Rendering pipeline for anime-like shading.

Jia, Lei.

2010

Jia, L. (2010). Rendering pipeline for anime-like shading. Master' s thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/43293>

<https://doi.org/10.32657/10356/43293>

RENDERING PIPELINE FOR ANIME-LIKE SHADING

A Thesis

Submitted to School of Computer Engineering
Nanyang Technological University

By

JIA LEI

Master of Engineering (M. Eng)

SCHOOL OF COMPUTER ENGINEERING

Year 2009/2010

Abstract

Anime, or Japanese Animation, origins in manga (Japanese comics) has its unique recognizable style and a wide range of audience not only in Japan but also around the world. Traditional 2D anime production has always been time-consuming and labor-intensive, especially for the coloring of the shading part, which is the most distinguished part of anime from other cartoons. In this thesis, we propose a semi-automatic shading pipeline based on surface reconstruction for 2D animation, especially for Japanese anime style to provide more efficient method of performing the tasks. This technique will be investigated and introduced into an existing animation system known as CACAni (Computer – Assisted Cel Animation).

In traditional 2D animation production, each stroke in every frame, including the highlight and shade strokes, has to be drawn by hand. Hence it takes long time to complete an animation. Our technique reduces time needed for the production by automatic shade and highlight generation based on a stroke-based surface reconstruction method. The only input required is a hand-drawn character.

We present surface reconstruction via 2D strokes and vector field on the strokes based on a two-step method. In the first step, from given sparse strokes drawn by artists and a vector field on the strokes, we propose a nonlinear vector interpolation combining total variation (TV) regularization and H^1 regularization with a curl-free constraint in order to obtain a dense vector field. In the second step, we propose a height map reconstruction algorithm which integrates the dense vector field in the first step. Since vectors on the strokes are interpreted as a projection of surface

normal vectors into the plane, we can design different types of strokes to generate geometrically crucial shapes such as ridge, valley, jump, bump, and dip on the surface. The stroke types help users to create a surface which they intuitively imagine from 2D strokes. The reconstructed surface faithfully reflects the assigned vector information on strokes. Jump discontinuities in surface and discontinuities of surface gradient can be well reconstructed without any surface distortion. We compare our results with conventional methods via many examples.

In addition, since artists tend to touch up the rendering results to reflect their intension and preference, we provide them with several ways to intuitively edit the obtained rendering results. Artist-oriented shading and highlighting is thus achieved semi-automatically with little user interactions.

The work is a part of a larger project, the main contribution of the writer is the shading pipeline, the editing operations of the shading result and the GPU computing for surface reconstruction.

Acknowledgement

I would like to dedicate my most sincere thanks to Professor Seah Hock Soon, my thesis adviser, for his guidance, encouragement and support throughout the research. Special thanks go to Dr. Tian Feng for his generous advice at the initial phase of my research. I appreciate the valuable suggestions and continuous support from Professor Tai Xue Cheng from the School of Physical and Mathematical Sciences.

Deep thanks go to my teammates, Dr. Qiu Jie, Dr. Eiji Sugisaki and Dr. Hahn Jooyoung. It is always my pleasure to work together with them.

I would like to express my gratitude to the artists in Game lab, Cao Youfang, Desiree, Low Zirong, and Liew Hongze for providing great demonstration examples and valuable feedbacks on the project.

I must also thank my beloved family and friends for their indispensable source of motivation and encouragement. I deeply appreciate my lab mates for making the working life an enjoyable one. Lastly, I am grateful for all the help I have received in one way or another.

Table of Contents

Abstract	i
Acknowledgement	iii
Table of Contents	iv
List of Figures	vi
CHAPTER ONE: INTRODUCTION	- 1 -
1.1 Background	- 1 -
1.2 Objectives	- 2 -
1.3 Overview	- 4 -
CHAPTER TWO: REVIEW OF THEORY AND PREVIOUS WORK	- 6 -
2.1 Computer Animation	- 6 -
2.2 Surface Reconstruction	- 7 -
2.3 Toon-Shading	- 10 -
2.4 User Interface for Editing	- 13 -
2.5 CUDA	- 14 -
CHAPTER THREE: STROKE-BASED SURFACE RECONSTRUCTION AND GPU COMPUTING ..	- 15 -
3.1 Surface Reconstruction Process	- 15 -
3.2 Stroke Definition and Classification	- 17 -
3.2.1 Stroke Definition	- 17 -
3.2.2 Stroke Classification	- 18 -
3.3 GPU Computing	- 20 -
3.3.1 CUDA Process Flow	- 20 -
3.3.2 Results and Comparison	- 22 -
CHAPTER FOUR: SHADING PIPELINE	- 24 -
4.1 Toon-Shading	- 25 -

4.2	Editing Operations.....	- 27 -
4.2.1	Texture Color Editing.....	- 29 -
4.2.2	Editing Obtained Result Directly	- 30 -
4.2.3	Light Source Controls	- 31 -
4.2.4	3D view of the model	- 34 -
4.2.5	Vectorizing the Obtained Result	- 35 -
CHAPTER FIVE: RESULTS AND ANALYSIS		- 36 -
5.1	Surface Reconstruction	- 36 -
5.2	Shading Pipeline	- 43 -
CHAPTER SIX: CONCLUSION AND FUTURE WORK.....		- 47 -
Appendix		- 49 -
References.....		- 63 -

List of Figures

Figure 1.1: (a) Anime-Like Shading Result, (b) Example for support lines for highlight and shade. (c) Shading for representing negative emotion.....	-2-
Figure 1.2: Overview of the shading pipeline.....	-4-
Figure 2.1: Toon Shading Texture.....	-11-
Figure 2.2: Lighting Model.....	-11-
Figure 2.3: (a) Phong Shading Torus, (b) Toon Shading Torus.....	-11-
Figure 3.1: Overview of process	-14-
Figure 3.2: Cleaned-up vectorized strokes	-18-
Figure 3.3: Effects of the different types of strokes	-19-
Figure 3.4: Processing flow of CUDA	-21-
Figure 4.1: Overview of the shading pipeline	-24-
Figure 4.2: Overview of Toon Shading.....	-25-
Figure 4.3: Example of Texture. Black: Contour Line, Dark Purple: Shade Part, Purple: Base Color, White: Highlight.....	-26-
Figure 4.4: Editing User Interface.....	-27-
Figure 4.5: Texture Color Editing	-29-
Figure 4.6: Editing Obtained Result Directly.....	-30-
Figure 4.7: ρ - θ - ϕ light control UI.....	-31-
Figure 4.8: ρ - θ - ϕ value computation.....	-32-
Figure 4.9: phong shading and toon shading comparision result (a) phong shading, (b)toon shading	-34-
Figure 4.10: Editing Vectorized Result.....	-35-
Figure 5.1: Simple shape example.....	-36-
Figure 5.2: Dokkaebi example: (a) original image in Siggraph Asia 2010 website. (b) drawing traced by an artist. (c) vector setting, all vectors are orthogonal to the stroke tangent directions and with the same magnitude. (d) dense vectors obtained by the proposed

<i>nonlinear interpolation from the sparse vectors in (c). (e) reconstructed surface.....</i>	<i>-37-</i>
<i>Figure 5.3: Comparison with related works.....</i>	<i>-38-</i>
<i>Figure 5.4: Indian lady example: (a) line drawing. (e) assigned vectors on the strokes in (a). (b) and (f) are the dense vector field and reconstructed surface using the LUMO method, respectively. (c) and (g) dense vector field and reconstructed surface using the ShapePalettes method, respectively. (d) and (h) dense vector field and reconstructed surface using our proposed method, respectively.....</i>	<i>-39-</i>
<i>Figure 5.5: Female & male examples.....</i>	<i>-40-</i>
<i>Figure 5.6: Cat example: Comparison among LUMO, ShapePalettes, and our proposed method</i>	<i>-41-</i>
<i>Figure 5.7: Fisting animation Sequence</i>	<i>-42-</i>
<i>Figure 5.8: Hair Shading Result (a) Hairstyle outlines with the detected skeleton (b) Assigned initial vectors on the hairstyle outlines and skeletons (c) Result with skeletons, (d) Result without skeletons.....</i>	<i>-43-</i>
<i>Figure 5.9 Review of our shading pipeline (a) Input hand-drawing, (b) assigned initial vectors, (c) Processed Shading Result, (d) Minor Editing applied to (c)</i>	<i>-44-</i>
<i>Figure 5.10 Various Results reflecting artists' preference (a) Processed Shading Result, (b) Artist's work 1, (c) Artist's work 2.....</i>	<i>-45-</i>
 <i>Table 3.1: Average Computational cost for different data</i>	 <i>-22-</i>
<i>Table 3.2: Computational cost for some examples.....</i>	<i>-23-</i>
<i>Table 4.1: Comparison between ours and the other 3 editing UI.....</i>	<i>-28-</i>

CHAPTER ONE: INTRODUCTION

The work is a part of a larger project, the main contribution of the writer is the shading pipeline, the editing operations of the shading result and the GPU computing for surface reconstruction.

1.1 Background

CACAni (Computer Assisted Cel Animation) is a system that been developed for traditional animation production industry especially in 2D Anime. [Seah et al. 2006] [J Qiu et al. 2005] [Q Chen et al. 2006] [E Sugisaki et al. 2009]. The key functions for the system are line drawing, automatic in-betweening and coloring. There are several layers that correspond to each of these components, such as the character or object layer, shadow layer, and background layer. All layers are composed together for the final animation in Anime production. The time needed for the production can be saved with CACAni system's production pipeline. However, although CACAni has auto-coloring function, the coloring for the key frame is still fully based on artist's work, which is time consuming.

As one of the most common application areas for CACAni, Japanese cartoon animation so-called "Anime", has its unique recognizable style and a wide range of audience not only in Japan but also around the world. As a special art form, anime can have a wide variety of the visual styles according to an artist, a production studio, and even a production. Although such a variety exists, there are several

commonalities which can be viewed as the characteristics of anime. Shading style is one of the features distinguishing anime from other cartoons.

1.2 Objectives

Figure 1.1: (a) *Anime-Like Shading Result*, (b) *Example for support lines for highlight and shade*. (c) *Shading for representing negative emotion*

As illustrated in Figure 1.1(a), the styles of highlight and shade are essential elements of Anime-like shading, which are used to emphasize the lighting, depth, material and even emotion and feelings of the character. As shown in Figure 1.1(b), artists usually draw several kinds of lines to create highlight, shade, etc. In Anime, highlight and shade are rendered not only to depict the relationship between characters or object and light sources. They portray the geometry information such as flat or non-smooth, and the material information such as shiny or matte. In addition, highlight and shade can be utilized to grab the audience's attention to the characters and scenes. In fact, highlight and shade play an important role [Anjyo et al. 2000, 2003] to represent

where the light source is and at the same time they can express a character's emotion and feelings in 2D animation. Audience is able to recognize how the character feels from the shading and highlight which is not related to the actual lighting information (Figure 1.1(c)).

Despite the importance and usefulness of highlight and shade [Anjyo et al. 2003], artists often roughly draw them due to limited human resources and time constraints in production. Since to draw detailed highlight and shade in each individual frame requires a substantial amount of time [Lasseter, 1987], artists frequently do not have enough time to draw it to their artistic taste.

Therefore, we propose a rendering pipeline in this research to reduce time required for practical production by providing semi-automatic shading and highlighting. In addition, we also aim to give artists freedom to reflect their taste and preference by providing interactive and intuitive editing methods for the rendered image. Ultimately our goal is to enable artists to easily create anime-like shade and highlight based on their hand-drawn image in a shorter time than they usually take. To simplify the problem, this retouch focuses on the geometry, and ignores the shading effects due to other factors like representing self-shadows or expressing character emotions.

As we mentioned in the previous section, this technique will be introduced into the CACani system to reduce the time for coloring.

1.3 Overview

Figure 1.2: *Overview of the shading pipeline*

Figure 1.2 shows the overview of the process for our shading pipeline.

Our pipeline takes the line drawings of animations as the input, and the characters are usually drawn in layers. Based on the input, the initial vectors are assigned to the drawn character semi-automatically, which are subsequently adjusted with minor user-interactions. Then, using the adjusted initial vectors, we reconstruct the surface of the character. Chapter 3 will describe how we adjust the initial vector and reconstruct the surface.

Next, we apply toon-shading based on the surface normals to obtain the anime-like shading result. To reflect artist's taste and preference, we also provide them with interactive editing methods. More details are explained in Chapter 4.

In Chapter 5, the result images are illustrated and the advantages of our pipeline based on the obtained results are discussed.

Finally, in Chapter 6, the conclusion is given and the future work of our research is mentioned.

The appendix at the end gives more details on other technical parts used in our research which are not mainly contributed by the writer.

CHAPTER TWO: REVIEW OF THEORY AND PREVIOUS WORK

In this section, the overview of the computer graphics techniques for cartoons is given first, and related works specialized for surface reconstruction and shading and highlighting are then described.

2.1 Computer Animation

Disney's movies, such as *The Great Mouse Detective* (1986) and *Beauty and the Beast* (1991), are considered as the first hybrid uses of 3D and hand-drawn characters. After these movies, various kinds of cel animation movies have been created by the rapid progress of 3DCG technology [Anjyo, et al. 2000]. Lasseter [Lasseter, 1987] is very likely the first researcher to describe the basic principles of traditional 2D hand-drawn animation and their application to 3D computer animation. In his paper, he clearly described cartoon animation and what it requires of an artist. Recently, 3D computer graphics techniques are used in the Japanese "Anime" industry (e.g. *Ghost in the Shell* and *Innocence*). The movie *Appleseed* is a landmark anime movie featuring hyper realistic imagery and a hybrid 2D and 3D style [Krikke, 2006].

2.2 Surface Reconstruction

Sketch-based interfaces for modeling (SBIM) has been substantially explored by many researchers because of efficiency and intuitiveness, from the early systems like SKETCH [Zelevnik et al. 1996] and Teddy [Igarashi et al. 1999] to recent SmoothSketch [Karpenko and Hughes 2006] and FiberMesh [Nealen et al. 2007]. A thorough review of SBIM systems can be found in [Olsen et al. 2009]. Without lighting or shading cues used in photometric stereo [Woodham 1989; Hertzmann and Seitz 2005; Wu and Tang 2006a] or shape-from-shading [Zhang et al. 1999; Parados and Faugeras 2006], the task of 3D model reconstruction from 2D line drawings is more challenging than image-based 3D reconstruction. A lot of research efforts have been expended towards this challenging task based on contours [Igarashi et al. 1999], hidden contours [Karpenko and Hughes 2006], and structured annotations [Gingold et al. 2009]. The models created by these systems are limited in shape with rough, smooth, or rounded structures. Moreover, surface details such as the crease shape are not considered in these systems. Note that it can be later added by surficial augmentation techniques [Olsen et al. 2005].

Motivated by the above mentioned works, we are targeting at a sketch-based modeling system which can model complex 3D objects with simple strokes. As a first attempt to this direction, we present a research achievement on the quality surface reconstruction via 2D strokes and vector field on the strokes. This algorithm is proposed by Dr. Hahn Jooyoung, Dr. Qiu Jie and Dr. Eiji Sugisaki, and the writer is in charge of implementing and accelerating the algorithm based on GPU computing.

Many surface reconstruction algorithms [Agrawal et al. 2006; Frankot et al. 1987; Petrovic et al. 2001; Simchony et al. 1990] enforce the integrability to produce surface heights from the gradient field. For single-view modeling, the authors [Zhang et al. 2002] used a constrained optimization with point constraints, depth discontinuities, creases, planar region constraints, and fairing curve constraints. However, the method in [Zhang et al. 2002] requires a lot of user interactions to provide enough constraints for a desirable surface; see more details in [Wu et al. 2007].

Another single-view modeling system [Prasad and Fitzgibbon 2006] used a close form method to reconstruct curved 3D surfaces based on apparent contour, inflation constraints, and normal specification in the parameter space. But the parameter space may not be intuitive for the artists. Recently, the authors in [Ng et al. 2009] highlighted that it will be difficult for these algorithms to solve three problems in surface-from-gradients: handling sparse gradients, preserving sharp features, and preventing surface distortion. To solve the problems, Gaussian kernel approach without discrete integrability enforcement is used in [Ng et al. 2009]. The proposed algorithm is also capable of handling these problems because of TV regularization and curl-free constraints in a nonlinear vector interpolation. On the top of this, fast computational results are obtained via augmented Lagrangian method [Wu et al. 2009].

Based on the observation that humans are good at assigning local surface normals for specifying local shape [Koenderink 1998], the authors in [Johnston 2002; Wu et al. 2007] already used a two-step method in stroke-based surface reconstruction. In the

first step, dense surface normals are obtained from sparse user-input vectors on given strokes via linear vector interpolation methods. In the second step, the dense normals are integrated in order to reconstruct a 3D surface. The LUMO method [Johnston 2002] for interpolating vectors from a fixed vector field on sparse strokes is based on so-called Telegrapher's equation which is a damped wave equation. To generate suitable results for the lighting approximation, the layering technique should be additionally used.

In ShapePalettes [Wu et al. 2007], an energy function minimization is used and the main mechanism of interpolation is based on a parabolic type partial differential equation (PDE) with the fourth order term which minimizes the surface curvature. Since the wave speed in the diffusion equation is infinite, few iterations are needed to generate a suitable shape from the inpainting vectors in the ShapePalettes model. On the other hand, the LUMO model usually needs more iterations because of the finite speed of propagation. The advantage of LUMO model and ShapePalettes model is that the governing equations are linear PDEs so that efficient numerical solvers such as the multi-grid method or the fast Fourier transform can be applied. However, both models are based on the Laplace operator which enforces smoothness in interpolated vector field. That is, the assigned vectors on strokes are diffused into the domain and then eventually averaged out in the steady state. Therefore, reconstructed surfaces via the interpolated vectors from these models cannot have geometrically crucial structures such as jump discontinuities and sharp ridges or valleys.

The pioneering work for surface reconstruction from sparse information is the visible-surface representations in [Terzopoulos 1988]. The weighted H^1 norm of data

and its gradient with discrete fidelities locally controls smoothness of reconstructed surface. Since jump discontinuities under TV norm of data [Rudin et al. 1992] are easily preserved without weighting functions, discontinuities of surface gradient are also recovered by TV norm of data gradient; see more details in [Chambolle and Lions 1997]. It implies that TV regularization in functional minimization is crucial to preserve discontinuous structures in surface and its gradient field. We explore even further to design a model for stroke-based surface reconstruction via a combination of TV norm and H^1 norm. It gives artists more freedom to create various types of surfaces. The bottleneck of using the TV norm was the computational cost to reconstruct a surface. However, it is improved in [Wu et al. 2009] via an augmented Lagrangian method and the method can be easily used in our proposed model.

2.3 Toon-Shading

Toon-Shading is a popular way of expressing Cartoon-like rendering. It has been shown in a variety of 3D rendering software, video games, and animations. The idea is quite simple, at the same time powerful. The shading utilizes a texture that represents how the final shading is handled from the basic color to darker color.

Barla et al [Barla et al. 2006] proposed a novel idea for the extended toon shader. They described two extensions to the basic algorithm for handling view-dependent effects. They utilize a 2D texture which can handle so-called "toon-detail" that can vary according to the depth and surface orientation. The other extension was to extend the toon shader for the use of a modified normal field which can range from the initial normals to a simpler set of normals taken from so-called abstracted shape.

They basically developed a "level of abstraction" concept in the perspective of toon shading. Their concept is sophisticated. We adopt the basic toon shading to achieve an anime-like shading result.

The toon shading we used can be simply illustrated by Figure 2.1:

Figure 2.1: *Toon Shading Texture*

Figure 2.2: *Lighting Model*

As shown in Figures 2.1 and 2.2, we use a texture to color an object. In the shader, the texture coordinates t is computed as the dot product of the eye vector and the normal; texture coordinates s is computed as the dot product of the light vector and the normal.

One example of toon shading result is shown in Figure 2.3.

Figure 2.3: (a) *Phong Shading Torus*, (b) *Toon Shading Torus*

2.4 User Interface for Editing

Since our approach aims to be applied in anime production, we then refer to several user interfaces that have been proposed for editing shades practically in computer graphics.

Anjyo et al. [Anjyo et al. 2003] developed a system that can render stylized highlights on 3D object used in cel animation. Several user-friendly operations were implemented in their system to modify the highlight vectors. As an extension method to [Anjyo et al. 2006], users can directly manipulate stylized light and shade in real-time with explicit light source control. A set of simple algorithms enables the interactive controlling of highlight and shade by intuitive operation such as click-and-drag. Therefore, users have a more straightforward way to control highlight and shade than tuning parameters. Our editing approach uses their concept as reference.

Shimotori et al. [Shimotori et al. 2007] proposed an interactive way to directly edit shade by adjusting the surface normals on triangles by simple mouse operations. The idea of changing the surface normals facilitates the rendering of other frames as the normals are not affected by the camera movement. In terms of the editing concept, we also referred to their ways. However, our editing system is more intuitive, providing animators with several editing methods, which will be discussed later in section 4.2.

2.5 CUDA

CUDA (an acronym for Compute Unified Device Architecture) is a parallel computing architecture developed by NVIDIA. CUDA is the computing engine in NVIDIA graphics processing units (GPUs) that is accessible to software developers through variants of industry standard programming languages. Programmers use 'C for CUDA' (C with NVIDIA extensions and certain restrictions), compiled through a PathScale Open64 C compiler [DailyTech, 2008], to code algorithms for execution on the GPU.

CUDA gives developers access to the virtual instruction set and memory of the parallel computational elements in CUDA GPUs. Using CUDA, the latest NVIDIA GPUs become accessible for computation like CPUs. Unlike CPUs however, GPUs have a parallel throughput architecture that emphasizes executing many concurrent threads slowly, rather than executing a single thread very fast. This approach of solving general purpose problems on GPUs is known as GPGPU.

In the computer gaming industry, in addition to graphics rendering, GPUs are used in game physics calculations (physical effects like debris, smoke, fire, fluids); examples include PhysX and Bullet. CUDA has also been used to accelerate non-graphical applications in computational biology, cryptography and other fields by an order of magnitude or more. [Schatz et al. 2007].

CHAPTER THREE: STROKE-BASED SURFACE RECONSTRUCTION AND GPU COMPUTING

As mentioned in the previous section, Dr. Hahn Jooyoung, Dr. Qiu Jie and Dr Eiji Sugisaki proposed the stroke-based surface reconstruction algorithm. The writer implemented this algorithm using GPU computing (CUDA based) and comparing the experiments results with other algorithms.

3.1 Surface Reconstruction Process

The process for surface reconstruction from sparse strokes and a piecewise smooth vector field on the strokes is shown in Figure 3.1.

Figure 3.1: *Overview of process*

In the drawing stage, an artist first draws strokes to create the basic geometry and our system automatically assigns the vectors which are orthogonal to the stroke tangents. The vectors can be easily adjusted by the artist according to a desired surface shape. Since the vectors on the strokes are interpreted as a projection of surface normal vectors onto the plane, we can design different types of strokes to generate geometrically crucial shapes such as ridge, valley, jump, bump, and dip on the surface. The proposed model for surface reconstruction is divided into two steps. In the interpolation stage in Figure 3.1, we use a nonlinear vector interpolation to obtain dense normal vectors in the computational domain.

The TV regularization allows preserving sharp ridges or valleys and the curl-free constraints enforce interpolation between vectors on sparse strokes in a large domain. In the reconstruction stage in Figure 3.1, a height map is reconstructed via integrating the dense normal vectors in the first step. The TV regularization also allows the preservation of jump discontinuities in a reconstructed surface while eliminating surface distortion. The reconstructed surface faithfully reflects the assigned vector information on strokes. The main contributions of our research are as follows:

- Strokes are categorized to create sharp ridge or valley, jump, bump, and dip on the surface. The stroke types help users to create a surface which they intuitively imagine from 2D strokes.
- The proposed nonlinear vector interpolation and height map reconstruction faithfully reflects the assigned vector information on strokes. From a combination of TV regularization and H^1 regularization, geometrically crucial structures given by strokes settings are well preserved without any distortion.

Moreover, the sparse vectors are well interpolated on a large domain with curl-free constraints.

- Even though the nonlinear process is used in both vector interpolation and height map reconstruction, we obtain fast numerical results via an augmented Lagrangian method.

3.2 Stroke Definition and Classification

3.2.1 Stroke Definition

In terms of sketch-based modeling for surface reconstruction, it would be better to use simple and intuitive line drawings which capture the characteristics of surface. Since perception of shape is little affected by lighting and reflectance modification [Judd et al. 2007], we use line drawings without shading and highlighting strokes. Hatching strokes and stippling depicting texture or shading are also excluded in the range of our stroke definition. We also ignore texture, lighting, and shadows on a surface [Hertzmann and Zorin 2000]. The definition of strokes is the style of silhouette drawing which only captures the geometry of a shape. The cleaned-up vectorized strokes drawn by artists are mostly traced from the original raster drawings, as shown in Figure 3.2.

Figure 3.2: *Cleaned-up vectorized strokes*

3.2.2 Stroke Classification

Inspired by a recent study on where artists draw lines to convey 3D shapes [Cole et al. 2008], we roughly classify strokes into different types to convey local structures of surface. Even though there is no quantitative measurement to classify which structures are more crucial in surface reconstruction, human vision easily tends to detect jump, ridges or valleys, bump or dip, and smooth regions. The majority of sparsely drawn strokes in our research are silhouette lines, occluded contours, ridges or valleys. Therefore, we prefer to use the following types of strokes which help users to create a surface which they intuitively imagine from 2D strokes.

- Regular stroke: It reflects the rate of changes in the surface.
- Ridge stroke or valley stroke: It reflects sharp ridges or sharp valleys in the surface. It is interpreted as discontinuities of gradient of height map.
- Jump stroke: It creates a step jump in the height map of surface. It is considered as jump discontinuities in the height map.

- Bump stroke or dip stroke: It creates shapes of bump or dip on the surface. Even though it is also considered as discontinuities of gradient of height map, it is easily recognized as different shapes from sharp ridges or sharp valleys.

The main reason we can reconstruct a proper height map indicated by different types of strokes is that we assign geometrically meaningful vectors on sparse strokes depending on types of strokes and we use a nonlinear vector interpolation to obtain a dense normal vector field. Figure 3.3 shows reconstructed surfaces from different types of strokes. The regular, ridge, valley, jump, bump, and dip strokes are illustrated in black, red, blue, green, violet, and cyan respectively.

Figure 3.3: *Effects of the different types of strokes*

The detail of the algorithm of the stroke-based surface reconstruction is included in the appendix.

3.3 GPU Computing

During the process of implementing the surface reconstruction algorithm, some equations (see appendix A-3, A-4, and A-6) required Fourier Transform to be used repeatedly, which takes significant time if we use CPU to compute. Therefore, CUDA is used to implement this part since it provides Fast Fourier Transform function to solve this problem.

The inputs of the algorithm are images. These images with strokes will be processed pixel by pixel. As a result, large size of image takes long time to process. Implemented using CUDA can enable the GPU power to process the image data in parallel.

3.3.1 CUDA Process Flow

The parallel GPU computing is implemented as follows:

- Data is copied from the main CPU memory to GPU memory.
- CPU functions are to instruct the process to be executed in GPU.
- GPU execute in parallel in each core.
- Result from the GPU memory is copied to the main CPU memory.

The processing flow of CUDA is shown in Figure 3.4.

Figure 3.4: *Processing flow of CUDA*

One limitation of using CUDA is that CUDA uses a recursion-free, function-pointer-free subset of the C language, plus some simple extensions. C++ is not fully supported now. In the actual implementation, two different files use the same extension `.cu` – `algorithm.cu` and `algorithm_kernel.cu`. `algorithm.cu` includes all the declarations of the GPU memory and instructions of GPU function which can use C++, while `algorithm_kernel.cu` handles all the GPU computing in parallel, which uses pure C language.

3.3.2 Results and Comparison

The configuration of the computer used for our experiments is as follows:

Intel dual core at 2.27GHz CPU; 2.93 GB of RAM; NVIDIA Quadro FX 1800 display card.

Based on CUDA, parallel computing is implemented to process the image data, which is on average 10 times faster than normal CPU computing. In the experiments, if the domain size of the input data is less than 256^2 , real time computing can be achieved.

Table 3.1 shows the average cost of simple surface reconstruction (without second step) for different size of data using both CPU and GPU. The larger the image size is the more effective GPU computing works.

Image size	Average Computational Cost (ms)	
	GPU	CPU
256 *256	1615	12474
512*512	4616	43076
1024*1024	15742	169482

Table 3.1: Average Computational cost for different data

Table 3.2 shows the computational cost of some examples. All the examples are shown in Chapter Five, section 5.1. The iteration number is constrained by an error bound.

Example	Size	First Step		Second Step	
		Iter No.	Time (ms)	Iter No.	Time (ms)
Simple Shape	256^2	64	1843	82	2063
Dog	256^2	56	1672	82	2175
Dokkaebi	512^2	57	5152	82	7012
Indian Lady	512^2	42	4329	82	6687
Cat	512^2	42	4359	82	6813
Fist	512^2	49	5078	82	6703

Table 3.2: *Computational cost for some examples*

CHAPTER FOUR: SHADING PIPELINE

In this chapter, we describe our shading pipeline to obtain anime-like shading result. As described in the previous chapter, once we have obtained the surface normals, shading based on the normals are applied to the input image. This section is mainly contributed by the writer.

Figure 4.1: *Overview of the shading pipeline*

As shown in Figure 4.1, our pipeline describes an entire work flow from hand drawn character images to animator's desired results. In this work flow, only little animator's interaction is needed.

4.1 Toon-Shading

Toon shading is a well-known method to render target models for more than 10 years. It can be seen in a variety of 3D renderers, game industries, entertainment systems, and animations. The idea of the shading, quite simple but powerful, extends the Lambertian shading model by using dot products between normals and light vectors as well as normals and eye vectors at each pixel to index into a texture that represents how the shading is implemented (Figure 4.1). The result can be “highlight”, “shade”, basic part color, and more colors depending on the texture.

Figure 4.2: *Overview of Toon Shading*

In our case, to obtain the anime-like shade, we prepare a 4 colors texture that represents “highlight”, “shade”, basic part color, and the contour lines (usually black, shown in Figure 4.2). Based on the obtained surface normals, we can apply the toon-shading to the input image itself even though we do not have a 3D structure of the input drawing. First, we need to obtain dot product s between a normal vector and the light vector and another dot vector t between a normal vector and the eye vector at

the target vertex. Then, s is set to x-axis and t is set to y-axis. These axes are corresponded to the texture coordinates. Thus, based on these dot product values, the color on the target vertex is determined according to the result on the texture color. For instance in Figure 4.2, if the vertex has the value $t = 0.8$ and $s = 0.8$, the vertex color is determined as purple. Thus the rendering is implemented by mapping onto the target surface based on the results of these dot products.

Figure 4.3: *Example of Texture. Black: Contour Line, Dark Purple: Shade Part, Purple: Base Color, White: Highlight*

4.2 Editing Operations

The shading result obtained from the shading pipeline is singular. To allow artists to edit the obtained result to reflect their artistic features and preferences, we design an editing user interface to provide artists with several ways of doing editing intuitively. To see the several kinds of shading results, we also give 3D control (translation, zooming and height adjustment) to artists. Figure 4.3 shows the editing user interface.

Figure 4.4: *Editing User Interface*

Compared with the other 3 user interfaces we mentioned in Chapter 2.4, our UI provides more ways of editing the shading and highlighting result and coloring control to make the editing more flexible. Table 4.1 shows the comparison between ours and the other 3 editing user interfaces:

<u><i>Feature/Method</i></u>	<u><i>Anjyo</i></u> <u><i>2003</i></u>	<u><i>Anjyo</i></u> <u><i>2006</i></u>	<u><i>Shimotori</i></u> <u><i>2007</i></u>	<u><i>Ours</i></u>
<u><i>Highlighting Vector</i></u> <u><i>Control</i></u>	<u><i>Y</i></u>	<u><i>Y</i></u>	<u><i>Y</i></u>	<u><i>Y (3 ways)</i></u>
<u><i>Shading Vector Control</i></u>	<u><i>N</i></u>	<u><i>Y</i></u>	<u><i>Y</i></u>	<u><i>Y (3 ways)</i></u>
<u><i>Light Source Control</i></u>	<u><i>N</i></u>	<u><i>Y</i></u>	<u><i>Y</i></u>	<u><i>Y</i></u>
<u><i>Shading/Highlight Color</i></u> <u><i>Control</i></u>	<u><i>N</i></u>	<u><i>N</i></u>	<u><i>N</i></u>	<u><i>Y</i></u>

Table 4.1: Comparison between ours and the other 3 editing UI

4.2.1 Texture Color Editing

Since texture color is distributed to the shading color accordingly, sometimes it is easier to modify the color on the texture instead of modifying the shading results directly. As the example shown in Figure 4.4, if the user want to change the color of the shading boundary or the highlight color, modify the texture is much easier than modify the shading result.

Our interface allows artists to put the color onto the texture directly. When the mouse is pressed and moved on the texture, the corresponding pixels (a cluster of pixels) are filled by a chosen color. The actual toon shading texture is changed as a result of the drawing. After filling the color, the shading result will be changed based on the updated texture. This operation is suitable for boundary color editing.

Figure 4.5: *Texture Color Editing*

The Toon Shading Sphere will also reflect the changes, since the sphere is using the same texture on the model.

4.2.2 Editing Obtained Result Directly

To provide the artist a straightforward way to edit the obtained results rendered by our shading pipeline, we also design this user interface to allow the artist to change the shading results with the edited color information reflected back to the texture and the Sphere model (Figure 4.6).

Figure 4.6: *Editing Obtained Result Directly*

As discussed in the previous section, the texture coordinates t and s are corresponding to the dot product of the eye vector and the lighting vector with the normal vector respectively. Therefore, when a user draws the model, for each affected vertex, the corresponding texture coordinates can be computed, and then the texture will be updated according to the coordinates. As a result, the new textured result will be shown on the model and the Toon Shading Sphere.

In a similar way, whatever artists edit on the Toon Shading Sphere model will be reflected to the texture and the actual rendering results in real time. This Shading Sphere is designed to give the user an overall view of the texture color distribution.

4.2.3 Light Source Controls

To allow the user to modify the shading result more freely, several light source controls are also designed. For a single light, 3 type of lights can be used: directional light, point light and spot light. The user can also use multiple light sources.

Lighting Adjustment:

- For directional light: change the direction of the lighting vector.
- For point light and spot light, change the position of the light source.

To change the light source direction and position, the rho-theta-phi user interface, which is shown in Figure 4.7, is used.

Figure 4.7: *rho-theta-phi light control UI*

The user can drag to change the position of the red point. According to the coordinates of the red point, the rho, theta, and phi value can be computed as shown in Figure 4.8:

Figure 4.8: *rho-theta-phi value computation*

The actual light vector (x, y, z) is then computed based on the rho, theta and phi value as follow:

$$x = \text{rho} * \cos(\text{theta}),$$

$$y = \text{rho} * \sin(\text{theta}),$$

$$z = \text{rho} * \sin(\text{phi})$$

rho in this case is always one when used to change the direction of the light.

Spot light rendering with cut off adjustment:

- For spot light rendering, in the Toon shading shader: define the vector from the origin to the light position as $V1$ and the vector from each vertex to the light position as $V2$. If the angle between $V1$ and $V2$ is smaller than the cutoff angle of the spot light, the texture coordinate s will be calculated, or s will be set to 0.

Virtual Multiple lighting sources:

- For multiple directional lights: each light will be assigned a weight, and the average among all the lights based on the weights will be utilized in the shader to obtain the rendering results.
- For multiple spot lights: for each spot light, a texture coordinates s will be calculated, and the largest s will be utilized as the actual texture coordinates to render.

4.2.4 3D view of the model

For comparison purpose, besides the toon shading rendering, we also provide the normal 3D rendering with phong shading lighting model. All the rendering are implemented using opengl shader language (GLSL).

Figure 4.9 shows the comparison between the phong shading and toon shading results.

Figure 4.9: *phong shading and toon shading comparision result*

(a) phong shading, (b)toon shading

4.2.5 Vectorizing the Obtained Result

To edit the obtained result globally, we apply the vectorization method in [Chen et al. 2005] to the obtained result. First we need to extract contour and boundary lines from the obtained result by applying Laplacian Filter. Then, we apply the vectorization method to the extracted lines. The method can assign the strokes onto the extracted lines. Then artists can edit the strokes to change the color regions globally. Finally, the desired colors are filled in the regions. Artists can thus reflect their taste and preference. Figure 4.10 shows the procedure for this editing.

Figure 4.10: *Editing Vectorized Result*

CHAPTER FIVE: RESULTS AND ANALYSIS

5.1 Surface Reconstruction

In this section, comparisons are prepared to demonstrate the differences between our proposed models and closely related works such as LUMO [Johnston 2002] and ShapePalettes [Wu et al. 2007].

Figure 5.1: *Simple shape example*

In figure 5.1, a simple shape is drawn with three stokes, and vectors are assigned automatically with the direction orthogonal to stroke tangents. The results from different algorithms are illustrated in Figure 5.3 with vector maps and reconstructed surfaces. Clear differences are shown in the results from the different algorithms at the same iteration number. As shown in Figure 5.3, each algorithm generates a convergent vector field after 500 iterations. In the convergence results, our algorithm preserves the main surface features such as the streamlined ridges. However, these ridges are smoothed in results generated by the other two algorithms. The same smoothing effect can also be noticed in Figure 5.4. In Figure 5.4-(f) and (g), the

ripples on the clothes are smoothed out. However, our algorithm faithfully reflects the assigned vector information on strokes, and reconstructs the ripples clearly.

Figure 5.2, the dokkaebi example shows another example created using our proposed algorithm using only regular strokes. All the initial vectors are simply set to be orthogonal to the stroke tangent directions and with same magnitude. As shown in 5.2 (e), our system can model this kind of relief effect very well.

Figure 5.2: *Dokkaebi example: (a) original image in Siggraph Asia 2010 website. (b) drawing traced by an artist. (c) vector setting, all vectors are orthogonal to the stroke tangent directions and with the same magnitude. (d) dense vectors obtained by the proposed nonlinear interpolation from the sparse vectors in (c). (e) reconstructed surface.*

Figure 5.3: *Comparison with related works*

Another test is applied to show intuitiveness of the modeling using our approach. We asked an artist to trace the examples used in [Ng et al. 2009] based on her drawing habits, and set several strokes as valley, bump and dip strokes. Figure 5.5 shows the modeling intuitiveness using our method. Using fewer strokes, our method can generate similar quality of surfaces as reconstructed in [Ng et al. 2009] and these strokes are in accordance to the drawing style of artists. In practical modeling process, artists draw strokes in their own styles. They may spend a little time to set stroke types or modify vectors to create more desirable surfaces if necessary.

Figure 5.4: Indian lady example: (a) line drawing. (e) is the assigned vectors on the strokes in (a). (b) and (f) dense vector field and reconstructed surface using the LUMO method, respectively. (c) and (g) dense vector field and reconstructed surface using the ShapePalettes method, respectively. (d) and (h) dense vector field and reconstructed surface using our proposed method, respectively.

Figure 5.5 *Female & male examples*

Figure 5.6 also shows the difference between our method and the related works. Our method can reconstruct surface details more properly than other two methods.

Figure 5.6: *Cat example: Comparison among LUMO, ShapePalettes, and our proposed method*

Our method can also handle a simple animation by changing assigned vectors gradually. Figure 5.7 demonstrates several frames selected from a fisting animation sequence. By changing the magnitudes of the initial vectors on the fist and reconstructing the surface, the animation is created. More complicated animation can be created by changing the magnitudes or directions of vectors.

Figure 5.7: *Fisting animation Sequence*

Table 3.1 lists the iteration numbers and computational time for the examples in this paper. Using augmented Lagrangian method and implementing our proposed algorithm using CUDA, the computational cost is quite low, which verifies the proposed method is applicable for practical modeling tasks.

5.2 Shading Pipeline

In this section, we demonstrate the results created through our shading pipeline. Since shading for hair, which is very hard to obtain by appropriate shading results of previous works, is one of the most important elements to represent anime-like feature, we start with the hair shading result, which is using the skeleton of the hair as reference.

Figure 5.8: *Hair Shading Result (a) Hairstyle outlines with the detected skeleton (b) Assigned initial vectors on the hairstyle outlines and skeletons (c) Result with skeletons, (d) Result without skeletons.*

As shown in Figure 5.7, anime-like shading has the feature of sharp edges especially for hair shade. To obtain such sharp edge shading effect, the skeleton of each hair strand is automatically detected based on both the side outlines of the hair strand, and tangent vectors are assigned along the detected skeleton accordingly. Figure 5.7 (a) and Figure 5.7 (b) show the skeletons and vectors which are generated automatically. Based on the vector setting, anime-like shading effect is achieved as depicted in Figure 5.7 (c) with sharp shade edges. The result without the skeletons under the same lighting environment is shown in Figure 5.7 (c) for comparison. These images show our initial vector assignment successful creates the sharp edges on the hairs. Although we give an option to the artists to assign direction and amplitude to the initial vectors, our automatic initial vector assignment has an enough quality to be utilized for shading.

Figure 5.9: Review of our shading pipeline

(a) Input hand-drawing, (b) assigned initial vectors, (c) Processed Shading Result, (d)

Minor Editing applied to (c)

Figure 5.8 shows the entire pipeline for our shading. First our method requires an artist's hand-drawn image as input. Next, the (semi-)automatic initial vector assignment is applied onto the drawn strokes. To obtain the surface normals, the vector in-painting is applied to the surface of the drawn character in each layer based on the initial vector sets. To achieve the anime-like shading result, we render the image by using toon-shading algorithm. The result we obtained at this point already is satisfactory, since it is already very similar to the shading result done by artists. However, since our method aims to develop for use by artists, we provide them with intuitive editing operations. Figure 5.8 (d) shows the result after editing by an artist. Only some small changes are needed to be applied to our result. As seen in this figure our shading pipeline and editing system have successfully created an anime-like shading result with little artists' interaction.

Figure 5.10: *Various Results reflecting artists' preference*

(a) Processed Shading Result, (b) Artist's work 1, (c) Artist's work 2

Figure 5.9 shows the result created by two artists. They created these results in less than a few minutes. Giving them the freedom, one created mild edge on the shaded region (Figure 5.9 (b)). The other created a sharper edge shade result (Figure 5.9 (c)). He also changed the hair color to reflect his taste. They comment that our processed shading result is interesting, and could be very useful for speeding up their shading and highlighting process. In normal cases, depending on the character complexity, it may take artists more than half an hour to finish the shading and highlighting of such a character. With the help of our pipeline and technique, they can do this within a few minutes. This shows that our pipeline can support artists to reflect their taste and preference easily, which is highly desired by artists, and can save a lot of time for them.

CHAPTER SIX: CONCLUSION AND FUTURE WORK

In conclusion, our method can reconstruct surface faithfully based on artists' hand-drawn work, and provide anime-like shading results for artists. Compare to the traditional animation production, in which all shading and highlighting need to be done by artists' sense, our method can save a lot time for them.

From sparse 2D strokes and vector field on the strokes, we presented surface reconstruction based on a two-step method. In the first step, we proposed a nonlinear vector interpolation combining TV regularization and H^1 regularization with the curl-free constraint in order to obtain a dense vector field from a given sparse vector field. In the second step, we propose a height map reconstruction algorithm which integrates the dense vector field in the first step. The curl-free constraint enforces the sparse vectors to interpolate on the large domain and TV regularization allows jump discontinuities preservation in the reconstructed surface and discontinuities of its gradient without surface distortion. We also provided different types of strokes to generate geometrically crucial shapes such as ridge, valley, jump, bump, and dip on a surface. These stroke types help users to create a surface which they intuitively imagine from 2D strokes. Comparing with other methods, the proposed model faithfully reflects the assigned vector information on strokes into the reconstructed surface. We can obtain fast numerical results using augmented Lagrangian method and local editing.

Figures 5.8 and 5.9 show that our shading pipeline can create anime-like rendering result successfully. The advantage of our method is that we can directly utilize the hand-drawn static image as an input. In fact, we neither have to create a 3D character model nor adjust the input image. Thus artists do not have to change their current production pipeline. Artists are usually comfortable with their way of doing things. This also is an advantage of our framework.

However, there are several future enhancements. Even though we have achieved sharp edge shading result based on our vector assignment approach, sometimes the sharp edges do not appear appropriately. To obtain such edges, we need to propose more robust algorithm to in-paint such a vector field or sophisticated initial vector assignment approach.

Furthermore, although artists who have tried our pipeline enjoyed reflecting their ways to create shade, our pipeline can currently only handle static images. Since in-between creation is one of the essential elements of animation and could be replaced by computer-assisted method, our pipeline needs to create smooth transition for in-between shading while retaining anime-like shading feature. This is quite difficult because anime-like shading result may not follow the laws of physics in many cases. To solve the problem, we have to consider a time-dimensional vector in-painting algorithm.

Appendix

A-1: Glossary:

Terms	Definition
Surface Reconstruction	Construct the surface information (normal, height) to build a realistic 3D surface based on 2D drawings.
Regular Stroke	Strokes used to reflects the rate of changes of the surface.
Ridge Stroke	Strokes used to reflects sharp ridges in the surface.
Jump Stroke	Strokes used to reflects sharp valleys in the surface.
Bump Stroke	Stroke used to create bump shapes on the surface.
Dip Stroke	Stroke used to create dip shapes on the surface.

A-2: Vector Setting:

Based on the observation that humans are good at assigning local surface normals for specifying local shape [Koenderink 1998], sparse user-input normals have usually been used in visible-surface representation. Recently, the authors [Wu et al. 2007] used ShapePalettes to easily assign 3D normals on the strokes. It is a very intuitive system if we have a suitable 3D reference model which reasonably corresponds to a desired shape. If we start to assign the vectors on the strokes without 3D reference models, it is still difficult tasks of assigning proper vectors to obtain a desired surface.

In this thesis, we interpret the assigned vectors as the projection of 3D surface normals on desired height map $z = I(x, y)$ into a 2D plane. That is, the assigned vectors $\mathbf{n}$ on all strokes except jump strokes are considered as $-\nabla I$ and the direction of surface normal vector is:

$$(-\partial_x I, -\partial_y I, 1).$$

Note that the assigned vectors on jump strokes are not regarded as the projection of 3D surface normals. The direction of vector $\mathbf{n}^*$ indicates the height of surface I is decreasing along the direction and a magnitude of vector is used to control the rate of surface height. If the magnitude of the vector becomes larger, the change of the surface height becomes steeper. For the simplicity of presenting vectors on strokes, we use the HSV color map with $V = 1$ in Figure A-1-(a). A default vector setting on strokes is orthogonal to stroke tangents with a fixed magnitude as shown in Figure A-1-(b). For regular strokes, users can change the direction and magnitude of vectors flexibly, for example in Figure A-1-(c). The magnitudes for vectors near the T-junctions along the visible part of the occluded contour are automatically reduced to avoid affecting the vectors specified for the contour occluding it, as shown in Figure A-1-(d).

Figure A-1: *HSV vector representation scheme*

Figure A-2: *The changes of the magnitude or direction of vector settings affect the shape of surfaces: (a) Default setting with vectors orthogonal to the stroke tangents. (b) Increasing the magnitudes for vectors on strokes representing the nose makes the nose higher. (c) Rotating the vector directions on strokes representing ears twists the face.*

Figure A-2 shows visible effects on reconstructed surface by assigning different vectors on regular strokes. Increasing vector magnitudes on strokes representing the nose from (a) creates a higher nose in (b), and rotating the directions of vectors in (c) anti-clockwise for 15 degrees creates twisted face.

The ridge stroke or valley strokes is composed of adjacent parallel regular strokes with opposite vector directions. The direction of vectors on two regular strokes is repulsive for sharp ridge structure or attractive for sharp valley structure. It is the same method suggested in ShapePalettes [Wu et al. 2007]. The default vector directions on ridge strokes and valley strokes are same as regular strokes.

Jump, bump, and dip strokes are used to generate more exquisite structures on the top of the geometry created by regular strokes and ridge or valley strokes. They are composed of base stroke and strokes of interest (SOI). Jump stroke has two SOIs which are parallel and adjacent to its base stroke in Figure A-2-(a). The magnitude of vector on base stroke indicates the size of jump and the surface height on the base stroke is suddenly jumped down along the direction of vector. The vectors on SOIs are chosen as same as the projection of given surface normals at the position of SOIs.

Bump stroke or dip stroke have one SOI which is parallel and adjacent to its base stroke in Figures A-2-(b) and A-2-(c). The vectors on SOI are chosen as same as the projection of given surface normals at the position of SOI. The magnitude and direction of vectors on base stroke have same meaning as regular strokes. Both ridge or valley and bump or dip are discontinuities of surface gradient. However, the difference is that bump stroke or dip stroke use the projection of 3D surface normal on SOI which is generated by regular, ridge, or valley strokes.

Figure A-2: Base stroke and strokes of interest (SOI) on jump, bump, and dip strokes: The black, blue and red arrows indicate the normal directions for the regular stroke, surface, and base stroke respectively. The first, second, and third rows are conceptual drawings. The first and third row show the surfaces without and with jump, bump, and dip strokes respectively, and the second row shows the change of the surface normals on the base stroke. The fourth row shows the reconstructed surfaces. Note that the geometry on the SOIs are preserved.

A-3: Proposed Models:

We present a reconstruction of height function $z = I(x, y)$ on a computational domain $\Omega \subset \mathbf{R}^2$ from sparse strokes $\Gamma \subset \Omega$ and a piecewise smooth vector field $\mathbf{n}^*$ on Γ . From the proposed nonlinear vector interpolation in Section 3.1, we obtain the dense normal vector $\mathbf{n}$ on Ω . After that, we find a surface I whose gradient field fits the vector field $\mathbf{n}$ in Section 3.2. The whole procedure can be applied on local domain to change details of surface.

Nonlinear Vector Interpolation:

First of all, we review a theorem in vector calculus. If the domain $\Omega \subset \mathbf{R}^2$ is simply connected,

$$\nabla \times \mathbf{n} \equiv \partial_2 n_1 - \partial_1 n_2 = 0 \quad \Leftrightarrow \quad \exists I \text{ s.t. } \nabla I = \mathbf{n}.$$

Since we reconstruct a height map I whose gradient fits the interpolated vectors $\mathbf{n}$ from $\mathbf{n}^*$, the curl-free condition on the vector field $\mathbf{n}$ should be guaranteed. Note that the interpolated vector field $\mathbf{n}$ violates the curl-free condition on known jump discontinuities in surface and its gradient.

Now, we propose an energy minimization model with the curl-free constraint for nonlinear vector interpolation of a given vector field $\mathbf{n}^*$ on the strokes Γ ;

$$\min_{\nabla \times \mathbf{n} = 0} \int_{\Omega} (1 - g) |\nabla \mathbf{n}| + \int_{\Omega} g |\nabla \mathbf{n}|^2 + \eta \int_{\Gamma} |\mathbf{n} - \mathbf{n}^*|, \quad (1)$$

where η is a positive constant and g is an indicator function. Note that the integration domain Γ can include closed and open strokes. The curl-free constraint on the interpolated vector $\mathbf{n}$ is not only a necessary condition to find a surface I with $\nabla I = \mathbf{n}$ but also gives extra force to interpolate sparse vectors over the large domain. The latter property is observed in image inpainting in [Tai et al. 2006].

The reason why we combine TV regularization and H1 regularization in the proposed functional is that they generate different geometry in reconstructed surfaces. The TV norm preserves discontinuities in an interpolated vector field while the H1 norm cannot preserve them. That is, discontinuities in an assigned vector field on the strokes are well propagated into the domain under the TV regularization, but they are smeared out into the domain under the H1 regularization. If there is an extra information to indicate where such discontinuities are located in Ω , the model [Ng et al. 2009] also preserves the discontinuities. However, it is a very difficult task to find possible locations of discontinuities just from the vectors on strokes. The TV regularization does not need the extra information to preserve discontinuities in the vector field. Considering the difference between TV regularization and H1 regularization, it is reasonable to set the indicator function g as a small value around jump, ridge, valley, bump, and dip strokes and it is automatically done because the types of strokes are the prior information. More-over, if an artist may want to generate a smooth surface on a specific region, it is simply done with setting g close to 1 on the region.

Figure A-3: *The role of indicator function g in (1): (a) is regular strokes and (b) is the vector setting on the strokes. Direction of assigned vectors on the cross shape in the rectangle is parallel to stroke tangent. In (c), g is 0 on the red region and 1 elsewhere. (f) shows a combination of (d) and (e) according to the indicator function g in (c).*

In Figure A-3, we show clear difference between TV regularization and H^1 regularization. (a) is regular strokes and (b) is the vector setting on the strokes. Direction of assigned vectors on the cross shape in the rectangle is parallel to stroke tangent. We have four discontinuities in the vector field at the four corners of the outer rectangle and there are three more discontinuities on the cross shape in the rectangle. They are easily shown as sudden color changes on strokes. (d) is the result of TV regularization ($g = 0$ in (1)). As we expect, the discontinuities in the

interpolated vector field are well propagated and it is reflected as sharp ridges in the reconstructed surface. (e) is the result of H^1 regularization ($g = 1$ in (1)) which is a smooth surface on the interpolated regions. After we set g as 0 on the red region and 1 elsewhere in (c), we obtain a reasonable combination of (d) and (e) as shown in (f).

When artists want to create a surface which has discontinuities in its gradient, they simply need to select regions to set the function g . It is much simpler task than indicating where discontinuities are located in the domain, which is used in [Ng et al. 2009]. In Figure A-4, an example drawn by an artist is shown. Only regular strokes in Figure 3 are used and vectors on the strokes are default setting. We show results with $g = 1$ and $g = 0$ in (a) and (c), respectively. Two surfaces are naturally combined in (b) by simply setting $g = 1$ on the left half and $g = 0$ on the right half.

Figure A-4: Comparison of TV regularization and H^1 regularization: We use regular strokes shown in Figure 3 and default setting for assigning vectors. (a) is a result with H^1 regularization and (c) is a result with TV regularization. (b) is a result with $g = 1$ on the left half and $g = 0$ on the right half in (1)

A procedure of using different strokes is as follows. First of all, we draw a rough shape of desired surface by regular strokes. Ridge or valley strokes can also be used. These strokes and the default setting of assigned vectors on the strokes generate a surface and we modulate the magnitude or direction of vectors in order to modify the surface. From many examples in Section 3.3, we are able to obtain desired surfaces which are intuitively imagined by only drawing regular strokes. In order to reconstruct more detailed structure, jump, bump, or dip strokes are provided. The vectors on SOIs should be assigned as the projection of the given 3D surface normal generated by regular, ridge, or valley strokes; see details in Figure 3.6. In the vector interpolation, Γ should include all SOIs and base strokes.

A-4: Height Map Reconstruction:

After the dense normal vector field $\mathbf{n}$ is obtained by the proposed nonlinear vector interpolation (1), the height map is reconstructed via the energy functional minimization;

$$\min_I \int_{\Omega} (1-h)|\nabla I - \mathbf{n}| + \int_{\Omega} h|\nabla I - \mathbf{n}|^2 + \xi \int_{\Sigma} |I - I_0|, \quad (2)$$

where I_0 is known height values on Σ and h is an indicator function. If there is no prior height information, we set $\xi = 0$.

The major difference between TV type norm $|\nabla I - \mathbf{n}|$ and H^1 type norm $|\nabla I - \mathbf{n}|^2$ is whether the jump discontinuities are allowed in the reconstructed height map or not. The TV type norm reconstructs the jump discontinuities without surface distortion. However, the H^1 type norm enforces the C^0 continuity of the height map because the

Euler-Lagrange equation from the second term in (2) is the Poisson equation. If we use the H^1 type norm on a whole domain, a reconstructed height map easily encounter overshooting or undershooting problem on the jump strokes. Considering the different effects from TV type norm and H^1 type norm, the value of the indicator function h should be very small around the jump strokes and 1 elsewhere. Since the jump strokes are prior information, the settings of h is automatically done.

A-5: Local Editing:

Local editing is necessary for an efficient and real time modification on the surface which is already reconstructed. Even though the computational cost is not very expensive on the domain size 512^2 , it is not a real time computation; see Table 3.1. We easily select new computational domain as the green region in Figure 3.9-(c). A bounding rectangle shown as the blue box is automatically found. We set Ω as the domain enclosed by the blue rectangle and Γ as the blue region, all strokes in the green region, and new strokes shown in purple color. Then, (1) and (2) are computed on new Ω and Γ . Note that we also can choose Σ as the green region and use I_0 as the height information on Σ . In our numerical experience, if the domain size is 64^2 , the whole procedure takes less than 1 second.

Figure 3.9: *Local editing: If some missing strokes in (a) are added, we do not need to compute the whole procedure. The local domain is selected as green and the local changes are reflected without changing the previous geometry. (b) and (d) are the reconstructed height maps from (a) and (c), respectively.*

A-6: Equations:

In order to efficiently solve energy functional minimization equations (1) and (2) in previous section, we briefly introduce the augmented Lagrangian method. First of all, we change (1) into a constraint minimization problem by introducing two variables $\mathbf{p}$ and $\mathbf{s}$:

$$\min_{\mathbf{n}} \int_{\Omega} (1 - g) |\mathbf{p}| + \int_{\Omega} g |\mathbf{p}|^2 + \eta \int_{\Gamma} |\mathbf{s} - \mathbf{n}^*|$$

with $\mathbf{p} = \nabla \mathbf{n}$, $\mathbf{s} = \mathbf{n}$, $\nabla \times \mathbf{n} = 0$.

Then, we use the augmented Lagrangian functional to solve the constraint minimization:

$$\begin{aligned}
\mathcal{L}(\mathbf{n}, \mathbf{p}, \mathbf{s}; \boldsymbol{\lambda}_r, \lambda_c, \boldsymbol{\lambda}_f) \equiv & \int_{\Omega} (1-g)|\mathbf{p}| + g|\mathbf{p}|^2 + \eta \int_{\Gamma} |\mathbf{s} - \mathbf{n}^*| \\
& + \int_{\Omega} \boldsymbol{\lambda}_r \cdot (\mathbf{p} - \nabla \mathbf{n}) + \frac{c_r}{2} \int_{\Omega} (\mathbf{p} - \nabla \mathbf{n})^2 \\
& + \int_{\Omega} \boldsymbol{\lambda}_f \cdot (\mathbf{s} - \mathbf{n}) + \frac{c_f}{2} \int_{\Omega} |\mathbf{s} - \mathbf{n}|^2 \\
& + \int_{\Omega} \omega_c \lambda_c (\nabla \times \mathbf{n}) + \frac{c_c}{2} \int_{\Omega} (\nabla \times \mathbf{n})^2,
\end{aligned} \tag{3}$$

where c_r , c_f , and c_c are positive penalty parameters, λ_r , λ_c , and λ_f are the Lagrangian multipliers, and ω_c is a weighting function. ω_c is adaptively chosen depending on the geometrical structure on given strokes. That is, it has very small value around ridge, valley, bump, dip, and jump strokes. We find the saddle point of the augmented Lagrangian functional (3) via maximizing the Lagrangian multipliers and minimizing variables $\mathbf{n}$, $\mathbf{p}$, $\mathbf{s}$. The minimization can be done by alternating minimization method. Since the Euler-Lagrangian equation for $\mathbf{n}$ is linear and there is a closed form solution for $\mathbf{p}$ and $\mathbf{s}$, the algorithm is fast and efficient.

Similarly, it is also possible to efficiently minimize (2) introducing the following augmented Lagrangian functional:

$$\begin{aligned}
\mathcal{L}(\mathbf{p}, I, J; \boldsymbol{\lambda}_r, \lambda_f) = & \int_{\Omega} (1-h)|\mathbf{p} - \mathbf{n}| + h|\mathbf{p} - \mathbf{n}|^2 + \xi \int_{\Sigma} |I - I_0| \\
& + \int_{\Omega} \boldsymbol{\lambda}_r \cdot (\mathbf{p} - \nabla I) + \frac{c_r}{2} \int_{\Omega} |\mathbf{p} - \nabla I|^2 \\
& + \int_{\Omega} \lambda_f (J - I) + \frac{c_f}{2} \int_{\Omega} (J - I)^2,
\end{aligned} \tag{4}$$

where c_r and c_f are positive penalty parameters and λ_r and λ_f are Lagrangian

multipliers. Note that c_r and c_f can be different values from (3). Detailed discretization and numerical algorithms are explained in [Hahn et al. 2010].

We use the same parameters for all examples in this paper and demo. $\eta = 100$, $c_r = 10$, $c_c = 1$, and $c_f = 1$ are used in (3). $\xi = 0$ and $c_r = 0.01$ are used in (4). Note that we do not need to use the extra variable J if $\xi = 0$, so c_f is not used. In order to stop the iteration, we use the relative L^1 error for each component in the Lagrangian multipliers. The error bounds for (3) and (4) are 10^{-2} and 10^{-3} , respectively.

References

AGRAWAL, A., RASKAR, R., AND CHELLAPPA, R. 2006. What is the Range of Surface Reconstructions from a Gradient Field? *Computer Vision C ECCV 2006*, pp. 578–591.

ANDRE, A., SAITO, S., AND NAKAJIMA, M. 2007. Crosssketch: freeform surface modeling with details. In *Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling (SBIM)*.

ANJYO K., ARIAS M., HORRY Y., MOMOSE Y.: Digital cel animation in Japan. In *Siggraph 2000 Conf. Abstracts and Applications* (2000), ACM Press, pp. 115–117.

ANJYO K., HIRAMITSU K.: Stylized highlights for cartoon rendering and animation. *IEEE Computer Graphics and Applications* 23, 4 (2003), 54–61.

ANJYO K., WEMLER S., BAXTER W.: Tweakable light and shade for cartoon animation. In *Proceedings of NPAR 2006* (2006), pp. 133–139.

BALLESTER, C., BERTALMIO, M., CASELLES, V., SAPIRO, G., AND VERDERA, J. 2001. Filling-in by joint interpolation of vector fields and gray levels. *IEEE Trans. Image Processing* 10, 8, 1200–1211.

BARLA P., THOLLOT J., MARKOSIAN L.: X-toon: An extended toon shader. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)* (2006), ACM.

BERTALMIO, M., SAPIRO, G., CASELLES, V., AND BALLESTER, 463 C. 2000. Image inpainting. In *Computer Graphics, SIGGRAPH 2000*, 417–424.

BERTALMIO, M., BERTOZZI, A. L., AND SAPIRO, G. 2001. Navier-Stokes, fluid dynamics, and image and video inpainting. In *Proc. Conf. Comp. Vision Pattern Rec.*, 355–362.

CHAMBOLLE, A., AND LIONS, P.-L. 1997. Image recovery via total variation minimization and related problems. *Numer. Math.* 76, 167 – 188.

CHAN, T. F., AND SHEN, J. 2005. Variational image inpainting. *Comm. Pure Applied Math.* 58, 579–619.

CHAN, T. F., KANG, S.-H., AND SHEN, J. 2002. Euler’s elastica and curvature based inpaintings. *SIAM J. Appl. Math.* 63, 2, 564–594.

CHEN Q., WU Z., TIAN F., SEAH H. S., QIU J., MELIKHOV K.: Vectorization of raster line-drawings in cartoons. In *Proceedings of International Conference on Computer Animation and Social Agents, CASA* (Oct. 2005), pp. 151–158.

- COLE, F., GOLOVINSKIY, A., LIMPAECHER, A., BARROS, H. S., FINKELSTEIN, A., FUNKHOUSER, T., AND RUSINKIEWICZ, S. 2008. Where do people draw lines? In *Proceedings of the 35th annual conference on Computer graphics and interactive techniques (Siggraph 2008)*.
- DECARLO, D., AND RUSINKIEWICZ, S. 2007. Highlight lines for conveying shape. In *Proceedings of NPAR 2007*, 63–70.
- DECARLO, D., FINKELSTEIN, A., RUSINKIEWICZ, S., AND SANTELLA, A. 2003. Suggestive Contours for Conveying Shape. *ACM Transactions on Graphics* 22, 3 (July), 848–855.
- SUGISAKI E, SEAH H. S, TIAN F, and MORISHIMA S. Interactive Shadowing for 2D Anime. *Computer Animation & Virtual Worlds*, 20(2-3):395-404, Jun 2009
- FRANKOT, R. T., CHELLAPPA, R., AND MEMBER, S. 1987. A Method for enforcing integrability in shape from shading algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 118–128.
- GINGOLD, Y., IGARASHI, T., AND ZORIN, D. 2009. Structured annotations for 2d-to-3d modeling. *ACM Transactions on Graphics (Siggraph Asia 2009)* 28, 5 (Dec), Article No. 148.
- HAHN, J., WU, C., AND TAI, X.-C. 2010. Augmented Lagrangian Method for Generalized TV-Stokes Models. TVCG submitted.
- HERTZMANN, A., AND SEITZ, S. M. 2005. Example-based photometric stereo: Shape reconstruction with general, varying brdfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27, 1254 – 1264.
- HERTZMANN, A., AND ZORIN, D. 2000. Illustrating smooth surfaces. In *Proceedings of the 27th annual conference on Computer graphics and interactive technique*, 517–526.
- HS Seah, F Tian, and ZK Wu. A Computational Framework for 2D Animation. In *NICOGRAPH International Conference*, Seoul, Korea, 2-3 Jun 2006.
- IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: a sketching interface for 3d freeform design. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques (Siggraph 1999)*, 409–416.
- QIU J, SEAH H. S., TIAN F, WU Z. K., and CHEN Q. Feature and Region Based Auto Painting for 2D Animation. *The Visual Computer*, 21(11):928-944, Oct 2005.
- JOHNSTON, S. F. 2002. Lumo: illumination for cel animation. In *NPAR '02: Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, ACM, New York, NY, USA, 45–52.

- JUDD, T., DURAND, F., AND ADELSON, E. 2007. Apparent Ridges for Line Drawing. *ACM Transactions on Graphics* 26, 3 (July), Article No. 19.
- KARPENKO, O. A., AND HUGHES, J. F. 2006. SmoothSketch: 3D free-form shapes from complex sketches. In *Proceedings of the 33th annual conference on Computer graphics and interactive techniques (Siggraph 2006)*, 589–598.
- KOENDERINK, J. J. 1998. Pictorial relief. *Philosophical Transactions: Mathematical, Physical and Engineering Sciences* 256, 1740 (May), 1071–1086.
- KRIKKE J.: Computer graphics advances the art of anime. *IEEE Computer Graphics and Applications* 26, 3 (2006), 14–19.
- LASSETER J.: Principle of traditional animation applied to 3d computer animation. *Computer Graphics* 21, 4 (1987), 35–44.
- NEALEN, A., IGARASHI, T., SORKINE, O., AND ALEXA, M. 2007. FiberMesh: designing freeform surfaces with 3D curves. *ACM Transactions on Graphics (Siggraph 2007)* 26, 3 (July), Article No. 41.
- NG, H.-S., WU, T.-P., AND TANG, C.-K. 2009. Surface-From-Gradients Without Discrete Integrability Enforcement: A Gaussian Kernel Approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
- OLSEN, L., SAMAVATI, F. F., SOUSA, M. C., AND JORGE, J. A. 2005. Sketch-based mesh augmentation. In *Proceedings of the 2nd eurographics workshop on sketch-based interfaces and modeling (SBIM)*.
- OLSEN, L., SAMAVATI, F. F., SOUSA, M. C., AND JORGE, J. A. 2009. Sketch-based modeling: A survey. *Computers & Graphics* 33, 1 (Feb), 88–103.
- PARADOS, E., AND FAUGERAS, O. 2006. *Shape from shading*. Springer.
- PETROVIC, N., COHEN, I., FREY, B. J., KOETTER, R., AND HUANG, T. S. 2001. Enforcing integrability for surface reconstruction algorithms using belief propagation in graphical models. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on* 1, 743.
- PRASAD, M., AND FITZGIBBON, A. 2006. Single view reconstruction of curved surfaces. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2*, 1345 – 1354.
- CHEN Q, TIAN F, SEAH H. S., WU Z. K., QIU J, and KONSTANTIN M. DBSC-based Animation Enhanced with Feature and Motion. *Computer Animation & Virtual Worlds*, 17(3-4):189-198, Jul 2006.

RUDIN, L. I., OSHEA R, S., AND FATEMI, E. 1992. Nonlinear total variation based noise removal algorithms. *Physica D* 60, 259–268.

SCHATZ, M.C., TRAPNELL, C., DELCHER, A.L., VARSHNEY, A. 2007. High-throughput Sequence Alignment using Graphics Processing Units. *BMC Bioinformatics*.

SIMCHONY, T., CHELLAPPA, R., AND SHAO, M. 1990. Direct analytical methods for solving poisson equations in computer vision problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12, 435–446.

SHIMOTORI Y., NAKAJIMA H., SUGISAKI E., MAEJIMA A., MORISHIMA S.: Interactive shade control for cartoon animation. In *ACM SIGGRAPH 2007 posters* (2007), p. Article No. 170.

TAI, X.-C., AND WU, C. 2009. Augmented Lagrangian method, dual methods and split Bregman iteration for ROF model. In *SSVM '09: Proceedings of the Second International Conference on Scale Space and Variational Methods in Computer Vision*, Springer-Verlag, Berlin, Heidelberg, 502–513.

TAI, X.-C., OSHER, S., AND HOLM, R. 2006. Image inpainting using TV-Stokes equation. In *Image Processing Based on Partial Differential Equations*, Springer, Heidelberg, 3–22.

TERZOPOULOS, D. 1988. The computation of visible-surface representations. *IEEE Trans. Pattern Anal. Mach. Intell.* 10, 4, 417–438.

WOODHAM, R. J. 1989. Photometric method for determining surface orientation from multiple images. *Shape from shading* 27, 513–531.

WU, T.-P., AND TANG, C.-K. 2006. Dense photometric stereo by expectation maximization. *Computer Vision - ECCV 2006* 3954/2006, 159–172.

WU, T.-P., AND TANG, C.-K. 2006. Visible surface reconstruction from normals with discontinuity consideration. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, Washington, DC, USA, 1793–1800.

WU, T.-P., TANG, C.-K., BROWN, M., AND SHUM, H.-Y. 2007. Shapepalettes: Interactive normal transfer via sketching. *ACM Transactions on Graphics* volume 26 Issue 3 07, Article No.44.

WU, T.-P., SUN, S., TANG, C.-K., AND SHUM, H.-Y. 2008. Interactive normal reconstruction from a single image. *ACM Transactions on Graphics* volume 27 Issue 5 08, Article No.119.

WU, C., ZHANG, J., AND TAI, X.-C. 2009. Augmented lagrangian method for total variation restoration with nonquadratic fidelity. Tech. rep., UCLA CAM Report 09-82.

ZELEZNIK, R. C., HERNDON, K. P., AND HUGHES, J. F. 1996. Sketch: an interface for sketching 3d scenes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 163–170.

ZHANG, R., TSAI, P.-S., CRYER, J. E., AND SHAH, M. 1999. Shape-from-shading: a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 21, 8 (Aug), 690–706.

ZHANG, L., DUGAS-PHOCION, G., AND SAMSON, J.-S. 2002. Single-view modeling of free-form scenes. *The Journal of Visualization and Computer Animation* 13, 225–235.