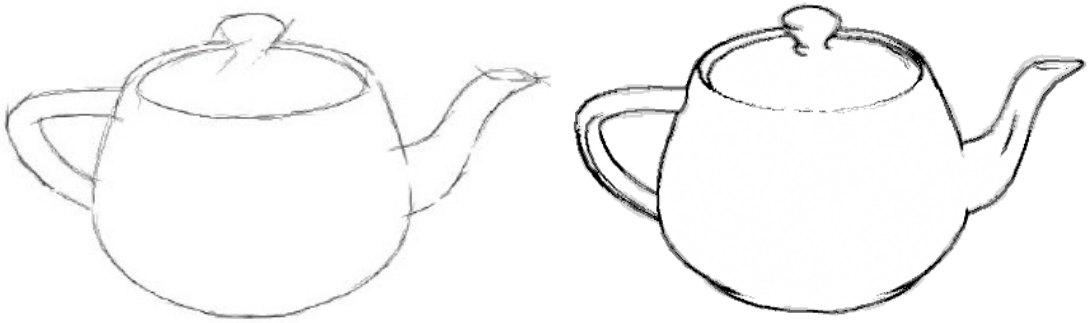


Figure 4-4: Figure in left shows final result from J.P. Lewis et al. [21]; Figure in right is our multiple contour drawing.

4.3 Interior Texture

We exploit stroke textures to render the interior surface to save otherwise the complex drawing works. Using strokes for interior shading reveals essential properties of strokes, such as brightness and thickness, and is much effective than drawing them one by one. In our system, we only need to prepare a small numbers of stroke materials, and system will render the textures using color-buffer to shade the surface. We also exploit direction field to implement the oriented variation of strokes. In this section, we will demonstrate the result using color-buffer texture blending and direction field.

Figure 4-5 shows examples of texture blending in color-buffer. We use only seven grayscale textures to imitate the shading effect in photorealistic rendering. We calculate the tone value of each vertex of triangle and blend the corresponding textures to shade the surface. Although we only have a few source of textures, the shading effect is smooth and colorful (see Figure 4-5 *(a)* and *(b)*). In Figure 4-5 *(c)* and *(d)*, we show another example of texture blending in the case texture is irregular.

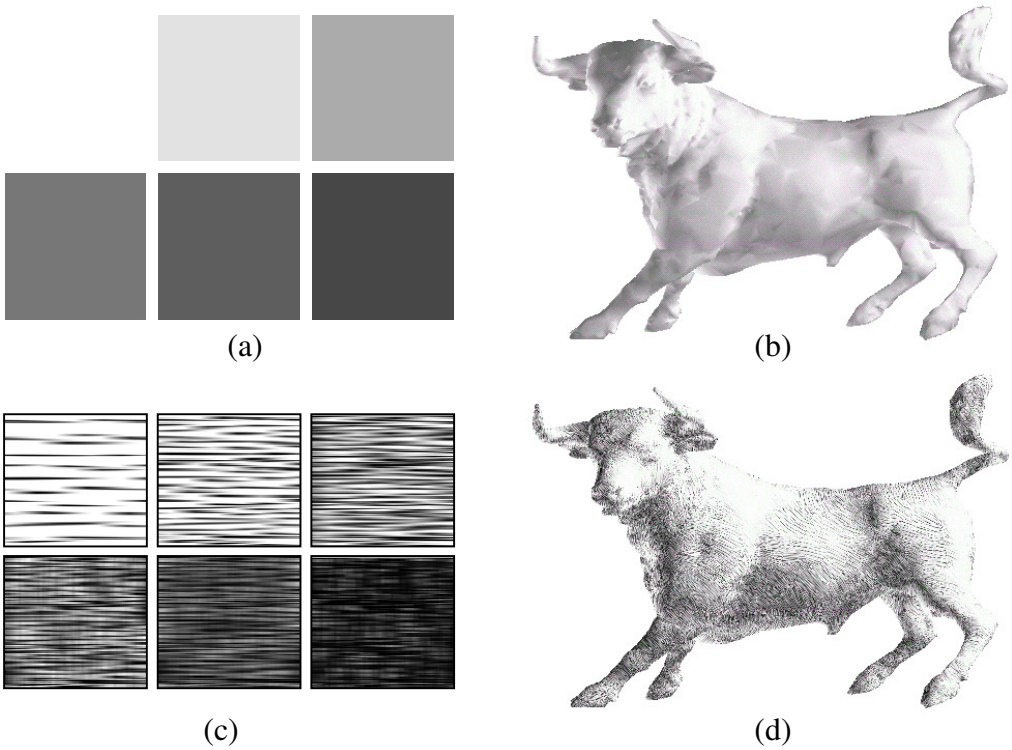


Figure 4-5: Illustration of texture multi-rendering in color-buffer with different texture sets.

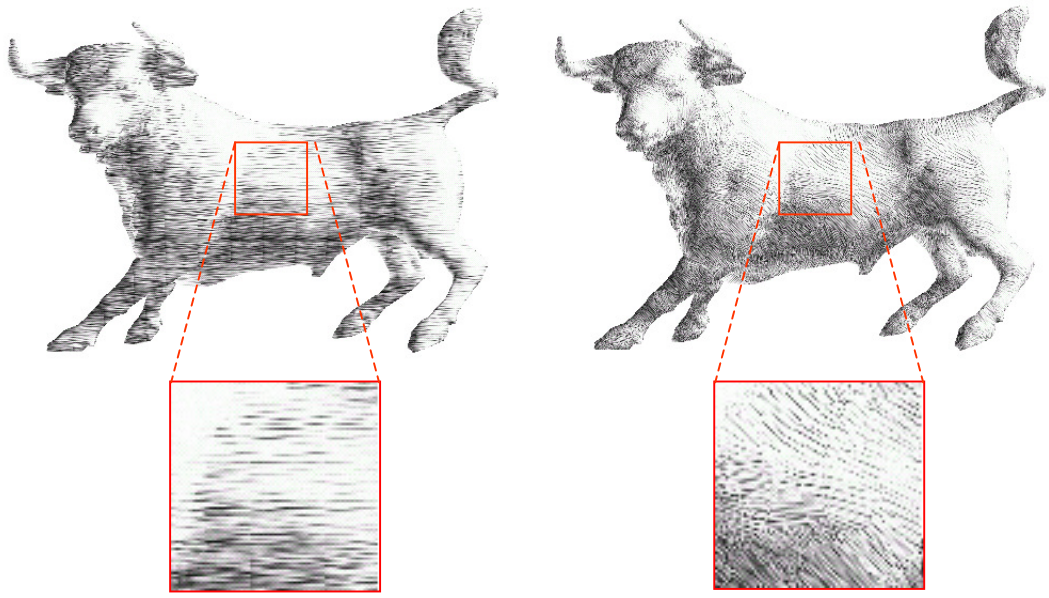


Figure 4-6: Result using direction field. Figure in left shows the stroke shading without direction field, and figure in right shows the effect of oriented strokes.

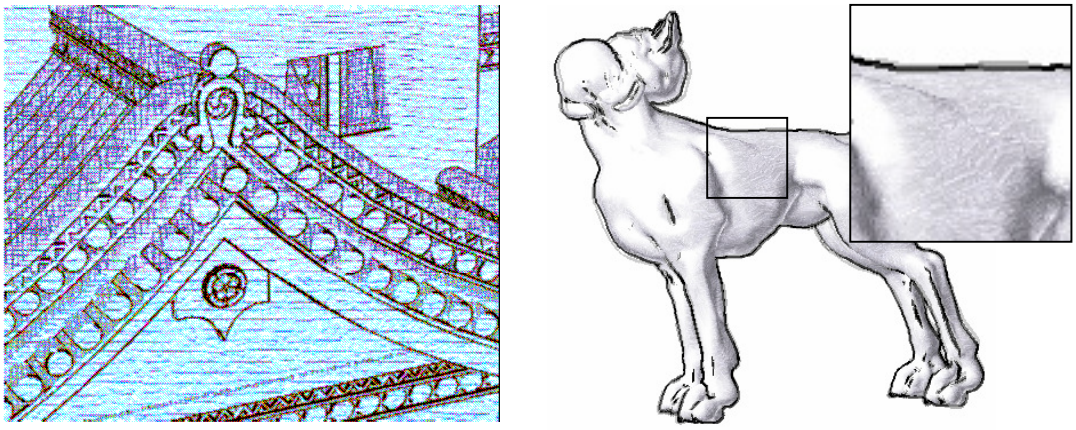


Figure 4-7: Figure in left shows final result from Adam Lake et al. [18]; Figure in right is our sketchy drawing using direction field.

Oriented strokes are usually used to describe the modeling of object. Our system utilizes direction field to enhance the variations of strokes. Figure 4-6 shows the resultant images using direction field. Figure 4-6(left) is result without using oriented stroke, that is, all of the strokes are parallel and inflexible. The result of Figure 4-6(right) shows the result using direction field, where the orientations of strokes can express the variation of modeling.

We also compared the texture blending result of our works with other similar works. Figure 4-7 is the results from Adam Lake et al. [18]. Their system combines pencil strokes and paper texture to implement pencil-sketch rendering. Their pencil strokes are fixed and without variation of orientation. However, we exploit stroke materials and direction field in our system, so that we can generate more natural sketchy style drawing.

4.4 Experimental Results

In this section, we show several sets of result to present different styles. We combine the NPR contour drawing and multi-rendering of interior shading to the final artistic drawing. Our system provides several sets of texture to simulate the different drawing styles.

First, we show the *"Sketchy"* style results in Figure 4-8. We use these sets of texture to simulate the artistic sketch drawings. Second, we show the *"Cross Sketchy"* style results in Figure 4-9. We use these sets of texture to simulate the cross-sketchy drawing technique. Figure 4-10 shows the *"Parallel Strokes"* style results, and we use the style to simulate the hatching drawing. At last, we demonstrate the *"Fog"* style, *"Stone"* style and *"Fur"* style in Figure 4-11. All textures are pre-generated, and the choice of texture sets is determined by users.

We utilize a few pencil stroke materials to imitate sketchy drawing. In Figure 4-8, 4-9, and 4-10, we can see strokes naturally follow the principal curvature directions and the image intensity smoothly changes with the variation of brightness on the surface. We can see the detail of texture blending in the zoom-in images. Figure 4-11 is the result of texture blending without using oriented stroke. These texture sets do not contain the property of direction, but our method still work well in these general cases.

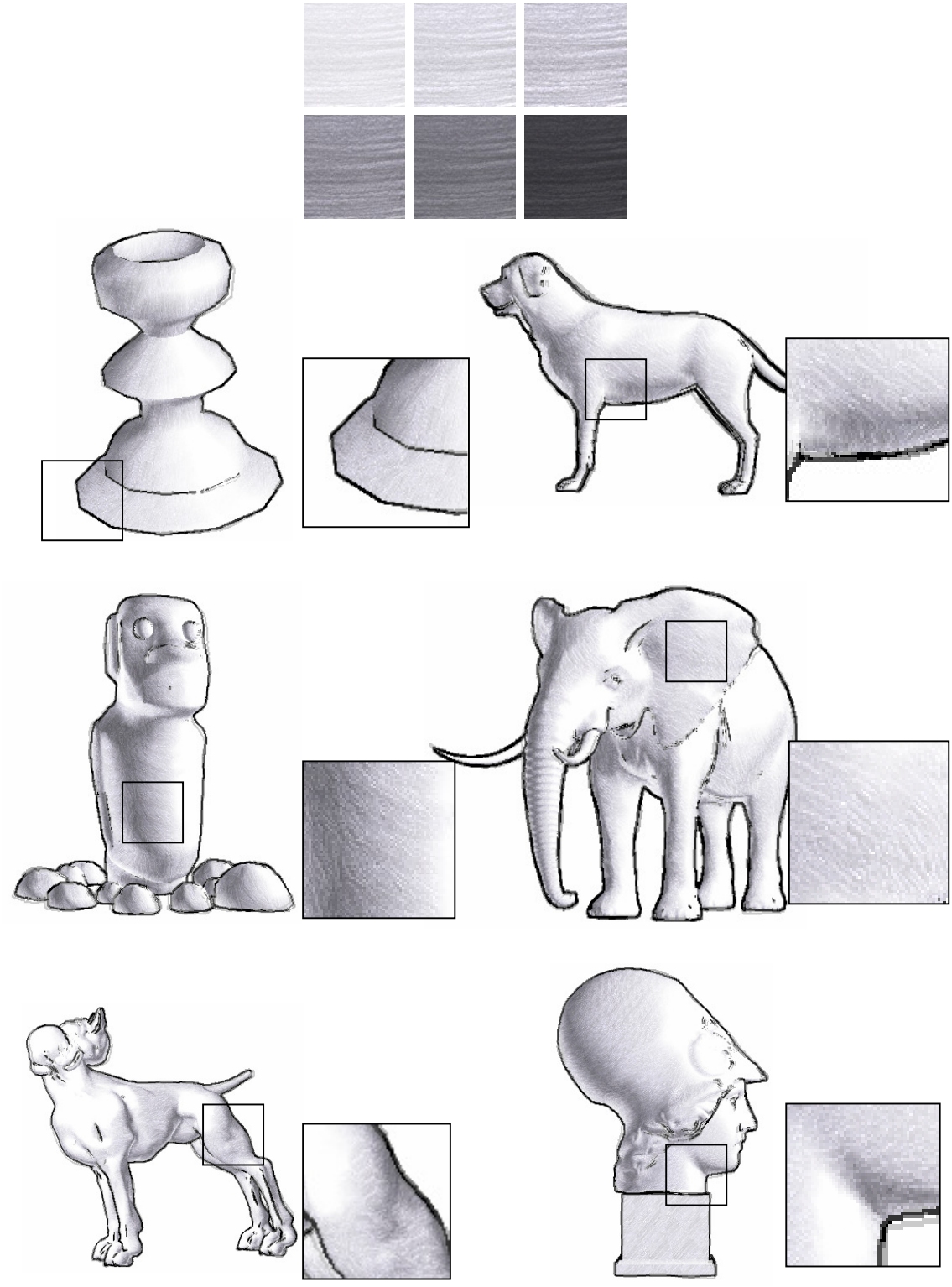


Figure 4-8: Results using *"Sketchy"* style.

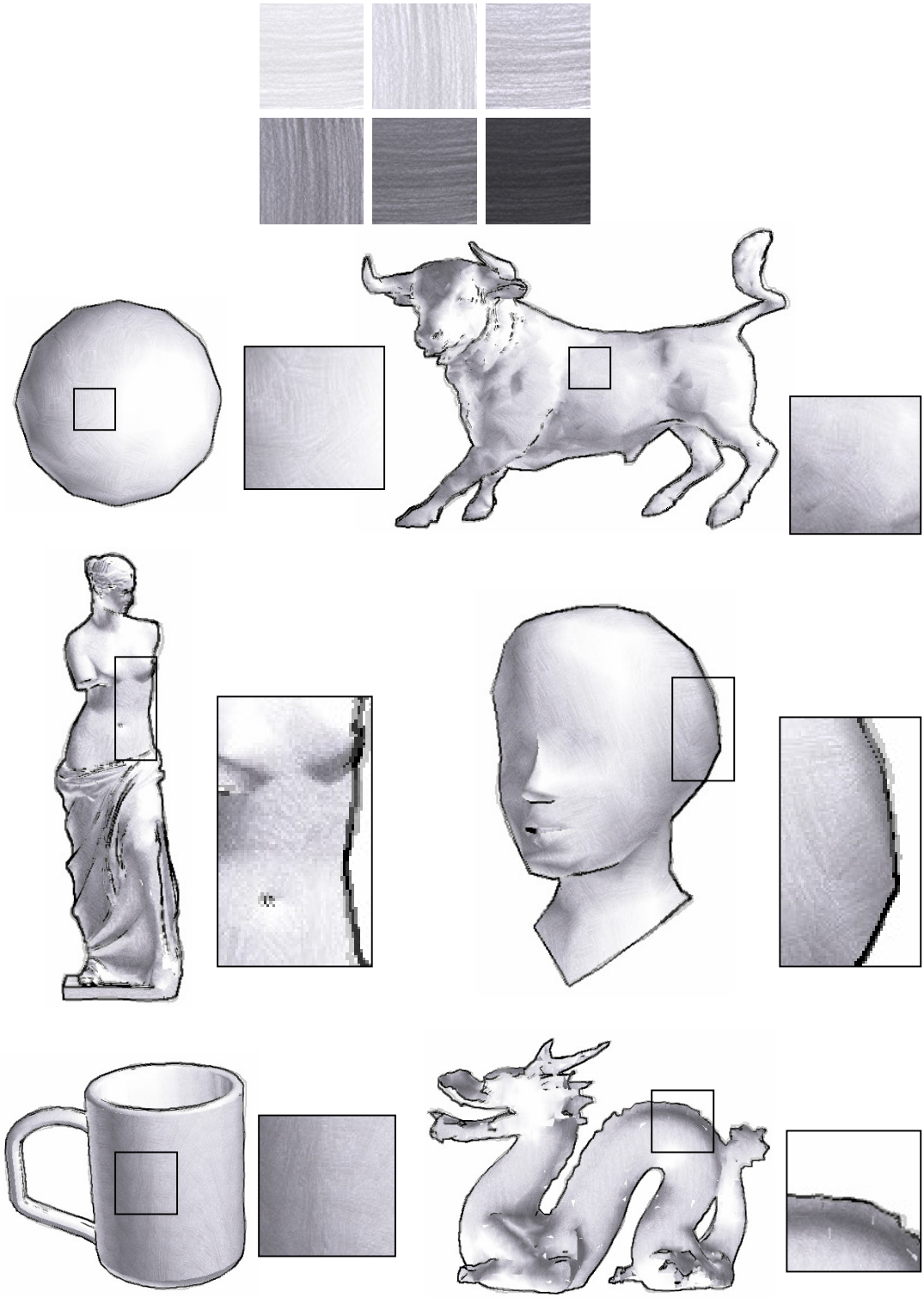


Figure 4-9: Results using *"Cross Sketchy"* style.

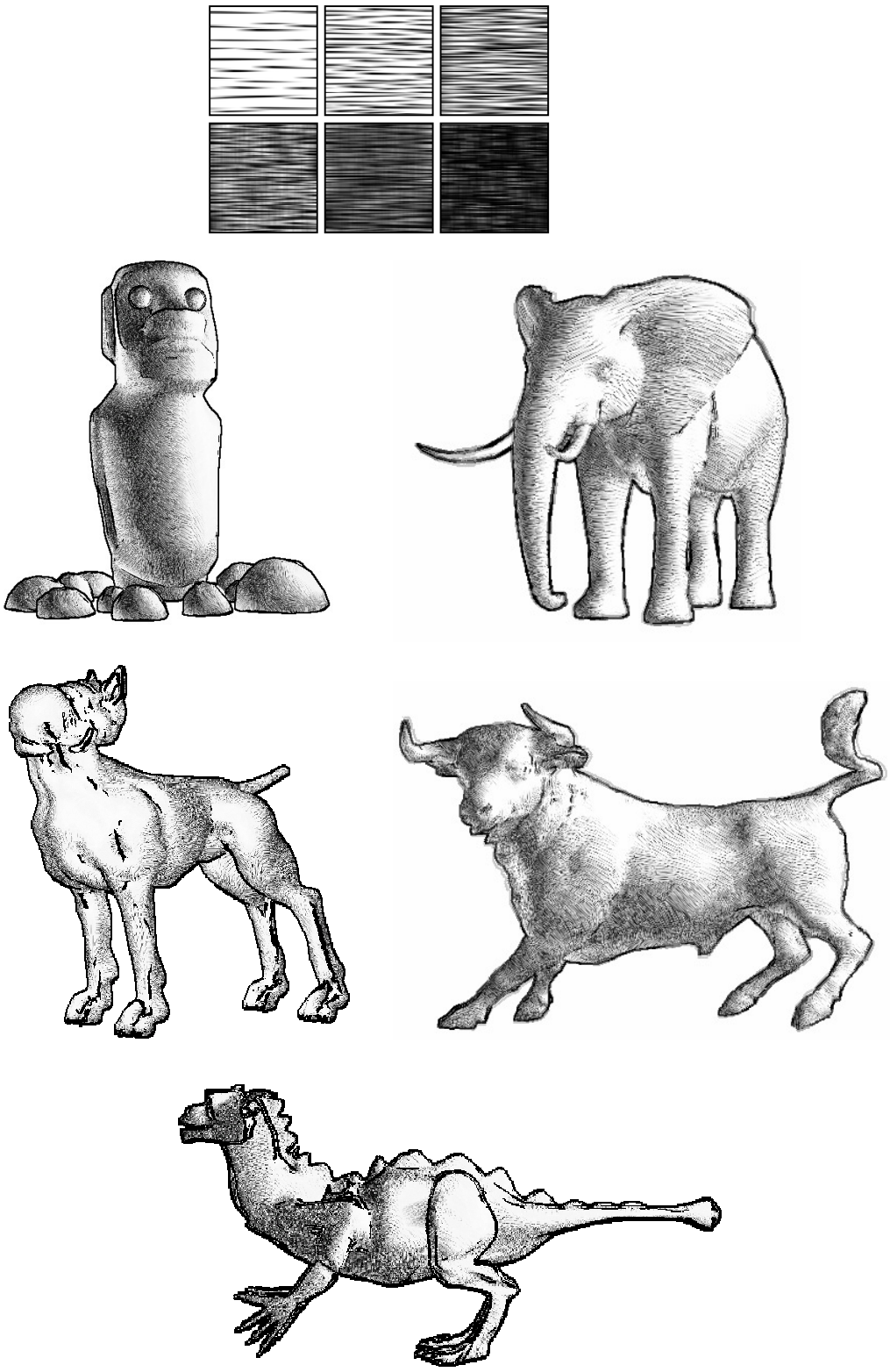
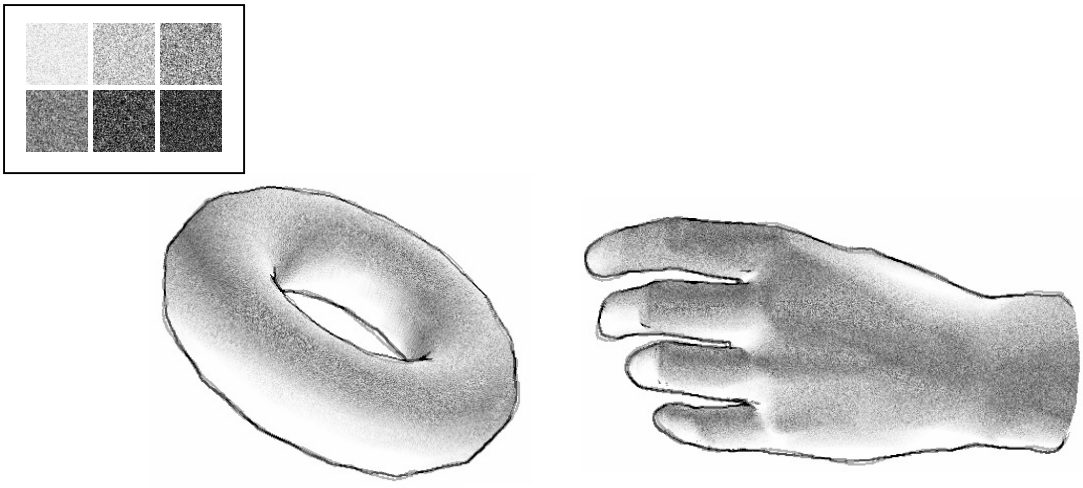
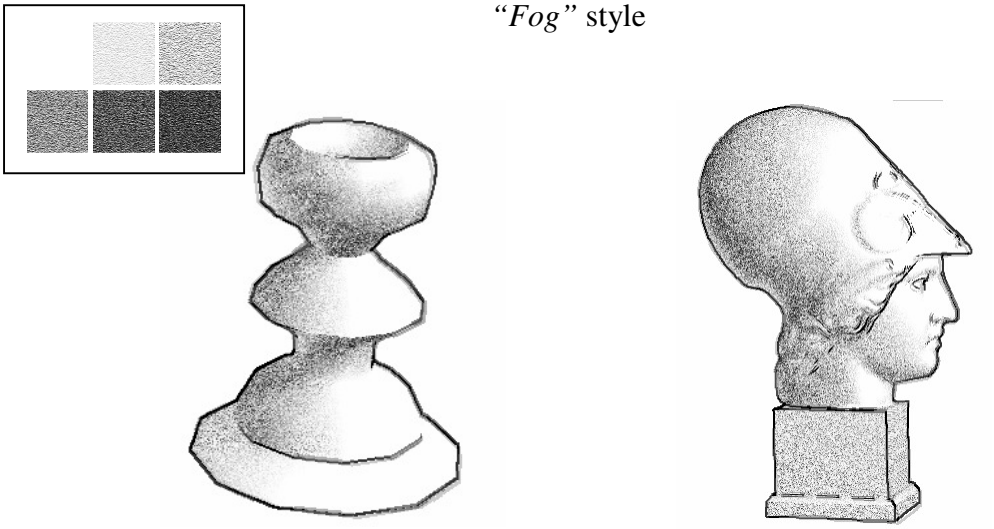


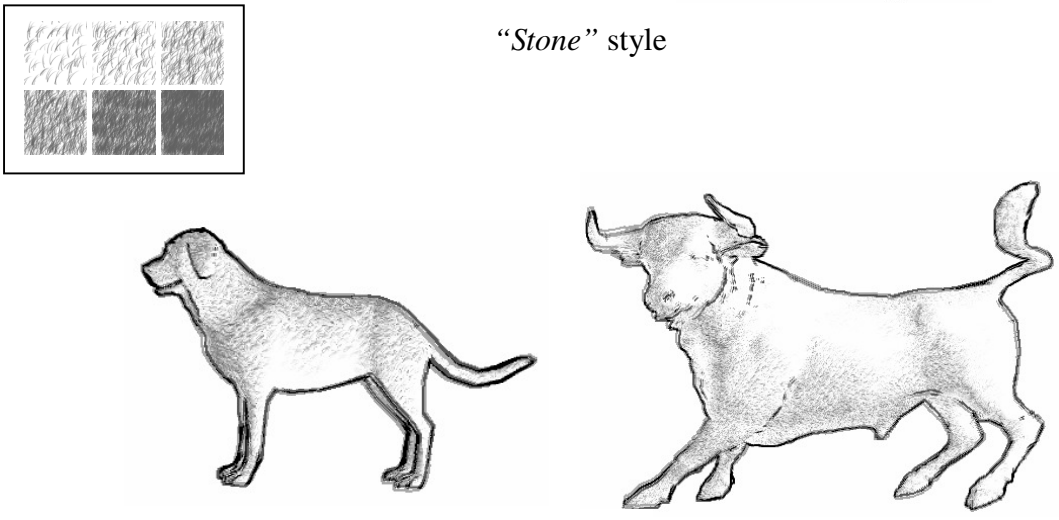
Figure 4-10: Results using *"Hatching"* style.



"Fog" style



"Stone" style



"Fur" style

Figure 4-11: Results of other artistic style not using direction filed.

Chapter 5

Conclusion and Future Works

We propose a fast rendering algorithm for generating sketchy drawings of arbitrary 3D geometries. We analyze the characteristics of human drawing and incorporated them into rendering process.

We introduce a fast algorithm of artist-like contour drawing in our system. We utilize depth map and normal map to detect and filter the principal edges. In order to imitate the characteristic of human drawing, we use the Perlin-noise technique to perturb the edges and combine the perturbed edges with different kinds of noises to the contour drawing. All these works are image-based algorithms, which ensure the NPR contour drawing can be generated in constant time.

Another contribution is that we enhance the interior shading using multi-rendering and direction field. In our work, we calculate the angle of vertex normal and light source to decide the intensity of surface. To implement effect of shadow and lighting on surface shading, we introduce multi-rendering to render textures at run-time. Therefore, intensity of shadow and lighting smoothly changes with the variation of brightness on the surface. Furthermore, in order to reveal the modeling of curvature, we also apply direction field algorithm in the interior shading.

Our system rotates stroke textures using direction field and integrates this method with multi-rendering technique. We can see strokes naturally follow the principal curvature directions. Based on the contour drawing and interior shading workflows, our framework permits an effective NPR rendering for many complex 3D scene.

We have already established a complete sketchy drawing framework. But there still remain a number of interesting areas for future works:

Smooth animation. Our sketchy rendering is effective enough to animate in real-time. But there still have unexpected problems when we zoom-in or zoom-out the models. Perturbed maps are unmodifiable in real time. Regardless of the distance between model and viewport, degrees of perturbation are constant. This effect causes visually unrealism in animation.

Stroke material growth. Although our system emphasizes the line-art sketching, we consider our framework can extend to many other artistic styles. We also wish our system can generate the stroke material automatically. We currently utilize the Perlin-noise on the contour perturbation. However, integrating the Perlin-noise with the material generating technique may better improve contour drawing and interior shading.

Interactive stroke direction control. Stroke direction is determined in preprocess step. The rotated angle may be opposite as to what user expects. It might be useful to design an interactive interface to control the direction field generation. User indicates a region of stroke direction to set all of principal curvature directions, and this can help our system to generate even high-quality NPR drawing.

Reference

[1] Barla, P., Thollot, J., and Markosian, L., 2006, *X-toon: an extended toon shader*, In Proceedings of NPAR ’06, pp. 127-133.

[2] Benichou, F., and Elber, G., 1999, *Output sensitive extraction of silhouettes from polygonal geometry*, In Proceedings of 7th Pacific Graphics Conference, IEEE CS Press, pp. 60-69.

[3] Buchanan, J., and Sousa, M., 2000, *The edge buffer: a data structure for easy silhouette rendering*, In Proceedings of NPAR ’00, Annecy, France, pp. 39-42.

[4] Chu, N.-S., and Tai, C.-L., 2005, *Moxi: real-time ink dispersion in absorbent paper*, In ACM Transactions on Graphics, vol. 24, no. 3, pp. 504-511.

[5] Curtis, C. J., Anderson, S. E., Seims, J. E., Fleischer, K. W., and Salesin, D. H., 1997, *Computer-generated watercolor*, In Proceedings of 24th Annual Conference on Computer Graphics and Interactive Techniques, pp. 421-430.

[6] DeCarlo, D., Finkelstein, A., Rusinkiewicz, S., and Santella, A., 2003, *Suggestive contours for conveying shape*, In Proceedings of International Conference on Computer Graphics and Interactive Techniques, pp. 848-855.

[7] Decaudin, P., 1996, *Rendu de scénes 3D imitant le style «dessin animé»*, In Rapport de Recherche 2919, Université de Technologie de Compiègne, France.

[8] Dominé, S., Rege, A., and Cebenoyan, C., 2002, *Real-time hatching (tribulations in)*, In Game Developers Conference, San Jose, CA.

[9] Freudenberg, B., Masuch, M., and Strothotte, T., 2002, *Real-time halftoning: a primitive for non-photorealistic shading*, In Proceedings of 13th Eurographics Workshop on Rendering, Pisa, Italy, pp. 1-4.

[10] Gooch, B., Gooch, A., Shirley, P., and Cohen, E., 1998, *A nonphotorealistic lighting model for automatic technical illustration*, In Proceedings of SIGGRAPH '98, pp. 447-452.

[11] Gooch, B., Gooch, A., Shirley, P., and Riesenfeld, R., 1999, *Interactive technical illustration*. In Proceedings of I3D '99, Atlanta, Georgia, United States, pp. 31-38.

[12] Gooch, B., and Gooch, A., 2001, *Non-Photorealistic Rendering*, Natick, MA: AK Peter, ISBN-13: 978-1568811338.

[13] Harris, M., Baxter, W., Scheuermann, T., and Lastra A., *Simulation of cloud dynamics on graphics hardware*, In Proceedings of ACM SIGGRAPH / Eurographics Workshop on Graphics Hardware, pp. 92-101.

[14] Hays, J., and Essa, I., 2004, *Image and video based painterly animation*, In Proceedings of 3rd International Symposium on Non-Photorealistic Animation and Rendering, Annecy, France.

[15] Hertzmann, A., 1998, *Painterly rendering with curved brush strokes of multiple sizes*, In Proceedings of ACM SIGGRAPH '98, Orlando, Florida, pp. 453-460.

[16] Hertzmann, A., and Zorin, D., 2000, *Illustrating smooth surfaces*, In Proceedings of SIGGRAPH '00, pp. 517-526.

[17] Isenberg, T., Halper, N., and Strothotte, T., 2002, *Stylizing silhouettes at interactive rates: from silhouette edges to silhouette strokes*, In Computer Graphics Forum, vol. 21, no. 3, pp. 249-258.

[18] Lake, A., Marshall, C., Harris, M., and Blackstein, M., 2000, *Stylized rendering techniques for scalable real-time 3D animation*, In Proceedings of NPAR '00, pp. 13-20.

[19] Lee, H., Kwon, S., and Lee, S., 2006, *Real-time pencil rendering*, In Proceedings of 4th International Symposium on Non-Photorealistic Animation and Rendering, pp. 37-45.

[20] Lei, E., and Chang, C. F., 2004, *Real-time rendering of watercolor effects for virtual environments*, In Proceedings of 5th Pacific Rim Conference on Multimedia, pp. 474-481.

[21] Lewis, J. P., Fong, N., Xiang, X. X., Soon, S.H., and Feng, T., 2005**,** *More optimal strokes for NPR sketching*, In Proceedings of 3rd International Conference on Computer Graphics and Interactive Techniques, pp. 47-50.

[22] Luft, T., and Deussen, O., 2005, *Interactive watercolor animations*, In Proceedings of Pacific Conference on Computer Graphics and Applications, pp. 7-9.

[23] Luft, T., Colditz, C., and Deussen, O., 2006, *Image enhancement by unsharp masking the depth buffer*, In ACM SIGGRAPH '06, pp. 1206-1213.

[24] Lum, E. B., and Ma, K. L., 2001, *Non-photorealistic rendering using watercolor inspired textures and illumination*, In Proceedings of 9th Pacific Conference on Computer Graphics and Applications, pp. 322-330.

[25] Mao, X., Nagasaka, Y., and Imamiya, A., 2001, *Automatic generation of pencil drawing from 2D images using line integral convolution*, In Proceedings of 7th International Conference on Computer Aided Design and Computer Graphics, pp. 240-248.

[26] Markosian, L., Kowalski, M., Goldstein, D., Trychin, S., and Hughes, J., 1997, *Real-time nonphotorealistic rendering*, In Proceedings of SIGGRAPH '97, pp. 415-420.

[27] McGuire, M., and Fein, A., 2006, *Real-time rendering of cartoon smoke and clouds*, In Proceedings of the 4th International Symposium on Non-Photorealistic Animation and Rendering, Annecy, France, pp. 21-26.

[28] McGuire, M., and Hughes, J., 2004, *Hardware-determined feature edges*, In Proceedings of NPAR '04, Annecy, France, pp. 135-147.

[29] Mitchell, J. L., Brennan, C., and Card, D., 2002, *Real-time image-space outlining for non-photorealistic rendering*, In Proceedings of SIGGRAPH, San Antonio, pp. 239.

[30] Nienhaus, M., and Döllner, J., 2003, *Edge-enhancement – an algorithm for real-time non-photorealistic rendering*, In Journal of WSCG, vol. 11, no. 2, pp. 346-353.

[31] Nienhaus, M., and Döllner, J., 2004, *Sketchy drawings*, In Proceedings of 3rd International Conference on Computer Graphics, pp. 73-81.

[32] Northrup, J. D., and Markosian, L., 2000, *Artistic silhouettes: a hybrid approach*, In Proceedings of 1st International Symposium on Non-Photorealistic Animation and Rendering, pp. 31-37.

[33] Praun, E., Hoppe, H., Webb, M., and Finkelstein, A., 2000, *Lapped textures*, In Proceedings of SIGGRAPH ’00, pp. 565-570.

[34] Praun, E., Hoppe, H., Webb, M., and Finkelstein, A., 2001, *Real-time hatching*, In Proceedings of SIGGRAPH ’01, pp. 579-584.

[35] Raskar, R., and Cohen, M., 1999, *Image precision silhouette edges*, In Proceedings of I3D ’99, pp. 135-140.

[36] Rossignac, J. R., and Van Emmerik, M., 1992, *Hidden contours on a frame-buffer*, In Proceedings of 7th Eurographics Workshop on Computer Graphics Hardware, pp. 188-204.

[37] Rustagi, P., 1989, *Silhouette line display from shaded models*, In Iris Universe Fall ’89, pp. 42-44.

[38] Saito, T., and Takahashi, T., 1990, *Comprehensible rendering of 3-D shapes*, In Proceedings of SIGGRAPH ’90, pp. 197-206.

[39] Salisbury, M. P., Wong, M., Hughes, J. F., and Salesin, D. H., 1997, *Orientable textures for image-based pen-and-ink illustration*, In Proceedings of ACM SIGGRAPH ’97, pp. 401-406.

[40] Sander, P., Gu, X., Gortler, S., Hoppe, H., and Snyder, J., 2000, *Silhouette clipping*, In Proceedings of SIGGRAPH ’00, pp. 327-334.

[41] Small, D., 1991, *Simulating watercolor by modeling diffusion*, In Proceedings of SPIE ’91, pp. 70-76.

[42] Sobel, I., 1990, *An isotropic 3x3 image gradient operator*, In Machine Vision for Three-Dimensional Scenes, Academic Press, pp. 376-379.

[43] Sousa, M. C., and Buchanan, J. W., 1999, *Computer-generated graphite pencil rendering of 3D polygonal models*, In Computer Graphics Forum, vol. 18, no. 3, pp. 195-208.

[44] Sousa, M. C., and Buchanan, J. W., 2000, *Observational models of graphite pencil materials*, In Eurographics, In Computer Graphics Forum, vol. 19, no. 1, pp. 27-49.

[45] Sousa, M. C., and Prusinkiewicz, P., 2003, *A few good lines: suggestive drawing of 3D models*, In Eurographics, Computer Graphics Forum, vol. 22, no. 3, pp. 381-390.

[46] Webb, M., Praun, E., Finkelstein, A., and Hoope, H., 2002, *Fine tone control in hardware hatching*, In Proceedings of NPAR ’02, pp. 53-58.

[47] Wilson, B., and Ma, K. L., 2004, *Rendering complexity in computer-generated pen-and-ink illustrations*, In Proceedings of NPAR, pp. 129-137.

[48] Winkenbach, G., and Salesin, D. H., 1994, *Computer-generated pen-and-ink illustration*, In Proceedings of Computer Graphics on ACM SIGGRAPH '94, pp. 91-100.

[49] Van Laerhoven, T., Liesenborgs, J., and Van Reeth, F., 2004, *Real-time watercolor painting on a distributed paper model*, In Proceedings of Computer Graphics International, pp. 640-643.

[50] Xu, S., Xu, Y., Kang, S. B., Salesin, D. H., Pan, Y., and Shum, H. Y., 2006, *Aminating chinese paintings yhrough stroke-based decomposition*, In ACM Transactions on Graphics, vol. 25, no. 2, pp. 239-267.

[51] Zakaria, N., and Seidel, H., 2004, *Interactive stylized silhouette for point-sampled geometry*, In Proceedings of 2nd International Conference on Computer Graphics and Interactive Techniques, pp. 242-249.