Artist-friendly Framework for Stylized Rendering

（アーティストによる陰影デザインのためのフレームワーク）

by

Hideki Todo

藤堂　英樹

A Doctor Thesis (Abstract)

博士論文（要約）

Submitted to

the Graduate School of the University of Tokyo

on September 27, 2013

in Partial Fulfillment of the Requirements

for the Degree of Doctor of Information Science and Technology

in Computer Science

Thesis Supervisor: Takeo Igarashi　五十嵐　健夫

Professor of Computer Science

ABSTRACT

In recent days, 3D computer graphics techniques are widely used in digital animation and video games for efficiently producing animation. Advances in stylized rendering techniques that can emulate hand-drawn stylized shading styles make 3D cartoon characters more common in digital animation films. However, these stylized rendering results are generated from physical lighting result according to predefined procedures. Providing efficient and intuitive interface for artists to design their expressive shading styles remains as a challenge.

In this thesis, we introduce a new framework, *integration of artistic depictions with physics-based lighting*, for designing artist-friendly shading model and interface. This framework is based on two principles: (1) directable shading model for artistic control and (2) seamless integration with 3D lighting. Based on the principles, we apply this framework to the following three different levels of shading design process, from small scale to large scale control.

First, we present *locally controllable shading with intuitive paint interface*. For directable control over shaded area, we propose a method to modify computed lighting term with a scalar offset function, obtained by painting process. Our approach enables appearance-based design for the desired changes to light and shade.

Second, we present *shading stylization based on model features*. This method allows interactive design for lighting enhancements based on model features, which would require time consuming painting process with the first method. Our system enables commonly used hand-drawn lighting effects, such as straight lighting effect on flat planes and edge emphasizing lighting effect on sharp edges.

Third, we present *practical shading model for expressive shading styles* for even larger scale control. In this method we focus on overall shading appearance while the first and second methods are limited to simple shading tones. The artist can design his shading style directly on a reference sphere. Our system then transfers the designed shading style to the target model based on 3D light and view settings.

Our framework enables interactively design of expressive stylized shading styles using compact and consistent representations. These successful results suggest the validity of our two principles for stylized shading. Finally, we discuss limitations and future research directions based on our finding in the thesis.

論文要旨

近年，3DCG は効率よくアニメーションを制作できるため，映像作品やゲームに幅広く利用されている．3DCG の陰影を手描き風に表現する技術も身近になり，手描きと 3DCG を組み合わせたアニメーション作品も数多く見られるようになった．しかし，既存の手描き風の陰影表現の技術では，物理計算された明るさ情報を直接機械的に手描き風の陰影に変換しており，アーティストが陰影を自在に制御するという点では課題が多く残っている．

そこで，我々は，アーティストが演出を行うための陰影の表現形式とインターフェースを設計する際の指針として「物理と演出を融合した手描き陰影表現のフレームワーク」を提案する．より詳細には，直観的かつ効率的な陰影のデザインを支援するため，(1) アーティストが演出可能な陰影モデルと (2) 既存のライティングとの親和性の双方を満たすような形で設計する．本論文ではこの設計指針に基づき，局所的制御から大域的制御まで異なる 3 つのレベルの特性に応じたデザイン手法を提案する．

第一に,「ペイントによる局所的な陰影制御法」を提案する．この手法では，局所制御による陰影の演出を実現するため，物理的に計算されたライティング結果をペイント情報に基づいて補正する，というアプローチを取った．直観的なペイント UI を提供することで，見た目ベースでの陰影のデザインを実現できる．

第二に,「形状の特徴表現のためのライティング強調手法」を提案する．この手法は，第一の手法では調整が難しい大域的な形状の特徴部分に対し，アーティストのライティング演出のデザインを支援するものである．手描きによく見られるような平坦さを強調する直線的なライティングや鋭さを強調する輪郭線付近のライティングを，インタラクティブにデザインすることができる．

第三に，さらに全体の見た目を調整する手法として,「手描風陰影のマテリアルデザイン手法」を提案する．この手法では，第一・第二の手法では調整することができない陰影全体の見た目に注目している．アーティストはガイドとなる球に手描き独特の陰影効果をペイントでデザインすることができ，デザインした陰影効果はライトの動きに合わせて 3 次元オブジェクト全体に反映される．

どのシステムにおいても，物理と演出の融合を意識し，既存のライティングとの親和性を実現している．提案したフレームワークを用いることで，アーティストの複雑な陰影表現を，コンパクトかつ整合性のある表現形式でインタラクティブに作成することができる．これらの結果は，我々が提案した物理と演出を融合したフレームワークの有効性を示唆している．また，本研究で得られた知見を基に，将来研究の方向性についても議論する．

Acknowledgements

I would like to thank everybody who has supported me in this work.

First of all, my deepest appreciation goes to my supervisor, Takeo Igarashi, for introducing me to the pleasure of user interface and computer graphics research. He always encouraged me to explore new findings with his creative way of thinking, precious advices, interesting ideas. Without his continuous support, I would never have completed this work. Besides my supervisor, my sincere thanks also goes to my thesis committee members: Shigeo Takahashi, Katsushi Ikeuchi, Akiko Aizawa, Shigeo Morishima, Yasushi Yamaguchi, for providing insightful comments essential for improving this thesis.

One of the most important research activities in my life was a work experience at OLM Digital, Inc. as an intern and employee. Most of the ideas in this thesis were advanced in this experience. I would like to express my deepest gratitude to Ken Anjyo, who was my advisor there. He has taught me how to focus on important things for future animation industry. Discussions with him have been illuminating ways for progress in good directions. He has also introduced me to many researchers who work in different fields, which gives me many interesting problems and important hints for solving the problems. I would like to thank William Baxter, who provided me with insightful comments and suggestions to complete my SIGGRAPH and CASA paper [90, 91]. It was also a valuable experience for me to have intense discussions with Pascal Barla, who is one of top researchers in Non-Photorealistic Rendering research field. During my work experience on CREST project, I was able to start new projects about facial animation [3, 89], which are unfortunately not included in this thesis. For these projects, I would like to thank J.P. Lewis and Jaewoo Seo, who provided inspirational, supportive feedbacks. I would also thank to CREST team members: Yoshinori Dobashi, Kei Iwasaki, Masato Wakayama, Hiroyuki Ochiai, Yoshihiro Mizoguchi, Shizuo Kaji, Shun'ichi Yokoyama. In particular, Shun'ichi Yokoyama offered many suggestions and comments as a collaborator to accomplish my CGI paper [92]. Special thanks also to other OLM members: Ayumi Kimura, Satoshi Mizubata, Satoru Yamagishi, Yosuke Katsura, Marc Salvati, Tatsuo Yotsukura, Miki Kinoshita, Yuki Ishii, Shinji Morohashi, Makoto Sato, Jun Toyoshima, Jun Kondo, Masashi Kobayashi, Yoshinori Moriizumi. Without their guidances and persistent helps, this thesis would not be possible.

I would also thank to lab members: Shigeru Owada, Kazutaka Kurihara, Makoto Okabe, Masatomo Kobayashi, Yasushi Maruyama, Kenji Hara, Takashi Ijiri, Takeshi Nishida, Yoshinori Kawasaki, Nayuko Watanabe, Hidehiko Abe, HyoJong Shin, Kaisuke Nakajima, Yuki Igarashi, Kenshi Takayama, Jun Kato. In particular, I would thank Makoto Okabe for continuing the stimulating discussions, encouragements even after my graduation. After I moved back to the University of Tokyo, I spent good time with new lab mates and ERATO members: Daisuke Skamoto, Makoto Nakajima, Yuki Koyama, Naoki Sasaki, Koumei Fukahori, Genki Furumi, Fangzhou Wang, Masaaki Miki, Chen Hsiang-Ting, Li-feng Zhu, Lasse Laursen, Daniel Rea, Morten Nobel-Jørgensen, Nobuyuki Umetani, Yutaro Hiraoka.

Finally, I would like to thank my family. To my parents, Tsuyoshi and Eiko, who has always provided me with devoted love, financial support, and endless encouragements. To my wife, Saori, who always believes in me and support whole my life.

Additional thanks go to OLM Digital, Inc., AIM@SHAPE Shape Repository, and Keenan's 3D Model Repository for the 3D models used in this thesis. This work was funded in part by grants from IPA (Information Technology Promotion Agency Japan), JSPS Research Fellowship, the Japan Science and Technology Agency, CREST project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent progress in computer graphics has led to many 3D rendering techniques that are widely used in digital animation and video games. In 3D computer graphics, character animations with illuminations are efficiently produced from pre-designed 3D scenes by physical simulations. Accordingly, researches of stylized rendering have focused on making use of 3D scenes to reproduce abstracted styles of artists. For example, Lake et al. [50] proposed a real-time rendering technique to produce the banded, multi-tone shading of traditional hand-drawn cartoons. In this technique, the continuous gradation of light in diffuse, specular lighting is converted to multi-tone colors through a simple 1D color mapping process (see Figure 1.1). This technique, widely known as a cartoon shading, is now available as the built-in feature of much commercial 3D software [8–10, 60]. Beside the simple cartoon shading, artists can use various stylized shading techniques [11, 36, 37, 50, 58, 87, 107]. As a result, 3D characters now commonly exhibit stylized shading [19, 59, 72, 102].

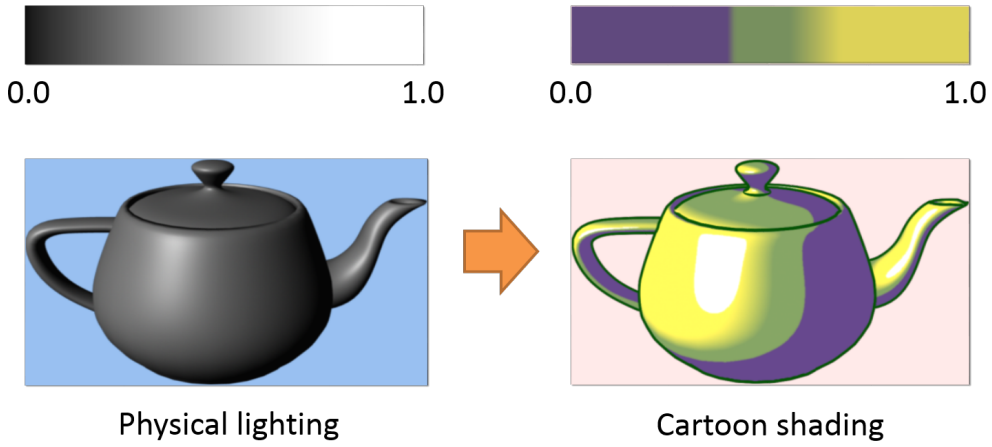


Figure 1.1: *Cartoon shading process. (Left) Physical lighting, showing gradation of light. The brightness values are computed from diffuse and specular reflectance models. (Right) Cartoon shading. The banded multi-tone appearance is obtained through simple 1D color mapping of the brightness values.*

However, conventional stylized shading techniques that produce rendering results as a simple conversion of a physical lighting model are insufficient for most artists. In the case of stylized shading applications such as digital cel animation, lights and shades often include artistic depictions to not only convey illumination or material, but also to emphasize character's mood or geometric feature. Such shading effects are more likely to be artificial, thus conventional shading approaches often result in undesirable

shading. The top images in Figure 1.2 show such an example in the case of cartoon shading, where the artist may want to add a shaded area below the right eye, as shown on the left image. In the second example (middle images), the artist may desire straight lighting with edge enhancement to show the flatness and sharp feature of the object. The bottom images show another example, where the artist may want small-scale stroke styles to have more expressive visual appearance. In all examples, the artist would like to have the directability to modify the rendered shading.

To modify such undesirable shading results, conventional tricks are often used in production work (see Figure 1.3). Additional lights would be a simple and efficient approach to design small local lighting effects. However, it is difficult to design artificial shading effects since this approach is strongly constrained by a physical lighting mechanism. This physical constraint can be relaxed by changing the geometry, but its indirect editing process requires additional trials and errors to obtain a desired result. The most flexible way for designing physically-incorrect shading effects would be animating textures, but it requires a lot of time consuming manual painting and key-framing tasks for artists. Despite the crucial demands for an artist-friendly control of stylized shading, it is difficult to handle them just using conventional tricks. In production environments, artists need both flexible and efficient way to support their creative process.

In the stylized rendering research fields, there are a few significant methods to support stylized shading design tasks of artists. Related to the first and second issues in Figure 1.2, several approaches provide the artist with highlight shape control [4,5,20,68,74]. However, their approaches are not sufficient for the shading case in the first issue, where the artist want to freely design an arbitrary shape that requires more integrations with original lighting than the highlight case. In addition, they cannot be used for shading stylizations in the second issue since their shape controls are applied to the overall lighting shapes. For the third issue, the multiple layered material design system [96] allows the artist to design complicated shading styles beyond simple cartoon shading styles. However, small-scale stroke styles as shown in Figure 1.2 cannot be designed using their system. The challenge remains to provide an efficient and intuitive interface for artists to design their own expressive shading styles.

1.1 Integration of Artistic Depictions with Physics-Based Lighting

The goal of this thesis is to establish efficient and effective stylized shading design methods for such practical demands in production work. As a first step toward a new methodology, we consider how to improve shading design processes to overcome the conventional shading issues shown in Figure 1.2. In contrast to previous researches, our shading design targets are difficult because of two requirements: more fine-grained controls over shading appearances and their 3D lighting interactions. First, artists want to design more detailed physically-incorrect shading effects (arbitrary lighting shapes, feature-dependent lighting effects, or small-scale stroke styles) beyond simple global light shape controls. Second, we need to provide suitable interactions between the physically-incorrect lighting effects and existing lighting controls to make use of the efficiency of 3D lighting mechanisms. To fulfill these two requirements, we introduce a new framework, *integration of artistic depictions with physics-based lighting*, for designing an artist-friendly stylized shading model and its interface. Figure 1.4 illustrates this framework, which consists of two principles:

Principle 1: Directable Shading Model for Artistic Control

Our first principle to meet directional demands is to introduce effective, compact shading models that let the artist modify the shading appearance with intuitive, interactive manners (**Principle 1**). In existing 3D systems, the artist needs to carefully control multiple elements at the same time: shapes, materials, cameras, and lights. These indirect controls make the shading process difficult. Thus, it is helpful to design a compact shading model that lets the artist modify the original shading using an intuitive, interactive design process. Its parameters and controls are designed to directly modify the shading appearance, thus each shading design process becomes more simple and flexible to get a desired shading result. For example, when we want to modify shaded areas, we can make an arbitrary shape by painting. In addition, shading stylizations with appearance-based parameters are also useful for emphasizing the specific model features such as surface flatness and edge feature. Our directable shading models aim to provide new intuitive shading design methodologies for stylized shading effects, which would be difficult to achieve using conventional light controls.

Principle 2: Seamless Integration with 3D Lighting

Our second principle to meet directional demands is to provide the directable shading models that fit into a existing 3D lighting process (**Principle 2**). In making 3D character animation, light and camera controls are essential for efficiently changing the lighting. To capitalize on these existing controls, we designed each directable shading model in a manner that can be affected by dynamic lighting. In addition, we also provide a Key-framing UI, which allows the artist to design desired animation in a convenient and familiar way. By following this principle, we can combine artistic depictions for expressive shading appearance and physics-based lighting for efficient rendering of the 3D scene.

1.2 Experimental Systems

To verify the effectiveness of our proposed stylized shading design framework, we present three shading design systems for different levels of shading design processes, from small scale to large scale controls.

Locally Controllable Shading with Intuitive Paint Interface. First, we present a 3D stylized shading system to add local light and shade using paint operations. The basic idea of this method is to modify the lighting term directly, adding a scalar offset function obtained from the painted area. The modified shading is consistent and seamlessly integrated with the original 3D lighting. Our system demonstrates how our method lets artists design light and shade locally as desired.

Shading Stylization Based on Model Features. Second, we present a 3D stylized lighting method that enhances models' features. Artists can create in 3D the same feature enhancements as are commonly used in 2D manual artworks: straight lighting on flat planes, edge enhancement on sharp edges, and detailed lighting for jagged shading. The central idea of this method is to use simple lighting transforms and offsets based on the model features. Our system demonstrates how our method is effective for designing shading stylizations over model features.

Practical Shading Model for Expressive Shading Styles. Third, we present a 3D stylized material design system for designing overall shading appearance with prominent features. Our system lets the artist paint his shading style on a reference sphere. The designed shading style is interactively transferred to the target model while the artist manipulates the light source. The basic idea is to introduce a new 2D texture projection process for expressive shading styles based on light space surface normals. We also explore practical shading stylization techniques by making use of the light space normal representation. Our system demonstrates how our method is useful for designing commonly used shading styles, such as minimal shading, illustrative shading, and stylized metallic, etc.

1.3 Contributions

Our goal is to provide an artist-friendly shading model and user interface for designing stylized shading effects which are effective for production work. The contributions of this work include the proposed framework for this goal and three experimental systems based on the framework.

New framework for an artist-friendly shading model and user interface. We present a new framework, called "integration of artistic depictions with physics-based lighting", as a general key guideline for efficient and effective stylized shading design. We propose two principles for this key guideline. **Principle 1** is directable shading model for artistic control, which allows the artist to interactively design shading appearance using intuitive user interfaces. **Principle 2** is seamless integration of the directable shading with 3D lighting, which enables dynamic controls of shading appearance using familiar 3D UIs. By following these principles, we can merge non-physical behavior of artistic depictions and physical behavior of 3D lighting, which makes the shading design process more flexible to make stylized character animation. In contrast to previous systems, our framework can handle more detailed non-physical lighting effects with suited 3D lighting interactions.

Three experimental shading design systems. Based on the proposed framework, we developed shading design systems for small scale local shaded areas, middle scale model features, large scale shading materials. The first system was developed to control local shaded areas, where we provided a paint brush user interface to modify shaded area. This system allows the artist to freely design arbitrary shapes of the target shaded area for small scale controls. The second system was developed to enhance model features such as surface normals and edges, where we provided a 3D light UI for straight lighting effects and appearance-based parameters for edge enhancement and detail lighting effects. This system allows the artist to design the feature-dependent lighting effects for middle scale controls. The third system was developed to design overall shading materials, where we introduced a new shading model to design an expressive shading style beyond simple cartoon shading styles. This system allows the artist to design light-dependent shading stylizations for large scale controls. All systems are carefully designed according to principles of our framework, which provides efficient and effective stylized shading design process for each design target.

1.4 Outline

This thesis is organized as follows. In Chapter 2, we review existing methods of stylized shading. After briefly describing the overview, we review the stylized shading methods used in three major areas of research: early stylized rendering, style extensions, and directable cartoon shading. The last topic features artistic controls that are closely related to our framework. We also briefly discuss other rendering techniques related to stylized shading design.

In Chapter 3, we describe our approach for the artist-friendly shading design framework. We first review and analyze a general cartoon shading process used in typical production work. We then consider appropriate representation of directable shading models in accordance with two proposed principles: directable shading model for artistic control (**Principle 1**) and seamless integration with 3D lighting (**Principle 2**).

In Chapters 4-6, we present three experimental systems for the different levels of shading design process: **locally controllable shading with intuitive paint interface** (small scale) in Chapter 4, **shading stylization based on model features** (middle scale) in Chapter 5, **practical shading model for expressive shading styles** (large scale) in Chapter 6.

In Chapter 7, we examine the capabilities of the three experimental systems. We first summarize the overall features of the three methods from the perspective of our framework, and then compare the directable mechanisms used in these systems.

Chapter 8 presents our conclusions. We summarize the contributions of the experimental systems, and then discuss the limitations of the framework. Finally, we discuss future research directions.

1.5 Publications

The work presented in this thesis is the result of collaborations and projects that have been published as follows:

- The system for *locally controllable shading with intuitive paint interface* described in Chapter 4 was presented as "Locally controllable stylized shading" [90] at ACM SIGGRAPH 2007 in San Diego, USA, in collaboration with Ken Anjyo and William Baxter from OLM Digital, Inc. and Takeo Igarashi from the University of Tokyo.
- The system for *shading stylization based on model features* described in Chapter 5 was presented as "Stylized lighting for cartoon shader" [91] at the 22nd Annual Conference on Computer Animation and Social Agents (CASA 2009) in Amsterdam, the Netherlands, in collaboration with Ken Anjyo from OLM Digital, Inc. and Takeo Igarashi from the University of Tokyo.
- The system for *practical shading model for expressive shading styles* described in Chapter 6 was presented as "Lit-Sphere extension for artistic rendering" [92] at Computer Graphics International (CGI 2013) in Hannover, Germany, in collaboration with Ken Anjyo from OLM Digital, Inc. and Shun'ichi Yokoyama from IMI, Kyushu University.

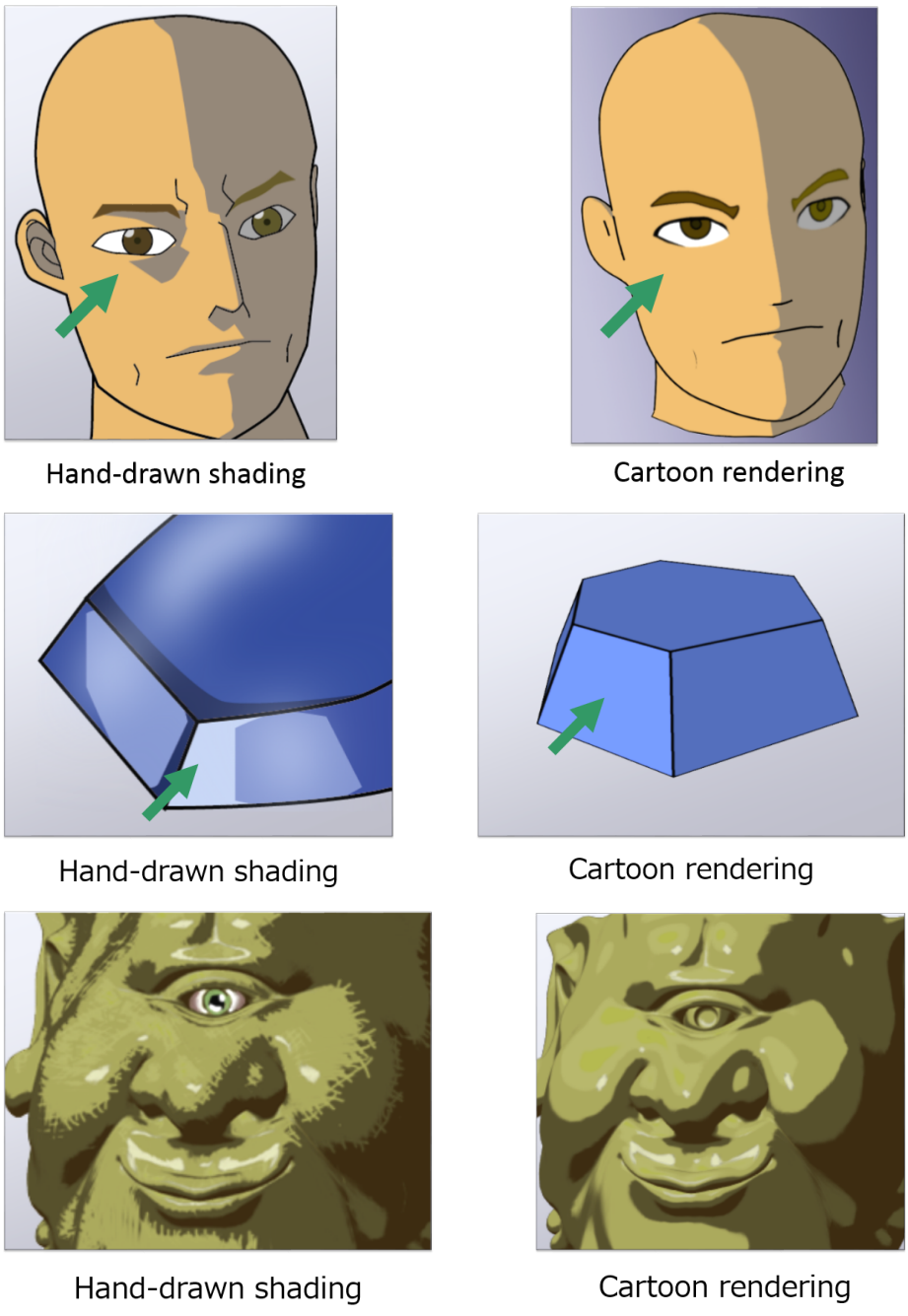


Figure 1.2: *Comparison of hand-drawn shading (left) with conventional cartoon shading (right). (Top) The cartoon shading fails to render the shaded area blow the right eye that emphasize the character's fierceness. (Middle) The cartoon shading fails to capture shading stylizations that enhances model's flatness and sharpness. (Bottom) The cartoon shading fails to represent small-scale stroke styles.*

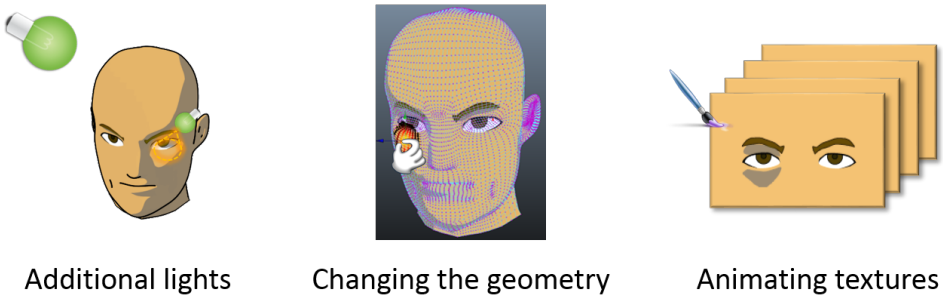


Figure 1.3: *Conventional tricks to modify undesirable shading result.*

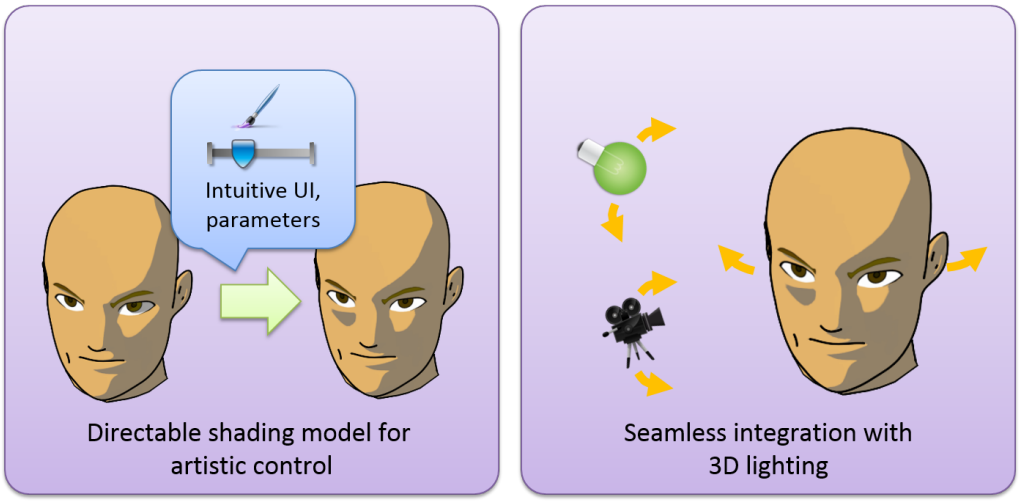


Figure 1.4: Integration of artistic depictions with physics-based lighting. *(Left) Artists can modify the original shading result using intuitive appearance-based UIs or parameters. (Right) Artists can manipulate the designed shading using existing 3D lighting and animation controls.*

Chapter 2

Related Work

In this chapter, we review existing methods for stylized rendering and discuss how they relate to our approach. Figure 2.1 shows the methods relevant to our work, the chapter sections in which they are discussed, and their classification according to two properties: directability (from less to highly directable) and expressiveness (from less to highly expressive). Highly directable methods focus on how to provide intuitive and interactive controls over shading appearance, whereas less directable methods permit limited controls using more automatic approaches. Highly expressive methods focus on how to achieve a rich variety of shading styles with prominent features, whereas less expressive methods are limited to simple shading styles such as cartoon shading. In Section 2.2, we review several fundamental methods for interactive 3D stylized rendering. In Section 2.3, these fundamental methods are extended to yield more expressive shading. There are several significant methods for directable control (Section 2.4 and Section 2.5), which are the main focus, which are the main focus of this thesis. These areas of research include our methods described in Chapters 4 and 5. We further explore how to establish both directability and expressiveness in Chapter 6. Finally, in Section 2.6 we briefly review two several other stylized rendering methods: painterly rendering and line drawing.

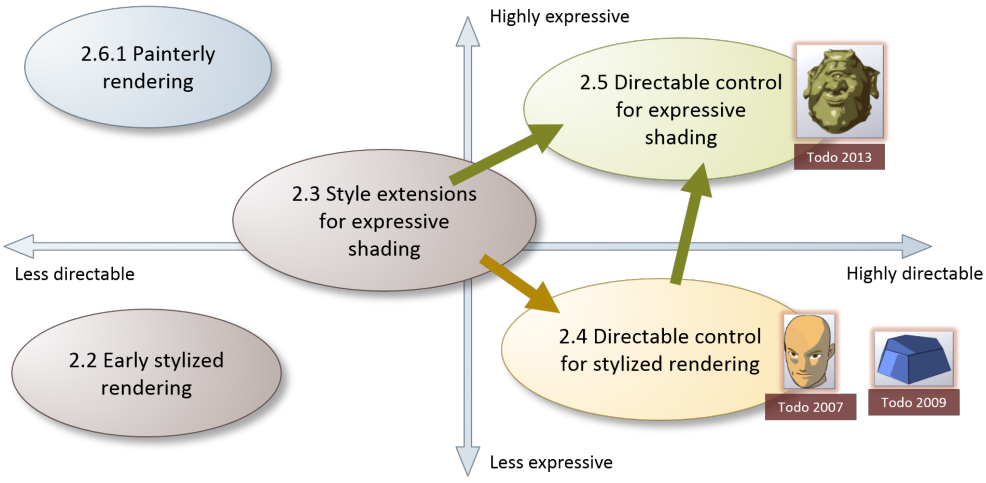


Figure 2.1: *Stylized rendering methods.*

2.1 Lighting Design for Photorealistic Scenes

Before describing the stylized rendering methods, we review the lighting design methods for photorealistic scenes, which form the foundations for the stylized rendering methods. For photorealistic appearance, simple reflectance models can be used: Lambert (diffuse), Phong (specular) [71], and Blinn-Phong (specular) [15]. For example, a lighting model for diffuse and specular effects can be defined as:

$$\mathbf{c} = \mathbf{c_d} I_d + \mathbf{c_s} I_s, \tag{2.1}$$

where diffuse term $I_d \in \mathbb{R}$ and specular term $I_s \in \mathbb{R}$ are obtained from the specific reflectance model. The final color $\mathbf{c}$ is adjusted by the diffuse color $\mathbf{c_d}$ and the specular color $\mathbf{c_s}$. Figure 2.2 illustrates a typical example using the Blinn-Phong lighting model. This model uses as inputs a light vector $\mathbf{L}$, a view vector $\mathbf{V}$, a surface normal vector $\mathbf{N}$, and the half vector $\mathbf{H} := (\mathbf{L}+\mathbf{V})/\|\mathbf{L}+\mathbf{V}\|$. The diffuse term I_d and specular term I_s are obtained by the dot products of $\mathbf{L}\cdot\mathbf{N}$ and $\mathbf{H}\cdot\mathbf{N}$ respectively.

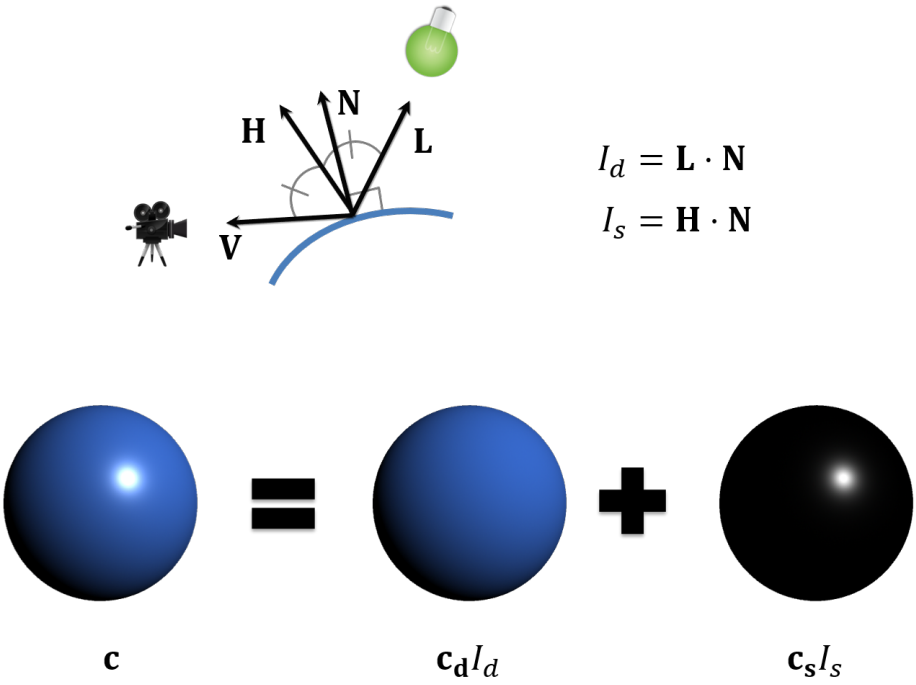


Figure 2.2: *Blinn-Phong lighting model. (Top) Vectors for computing Blinn-Phong lighting. The diffuse term I_d and specular term I_s are defined by dot products of these vectors. (Bottom) Visual illustration of the Binn-Phong equation.*

Beyond these simple reflectance models, more physically-plausible lighting effects can be modeled by using a bidirectional reflectance distribution function (BRDF) [6,7,23,55, 104] and related techniques: bidirectional scattering distribution function (BSDF) [38], and bidirectional surface scattering reflectance distribution function (BSSRDF) [32]. The fundamental difficulty in using these shading models is to finding the optimal light placement and the choice of parameters to obtain the desired shading. Several good approaches have attempted to measure the scattering profiles of physical materials [1, 26, 31,41,76,83,94,103,105]. Other approaches have tried to find the proper light placement from user-designated highlights and shadows in the scene [2, 22, 48, 51, 63, 69, 85, 88].

The advantage of BRDF related approaches is their ability to illuminate models with visual realism. Once the appropriate parameters are found for a specific material, it

can be successfully used for 3D animation. On the other hand, these approaches are a difficult for artists to use. Therefore, most stylized rendering approaches use simple lighting models for their fundamental mechanisms.

2.2 Early Stylized Rendering

2.2.1 Artistic Stylization for 2D Static Images

In the early stage of stylized rendering techniques, most of these shading representations were 2D static grayscale images, which are used to reproduce traditional artworks. In 1976, Floyd and Steinberg [35] proposed the fundamental idea of digital halftoning where the brightness values are converted into black and white pixels through thresholded quantization. Similar to this seminal work, 2D static grayscale brightness had been used for various printing artworks: stippling [29, 86], pen-and-ink illustration [30, 79–81], digital engraving [34, 66, 67], and woodcut illustration [57, 108].

In summary, their idea is to define a color map function $cm : \mathbb{R} \mapsto \mathbb{C}$ for the brightness value $I \in \mathbb{R}$, where $\mathbb{C}$ denotes a color space. They considered only the simple case of static 2D input of the brightness value I. Thus, they were limited in handling dynamic shading changes. In this thesis, we focus more on 3D rendering techniques, which provide the artist with interactive shading design for 3D character animation.

2.2.2 Stylized Rendering for 3D Scenes

In 3D rendering, stylized shading is based on the simple lighting models described in Equation 2.1. For example, The Technical Illustration Shader of Gooch and Gooch [36, 37] uses the Half-Lambertian diffuse term to produce cool-to-warm color gradients. One significant invention by Lake et al. [50] is an interactive cartoon shader where the diffuse term is converted into banded multi-tone colors through simple 1D color mapping. Mitchell et al. [58] modified the Lambertian and the Phong shading models for a customized illustrative look in their video game applications.

Figure 2.3: *Examples of typical stylized rendering methods for 3D scenes. (Left) Technical Illustration Shader developed by Gooch and Gooch [36,37] (©1999 ACM). (Middle) Interactive stylized rendering proposed by Lake et al. [50] (©2000 ACM). (Right) Illustrative rendering used by Mitchell et al. [58] (©2007 ACM).*

The primary advantage of these approaches is their simplicity: a single 1D color map function is sufficient to model the stylized shading. The final shading color $\mathbf{c} \in \mathbb{C}$ is obtained by:

$$\mathbf{c} = cm_d^{1D}(I_d) + cm_s^{1D}(I_s), \tag{2.2}$$

where the 1D color map functions $cm_d^{1D} : \mathbb{R} \mapsto \mathbb{C}$ and $cm_s^{1D} : \mathbb{R} \mapsto \mathbb{C}$ are applied to the diffuse term I_d and specular term I_s. This simple mechanism permits interactive shading design using a 3D lighting process, so it is widely used as a foundation for other stylized rendering methods. In the next section, we review existing methods for more expressive shading styles derived from the 1D color map approach.

2.3 Style Extensions for Expressive Shading

2.3.1 2D Color Map Functions

More complex effects can be obtained using 2D color map functions. Winnemöller and Bangay [107] introduced a 2D color map function to capture the stylistic behavior of specular effects:

$$\mathbf{c} = cm^{2D}(I_d, I_s), \tag{2.3}$$

where the 2D color map function $cm^{2D} : \mathbb{R}^2 \mapsto \mathbb{C}$ takes the two variables I_d and I_s. Barla et al. [11] generalized this idea for various shading stylizations such as level-of-detail, depth-of-field, and back-lighting. In their approach, the specular term I_s is replaced by a general attribute term $I_a \in \mathbb{R}$. Using these inputs, the final shading is controlled by a 2D texture, which stores the 2D color map function cm^{2D}. Figure 2.4 illustrates this application of a 2D color map to create a diffuse-dependent specular effect.

The Lit-Sphere model of Sloan et al. [87] takes another approach to the use of 2D texture: the 2D shading tones are based on the view-space surface normals (see Figure 2.5). For a given surface normal vector in view space $\mathbf{N_v} := (N_{vx}, N_{vy}, N_{vz})$, the Lit-Sphere shading model maps a color as:

$$\mathbf{c} = cm^{2D}(N_{vx}, N_{vy}), \tag{2.4}$$

where the 2D color map function (stored in a 2D texture) takes the components of the surface normal vector N_{vx} and N_{vy}. Sloan et al. demonstrated various examples of expressive 2D shading tones to reproduce typical shading styles of traditional artworks. This technique was extended to volume rendering with blended multiple Lit-Sphere shading [17].

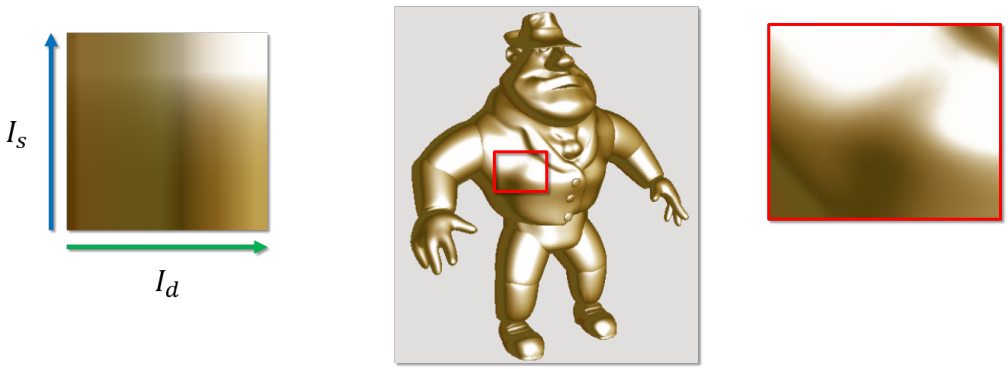


Figure 2.4: *Example of a 2D color map from X-Toon [11] (©2006 ACM). The 2D texture color is referenced by the diffuse term I_d and the specular term I_s. The highlight is present only when both the diffuse term and the specular term are high, which effectively emulates a metallic appearance.*

While these approaches provide additional functionalities for designing expressive shading styles, 2D textures are not suitable for dynamic control, which is crucial for creating

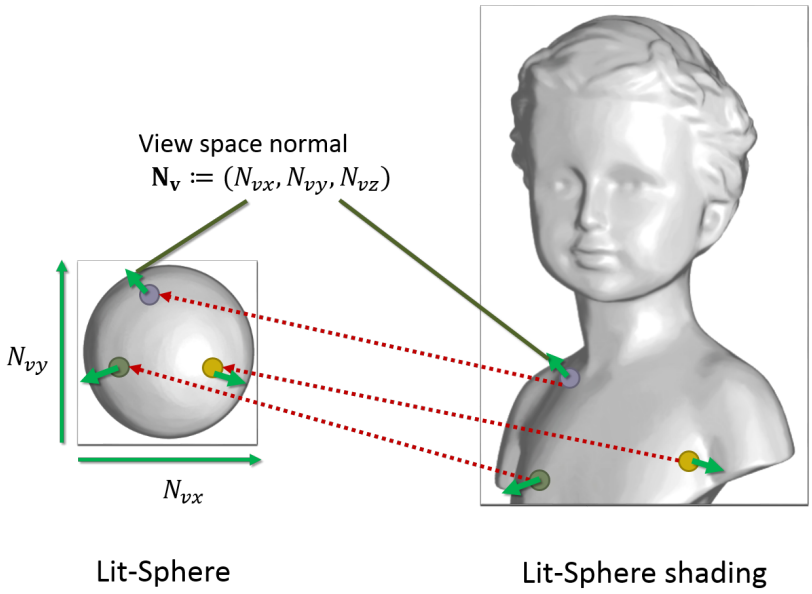


Figure 2.5: *Example of a 2D color map using Lit-Sphere [87]. The 2D texture color is referenced by the view space normal vector* $\mathbf{N_v} := (N_{vx}, N_{vy}, N_{vz})$. *This enables a view-dependent shading effect with effective 2D shading tones.*

animation. In contrast, all of our methods presented in Chapters 4- 6 permit dynamic control, which is seamlessly integrated with the familiar 3D shading design process.

2.3.2 Surface Feature Enhancement

Several approaches have used shape depiction to extend conventional stylized shading styles. In practical applications such as video games, ambient occlusion [18, 70] is widely used to add occluded shadow effects to diffuse shading. Exaggerate shading [75] uses multiple scale normals to show the bumpy details of an object (see the left image of Figure 2.6). The 3D Unsharp Masking technique of Ritschel et al. [73] modifies the outgoing radiance to enhance local contrast. Vergne et al. proposed methods to enhance shape depiction based on view-dependent geometric features [97, 98] (see the middle image of Figure 2.6). They also proposed radiance scaling techniques [99, 100] that are extensions of their previous methods for precomputed radiance data (see the right image of Figure 2.6).

In summary, these methods define a vector transform function $f_L : \mathbb{S}^2 \times \mathbb{G} \mapsto \mathbb{S}^2$ for the light vector $\mathbf{L} \in \mathbb{S}^2$ based on the geometric property $\mathbf{G} \in \mathbb{G}$, where $\mathbb{G}$ denotes the space of the geometric property. The methods focus on use of the geometric property $\mathbf{G}$ for providing better visual perception of geometric appearance. In contrast, our shading stylization method presented in Chapter 5 focus on appearance-based control, determined by model features.

2.4 Directable Control for Stylized Rendering

One important requirement of a shading design system is to provide the artist with directable control over the shading appearance. The cartoon highlights of [4, 5, 96] deal

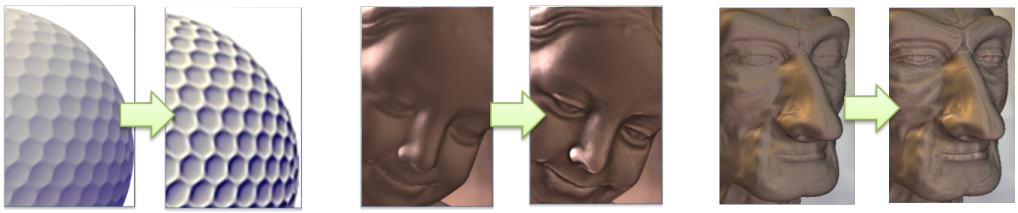


Figure 2.6: *Surface Feature Enhancement. (Left) Exaggerate shading presented by Rusinkiewicz et al. [75] (©2006 ACM). (Middle) Light warping technique proposed by Vergne et al. [97,98] (©2009 ACM). (Right) Radiance scaling techniques presented by Vergne et al. [99,100] (©2010 ACM).*

with shape transformations by dragging operations. Similarly, Ritschel et al. [74] proposed a method to deform lighting properties through a cloth simulation.

Some approaches give more direct control over the shapes of highlights. For example, Choi et al. [20] proposed the use of texture projection to design arbitrary highlight shapes. Pacanowski et al. [68] provided intuitive painting methods to control highlight shapes.

Figure 2.7: *Directable control of stylized rendering. (Left) Directable cartoon highlights presented by Anjyo et al. [4] (©2006 ACM). (Right) Projective texture for highlights proposed by Choi et al. [20] (©2006 Springer).*

Among these methods, the approach of Anjyo et al. [4] is the most relevant to our work because they focused on an artist-friendly system that included dynamic control. The highlight shape could be interactively designed using simple transform operations. In their approach, the shape of the highlight is deformed by a vector transform function $f_H : \mathbb{S}^2 \mapsto \mathbb{S}^2$ for the half vector $\mathbf{H}$. With a set of simple parameters, the vector transform function f_H permits interactive design of symbolic highlight shapes.

Whereas Anjyo et al. [4] focused on the global transformation of a simple circular highlight shape, our methods provide detailed control over the shape of local lighting effects (Chapter 4) and shading stylization based on model features (Chapter 5). In addition, our practical shading model (Chapter 6) creates a more expressive shading appearance than simple shading appearance of these methods.

2.5 Directable Control for Expressive Shading

Providing directable control of expressive shading is major challenge in stylized rendering researches and their applications. There is a significant demand for fine-grained control over expressive shading styles. However, there have been very few studies on how to provide the interactive techniques to meet this demand.

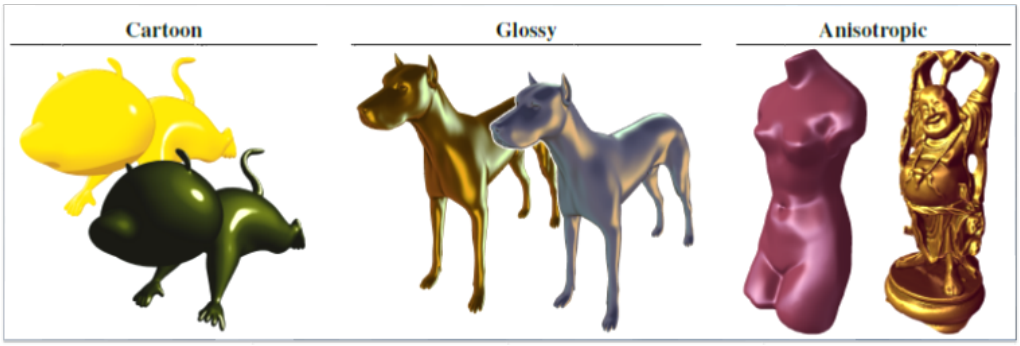


Figure 2.8: *Various shading styles presented by Vanderhaeghe et al. [96] (©2011 ACM). Their method allows the design of multiple shading primitives, including dynamic control over shapes and reflectance properties.*

Among the many shading techniques, one recently proposed by Vanderhaeghe et al. [96] may provide the best solution to date to the difficult problem of dynamic control. Their method gives the artist control over shapes of multiple lighting and their reflectance properties, based on proposed shading primitives. However, each shading primitive can handle only conventional 1D, not 2D shading tones.

Inspired by their work, we explored practical shading models to design 2D shading tones with suitable dynamic shading stylization (see Chapter 6). Although additional capabilities are required to meet the many demands of artists, we believe that our approach provides a practical solution to the key challenge of dynamic control of shading design in stylized rendering.

2.6 Other Stylized Rendering Methods

In this section, we briefly review other stylized rendering methods for expressive artistic styles, although not specifically related to shading.

2.6.1 Painterly Rendering

The approaches described so far focused on effective shading models for specific target appearances. On the other hand, painterly rendering techniques focus on overlapping brush strokes. In 1996, Meier [56] proposed a painterly rendering pipeline, in which the system applies a brush stroke style to static object-space particles. This work was extended to dynamic particle systems [12, 14, 47], where the particles are placed by temporally coherent noise function. In this approach, shading information is used only to specify the color of each particle. Kulla et al. [49] and Yen et al. [109] relied more on shading information to determine the transition of brush stroke styles affected by brightness values.

Figure 2.9: *Painterly rendering. (Left) Painterly rendering method presented by Meier [56] (©1996 ACM). (Right) Recent method of coherent shading stylization proposed by Bénard [13] (©2013 ACM).*

Although these approaches can deal with detailed shading appearance using brush stroke styles, few digital animations and computer games use these methods. Dynamic control of brush strokes is more difficult and time-consuming than shading control. Nevertheless, the animation industry is researching intuitive and efficient control over brush stroke styles [13, 25, 84, 106]. We expect that these rendering styles using appropriate brush stroke controls will be employed by artists in the future.

2.6.2 Line Drawing

Another important element of stylized rendering is line drawing, which has been of interest to the stylized rendering community since the work of Saito and Takahashi [78]. In 1997, Markosian et al. [53] proposed an interactive stylized line drawing method responding to views. Northrup and Markosian [61] extended this work to include temporal coherence and line stylization. A silhouette detection algorithm was improved by Hertzmann et al. [39] and Sander et al. [82] for efficient line rendering. DeCarlo et al. [27] proposed suggestive contours, which depict the shape with interior contours. Lee et al. [52] presented an image space approach for finding edges and ridges. Apparent ridges presented by Judd et al. [42] extract view-dependent ridges in an object space approach.

Figure 2.10: *Line drawing styles presented in WYSIWYG NPR [44] (©2002 ACM).*

WYSIWYG NPR proposed by Kalnins et al. [44] is unique in that the system allows the

artist to design annotated strokes and brush styles directly on the 3D model. They extended this work to maintain temporal coherency for stylized silhouettes [45]. OverCoat, a system recently presented by Schmid et al. [84] also aims to provide an artist-friendly framework for line drawing.

Although this thesis is focused on shading design, line drawing is also an important visual element of stylized rendering. More expressive results could be obtained by combining such line stylization techniques with the shading methods of our system.

2.7 Summary

In this chapter, we reviewed existing methods of stylized rendering from the perspective of directable controls which are essential for an artist-friendly stylized shading design framework. Like the stylized shading methods described above, our approach is also based on the fundamental methods of early stylized rendering in Section 2.2. Style extensions in Section 2.3 provide additional functionalities for designing expressive shading styles, but often lack the capability for dynamic control of the shading appearance through an intuitive and interactive interface. Some approaches allow more direct control over the lighting shape (Section 2.4) but provide little in the way of shading style controls.

Inspired by these approaches, we sought to provide an intuitive and interactive methods for stylized shading design for production work using our artist-friendly shading design framework. In contrast to other researches, our methods in Chapters 4- 6 provide new shading representations for efficient shading design to meet typical directional demands where non-physical artistic depictions are seamlessly integrated into physics-based lighting.

Chapter 3

Our Approach for Artist-Friendly Stylized Shading Design

In the reminder of this thesis, we will apply the proposed principles to different levels of shading editing to verify the effectiveness of our artist-friendly shading design framework. To achieve this, introducing well-designed behavior of shading models is essential for an intuitive and efficient design process. In this chapter, we consider appropriate representations of directable shading models for different levels of shading design processes, from small scale to large scale control. We start by reviewing and analyzing the general cartoon shading process, that is commonly used in a production work. Based on this analysis, we introduce directable shading mechanisms for the proposed shading design systems in Chapters 4- 6.

3.1 Analysis of General Cartoon Shading Process

A general cartoon shading model is strongly constrained by a physical lighting model, therefore it is difficult to control the shading appearance in an intuitive and appearance-based way. To explain its shading mechanism more concisely, we details the cartoon shading process in Equation 2.2:

$$\mathbf{c} = cm_d^{1D}(\mathbf{L}\cdot\mathbf{N}) + cm_s^{1D}(\mathbf{H}\cdot\mathbf{N}), \tag{3.1}$$

where the inputs are the light vector $\mathbf{L}$, the surface normal vector $\mathbf{N}$, and the half vector $\mathbf{H} := (\mathbf{L}+\mathbf{V})/\|\mathbf{L}+\mathbf{V}\|$, where $\mathbf{V}$ is the viewing vector. Based on the diffuse term $\mathbf{L}\cdot\mathbf{N} \in [-1,1]$ and specular term $\mathbf{H}\cdot\mathbf{N} \in [-1,1]$, the final color $\mathbf{c}$ is obtained from the 1D color mapping functions $cm_d^{1D} : [-1,1] \to \mathbb{C}$ for diffuse shading and $cm_s^{1D} : [-1,1] \to \mathbb{C}$ for specular shading. These elements are affected by the following sub design tasks:

- Shape modeling: a shape consists of a surface position and the surface normal $\mathbf{N}$. The surface normal $\mathbf{N}$ affects both the diffuse and specular term. The surface position indirectly affects the half vector $\mathbf{H}$ since per-position view vectors $\mathbf{V}$ are used to compute the half vector.
- Material design: the artist designs the color mapping functions (cm_d^{1D}, cm_s^{1D}) using a few simple parameters. These functions are used to determine sample shading colors based on the brightness terms (diffuse, specular).
- Camera design: camera manipulation affects the view vector $\mathbf{V}$, which is used to obtain the half vector $\mathbf{H}$.

- Lighting design: the light vector **L** is determined by the location of the light source and the type of light. It is used to compute the diffuse term and affects the half vector **H**.

In these design tasks, the artist needs to control the shading elements carefully to obtain the desired appearance. However, these indirect controls for shading design are time-consuming and impractical in production environments. Ideally, artists would use more intuitive and directable controls to obtain the shading desired.

3.2 Our Approach for Directable Shading Model

As explained above, the issue of the general cartoon shading process is that its editing tasks are interconnected and indirect for changing the shading appearance. To solve the issue, we consider appropriate directable shading models by following our artist-friendly shading design framework. In accordance with **Principle 1**, the requirement of a directable shading model is to provide intuitive controls which directly affect specific visual features. In accordance with **Principle 2**, we extend the original cartoon shading model to achieve seamless integration with existing 3D lighting controls. Accordingly, we introduce a general form of directable shading model as:

$$\mathbf{c} = cm_d(f_d(\mathbf{L},\mathbf{N}) + o_d(x)) + cm_s(f_s(\mathbf{H},\mathbf{N}) + o_s(x)), \tag{3.2}$$

where we use key directable mechanisms: lighting transforms and lighting offsets. The lighting transforms $f_d(\mathbf{L},\mathbf{N})$ and $f_s(\mathbf{H},\mathbf{N})$ deform the diffuse and specular lighting to change the overall lighting shape. The lighting offsets $o_d(x)$ and $o_s(x)$ are used to add smaller scale local lighting effect. With these input lighting, the final color **c** is obtained through the color mapping functions cm_d and cm_s. To meet directional demands for different levels of the shading editing, we reformulate these key directable shading mechanisms for each shading design system in Chapters 4- 6 as follows.

Locally controllable shading with intuitive paint interface: Our shading model in Chapter 4 lets the artist modify the shaded area with a local painting operation. We provide the directable control by adding lighting offsets to the brightness term directly:

$$\mathbf{c} = cm_d^{1D}(\mathbf{L}\cdot\mathbf{N} + o_d^{1D}(\mathbf{p})) + cm_s^{1D}(\mathbf{H}\cdot\mathbf{N} + o_s^{1D}(\mathbf{p})), \tag{3.3}$$

where the diffuse term $\mathbf{L}\cdot\mathbf{N}$ and specular term $\mathbf{H}\cdot\mathbf{N}$ are modified by adding the corresponding scalar offset functions $o_d^{1D}(\mathbf{p}) \in [-1,1]$ and $o_s^{1D}(\mathbf{p}) \in [-1,1]$ that are defined on a surface point **p**. This method is suitable for an artist who wants to freely add local lighting effects. The paint operation has no direct effect on the light vector **L**, surface vector **N**, or the half vector **H**. In addition, the modification must be local on the painted area. We therefore use the scalar offset functions defined on the surface in this method.

Shading stylization based on model features: Our shading model in Chapter 5 allows the artist to design commonly used feature enhancements such as straight lighting, edge enhancement, and detailed lighting effects. We provide the directable control by applying lighting transforms and lighting offsets based on model features:

$$\mathbf{c} = cm_d^{1D}(f_d(\mathbf{L},\mathbf{N}) + o_d^{1D}(E)) + cm_s^{1D}(f_s(\mathbf{H},\mathbf{N}) + o_s^{1D}(E)), \tag{3.4}$$

where the diffuse and specular lighting are deformed by applying lighting transform functions $f_d(\mathbf{L},\mathbf{N}) \in [-1,1]$ and $f_s(\mathbf{H},\mathbf{N}) \in [-1,1]$, respectively. $o_d^{1D}(E)$ and $o_s^{1D}(E)$ are lighting offset functions where $E \in \mathbb{R}$ is the edge distance. These lighting transforms and lighting offset are designed based on the model features: the flat surface normal

and the edge distance field. To show an object's flatness, we linked the straight lighting with the surface normal vector **N**. We chose to use the lighting transforms for this lighting effect, because they are effective to control the shape of the lighting. In the case of edge enhancements, the lighting effect is considered a local effect, compared with straight lighting effect. Therefore, we chose the scalar offset functions defined in the edge distance field for this lighting effect.

Practical shading model for expressive shading styles: Our shading model in Chapter 6 lets the artist design an overall material with detailed shading appearance. We provide directable control by introducing a new lighting procedure:

$$\mathbf{c} = cm_d^{2D}(f_d^{2D}(\mathbf{L},\mathbf{N}) + o_d^{2D}(h)) + cm_s^{2D}(f_s^{2D}(\mathbf{H},\mathbf{N}) + o_s^{2D}(h)), \tag{3.5}$$

where we introduce the 2D shading functions for more expressive global 2D shading effects. The lighting transform functions $f_d^{2D}(\mathbf{L},\mathbf{N}) \in [-1,1]\times[-1,1]$ and $f_s^{2D}(\mathbf{H},\mathbf{N}) \in [-1,1]\times[-1,1]$ are reformulated to fit into the 2D coordinate representations. We also reformulate the offset functions $o_d^{2D}(h)$ and $o_s^{2D}(h)$ as functions of the attribute value $h \in [-1,1]$. The final color **c** is obtained by the 2D color mapping functions cm_d^{2D} : $[-1,1]\times[-1,1] \mapsto \mathbb{C}$ and $cm_s^{2D} : [-1,1]\times[-1,1] \mapsto \mathbb{C}$. The primary challenge here is to achieve more expressive shading styles beyond the simple styles of cartoon shading. We chose the 2D shading representation because it can represent a more complex 2D color distributions than the limited 1D color distributions of cartoon shading.

3.3 Summary

In this chapter, we introduced key directable shading mechanisms for our artist-friendly stylized shading design framework. Analyzing the general cartoon shading process used in production work, we found that the main difficulty consists in that conventional controls are indirect for changing the shading appearance. Based on the analysis, our directable shading models aim to achieving intuitive behaviors for supporting creative design of artists.

In the following Chapters 4-6, we will present how these directable shading models in greater detail and verify the effectiveness of our proposed framework.

Chapter 4

Locally Controllable Shading with Intuitive Paint Interface

4.1 Overview

The first experiment is to apply our framework to artist-friendly user interface for local edits of stylized shading. In the case of local controls, the ability to add intentional, but often unrealistic shading effects is indispensable for cartoon animations. In this chapter, we present an interactive system that allow the artist to freely paint local lights and shades to a model. In accordance with **Principle 1** (directable shading model for artistic control), we design the shading model which enables an intuitive, direct manipulation method based on a paint-brush metaphor, to control and edit the light and shade as desired. The key idea for this directable shading model is to modify brightness term directly, adding a scalar lighting offset function. This complies with our **Principle 2** (seamless integration with 3D lighting) in that the modified shading can be manipulated by multiple different light types of light sources such as directional lights, points lights, and spot lights. Besides, artists can also use a convenient key-framing technique for fine-tuning of stylistic animation in a familiar way. Finally, our system demonstrates how our method can enhance both the quality and range of applicability of conventional stylized shading for interactive applications.

4.2 Introduction

Here we consider the problem of how to provide artists with intuitive, fine-grained control over stylized light and shade on a 3D object. Over the past decade, a variety of stylized rendering techniques have been developed to facilitate visual interpretation of 3D objects. Most of these techniques are designed to elucidate particular attributes inherent to the object. For example, Gooch and Gooch [36] developed a lighting model that changes hue to convey surface orientation, edge locations, and highlights for 3D technical illustration. The multi-scale shading method by [75] depicts 3D shape details at all frequencies possible.

On the other hand, in application fields such as digital animation and video games, there is a significant demand for locally controllable stylized light and shade, which can achieve results that are directable, intentional, and often fictive, yet ultimately more attractive for it. For example, the conventional cartoon shader used routinely in 3D animation often creates undesirable shaded areas. These can arise from the complexity of the underlying geometry or the complexity of the lighting, or just as a result of the basic

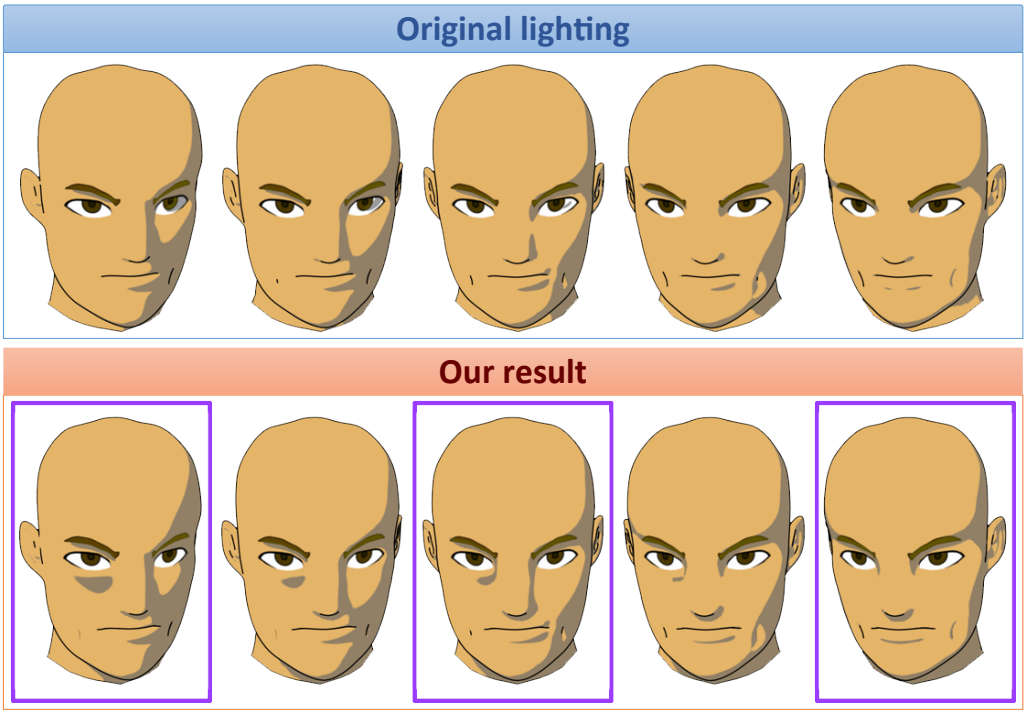


Figure 4.1: *Comparison of conventional cartoon shading (top row) with our result (bottom row). Edits were made at the three key frames indicated including: added shaded area below left eye for expressive impact, deleted dark area around right eye, and added shaded area below nose to emphasize three-dimensionality. These local edits integrate seamlessly with the global lighting, animate smoothly, and require no modification to the external lighting setup.*

physics of illumination. The left image in Figure 4.1 shows such an example, where the dark area partly covers the right eye of the character. Directors would like to have the ability have such features removed while retaining other dark areas. In other cases, they might like to request that a shaded area be added below the left eye, as shown in the second image from the left in Figure 4.1, in order to emphasize the character's fierceness. However, satisfying these diverse artistic requirements simultaneously would be very hard or almost impossible using only existing conventional lighting control and/or by fine-tuning the parameters used. Changing the geometry of the model or animating textures or light maps might be helpful for achieving this, but these are time-consuming and impractical on a production schedule. Despite the crucial importance of such fine-grained artistic control of stylized light and shade, very little research exists on how to provide such control or suitable interactive techniques to support it.

Our goal is to develop such artist-friendly methodologies for stylistic depiction of light and shade. To explain our approach more concisely, we restrict the discussion for now to making 3D cartoon animation. In this case, due to the nature of stylistic depiction, the techniques used need not be physically realistic; however, they must possess a certain sense of *plausibility* while meeting directorial demands. This emphasis on expressiveness over physical-realism implies that we must rely on the animator's creativity–more than automatic physically-based algorithms–to get a desired animation. Therefore, a stylized shading approach should provide a simple, intuitive user interface so that the animator can easily and interactively translate his or her creative vision into reality. Figure 4.2 shows the proposed requirements of a user interface to fulfill the demands of artists. Paint brush metaphor is a simple but effective way to specify a desired shaded

area. A keyframe-based technique is appropriate, since it allows fine-tuning of stylistic animation in a traditional, but convenient and familiar way for animators. Additionally, real-time preview of the animation is also indispensable. These basic requirements for making stylized animation have led us to consider naïve key-framing as a first approach towards a new methodology.

The central idea of our approach is to effect the desired changes to light and shade boundaries by modifying the Lambertian $\mathbf{L} \cdot \mathbf{N}$ brightness term directly, adding a scalar lighting offset function. This avoids the need to manipulate light vectors and normals and can be efficiently implemented using scalar-valued radial basis functions [101]. The right images in Figure 4.1 are from an animation created using our techniques, while the leftmost shows the scene before modifications.

The rest of the chapter is organized as follows. After briefly surveying related work in Section 4.3, we describe the main ideas underlying the algorithms in Section 4.5. In Section 4.6, we describe some implementation details of our prototype system. Section 4.7 demonstrates animation examples and discusses our results. We conclude with some limitations and future work in Section 4.8.

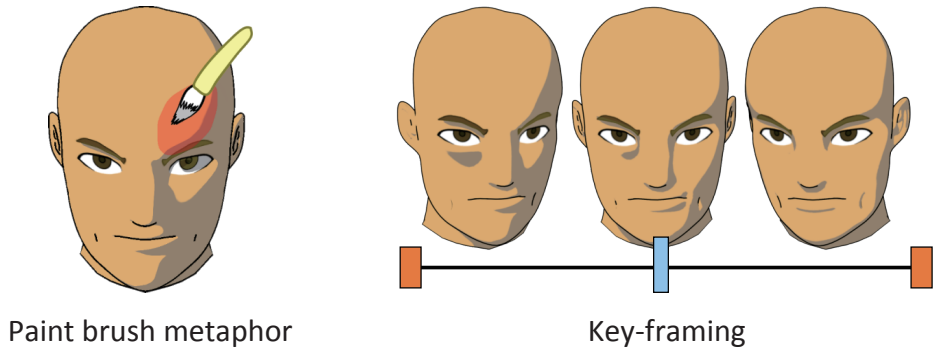


Figure 4.2: *Intuitive user interface proposed in our system. (Left) Paint brush metaphor provides an easy way to modify the shaded area. (Right) Key-framing is convenient and familiar for artists to control animation.*

4.3 Background

A number of stylized rendering techniques, such as those in [36], have been developed to emulate various stylistic appearances. For stylized rendering of 3D objects, Lake et al. [50] proposed several fundamental real-time rendering techniques, including a traditional cartoon shader. The Lit-Sphere method by Sloan et al. [87] can describe view-independent tone detail, using a painted spherical environment map. The WYSIWYG system by Kalnins et al. [44] allows direct drawing of strokes onto 3D objects, while learning strokes by example. The multi-scale shading technique by Rusinkiewicz et al. [75] can also control the appearance of shape detail by tuning parameters of the lighting model. Barla et al. [11] proposed an extension of the traditional cartoon shader, which can control view-dependent tone detail, including such effects as aerial perspective and depth of field. The cartoon highlights in [4, 96] allows a user to directly click-and-drag the highlights on a surface to design and animate them. After our work was published, Pacanowski et al [68] proposed an intuitive painting method for highlight design.

Existing work on user-specified indirect lighting design for photorealistic scene rendering is to some extent related to our approach as well. The design issue in photorealistic lighting is to find the light placement that results in the user-specified highlights and shadows in the scene (see [51] for more detailed discussion). There exist several good approaches ([48,69,85], for instance). The geometry-dependent lighting method by [51] may also be a useful indirect light design tool for visualizing scientific data. Okabe et al. [63] and Akers et al. [2] take other approaches to modifying lighting, providing an intuitive painting method for modifying the illumination of 3D models.

Our approach is inspired by all of the above methods. However, ours is unique in that it allows a user to add light and shade by painting them directly onto 3D objects without elaborate lighting control, to make stylistic animation by key-framing. In addition, we demonstrate that continuous tone detail can also be painted and animated as an extension of our approach.

4.4 User Interaction

This section describes a typical shading design process using our prototype system. As illustrated in Figure 4.3, our approach is based on the direct painting of shaded areas, displayed on a 3D view. For making animation, our system also provides a time slider to specify a target frame for each painting operation. The overall process of the approach we propose is:

1. Begin by making an initial 3D scene, which includes the lighting and animation settings, using a conventional 3D software tool. Multiple directional and/or light sources can be used for the initial lighting design.

2. At each keyframe, the artist designs and/or modifies the shaded area on a surface, using a paint-brush interface. This process is performed at interactive rates, prescribing the boundary constraint of the obtained area. Thereafter the new surface brightness distribution is automatically generated considering the boundary constraint.

3. The new surface brightness distributions at the keyframes are automatically transmitted to all the frames by linear interpolation. We thus obtain the desired animation of the shaded areas.real-time preview of the stylistic animation.

During the shading design process, the artist can freely change the viewpoint. Our system also provides real-time feedback to the editing actions at any time.

4.5 Algorithm

4.5.1 Overall Process

We begin by restricting ourselves to 3D cartoon animation, where each shaded area is assigned a uniform color by 1D color mapping [50]. Starting from a 3D scene created using conventional lighting and key-framing techniques, we consider how to locally add light and shade onto surfaces. In particular, we describe how to use a paint-brush metaphor to design the shaded area at keyframes. The painting process at a given keyframe involves interactively adding light and shade details or sculpting the shapes of shade boundaries. Such editing is straightforward with our technique, while it would be very time-consuming and difficult to manage using conventional lighting.

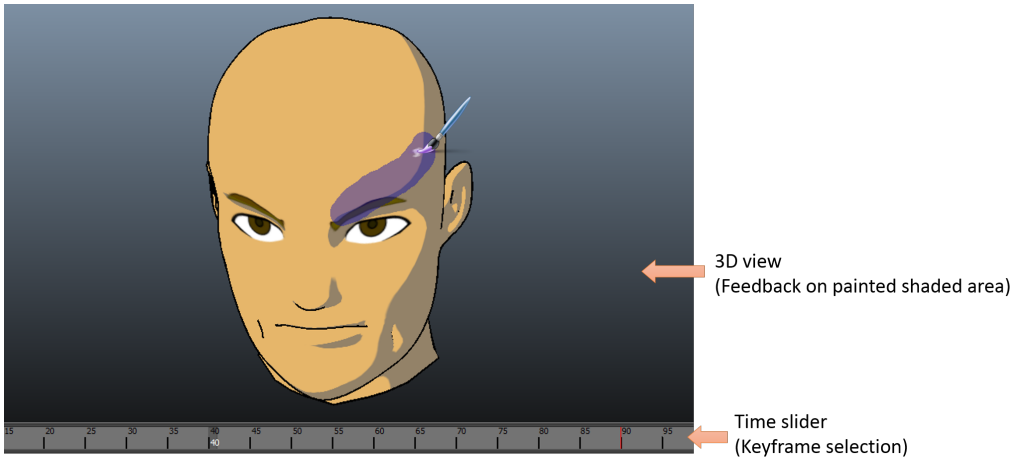


Figure 4.3: *A screen snapshot of our prototype system. On the 3D view, the user can paint shaded areas with real-time preview. The time slider is used to specify a target frame to paint.*

Our implementation is capable of dealing with deforming geometry and multiple directional, point, and/or spot light sources; however, without loss of generality, we explain our idea below in the context of a single light source. The extension to deformations and multiple light sources is straightforward. For a given threshold $0 < d_0 < 1$ a thresholded 1D color mapping creates two (possibly disconnected) regions, which we will call the *light* and *dark* areas. More precisely, using set notation we define the *light* area $\mathbf{B}_0$ on a surface $\mathbf{S}$, for a given threshold d_0 to be:

$$\mathbf{B}_0 := \{\mathbf{p} \in \mathbf{S} \mid \mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p}) \geq d_0\}, \tag{4.1}$$

where $\mathbf{L}(\mathbf{p})$ and $\mathbf{N}(\mathbf{p})$ are the unit vectors representing the light direction and surface normal at a point $\mathbf{p}$ on $\mathbf{S}$, respectively. The boundary between light and dark areas is obtained by replacing inequality ($\geq d_0$) with equality ($= d_0$) above. We will refer to the dot product $\mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p})$ in Equation 4.1 as the *intensity distribution*. Given these definitions, let us consider how to enlarge a portion of the light area, for example on the character's face in Figure 4.4, where the light area $\mathbf{B}_0$ is flesh colored. Let the area $\mathbf{C}_0$ with boundary $\partial\mathbf{C}_0$ (drawn in red in Figure 4.4) be an area painted with our brush-type interface (see the next section for specifics). The area $\mathbf{C}_0 - \mathbf{B}_0$ is the area that the user wishes to add to the original area $\mathbf{B}_0$. The core idea behind our approach is to *modify the intensity distribution* in order to make the light area change as desired, *i.e.* so that it becomes $\mathbf{B}_0 \cup \mathbf{C}_0$. The intensity distribution is a scalar function, so this greatly simplifies the problem when compared to working directly with light vectors and normals. The overall strategy is as follows. We first construct an *offset function* $o_1(\mathbf{p})$ defined globally on $\mathbf{S}$. This prescribes the new light area by replacing the original intensity distribution in Equation 4.1 with $\mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p}) + o_1(\mathbf{p})$(see Figure 4.4). Note that,

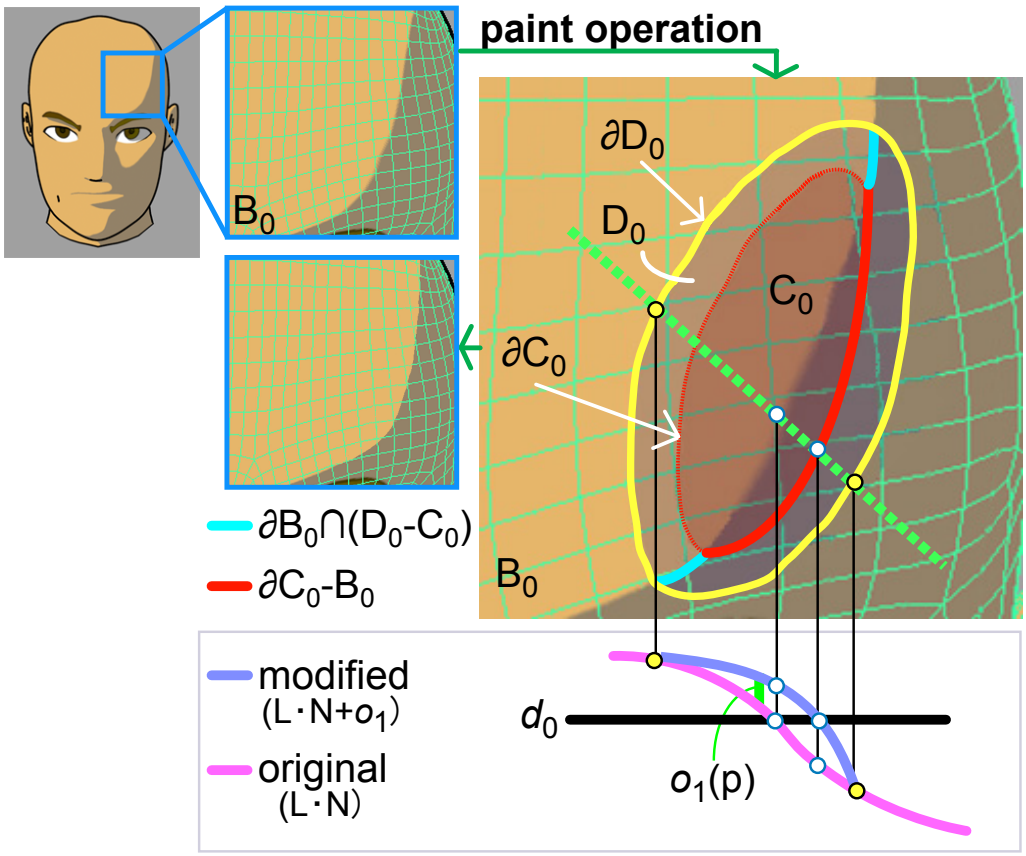


Figure 4.4: *Modifying a shaded area* $\mathbf{B}_0$ *with the paint brush interface: The resulting new area* $\mathbf{B}_0 \cup \mathbf{C}_0$ *can be represented functionally by introducing an offset function that modifies the standard* $\mathbf{L} \cdot \mathbf{N}$ *lighting term. The bottom graph shows a 1D intensity distribution along the green line.*

though globally defined, the offset function should be mostly zero except in the region immediately surrounding the desired edit.

After making a modification at one keyframe, we can create a different offset function to define the light area at a second keyframe. By smoothly interpolating the offset functions between keyframes, we can achieve smooth animation of the light areas between frames as well. The procedure can be repeated for every pair of adjacent keyframes, resulting in an animated light area on $\mathbf{S}$ using just local edits with a paint-brush.

4.5.2 The Lighting Offset Function and Key-framing

Next, we describe how to construct the lighting offset function for a "painted" light area. Given the original light area $\mathbf{B}_0$ from Equation 4.1 and the painted area $\mathbf{C}_0$, as shown in Figure 4.4. The offset function $o_1(\mathbf{p})$ for $\mathbf{B}_0 \cup \mathbf{C}_0$ should satisfy

$$\mathbf{B}_1 := \{\mathbf{p} \in \mathbf{S} \mid \mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p}) + o_1(\mathbf{p}) \geq d_0\} = \mathbf{B}_0 \cup \mathbf{C}_0, \tag{4.2}$$

where $o_1(\mathbf{p})$ is generated when the user finishes drawing $\mathbf{C}_0$. To fulfill condition (Equation 4.2), it is clear that the offset function should take values that are equal to $d_0 - \mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p})(\geq 0)$ on the new boundary $\partial\mathbf{C}_0 - \mathbf{B}_0$. On the other hand, to make the offset function "active" only in the neighborhood of $\mathbf{C}_0$, we wish to have an area $\mathbf{D}_0$, which includes $\mathbf{C}_0$, that limits the extent of the domain where modifications to the lighting are

applied (see Figure 4.4). In our current implementation, the distance between $\partial \mathbf{D}_0$ and $\partial \mathbf{C}_0$ is controlled by a slider in the user interface. The size of this region gives the user a way to limit the scope of modification (also see the detail in Section 4.7). Therefore $o_1(\mathbf{p})$ should minimally satisfy the following conditions:

$$o_1(\mathbf{p}) = \begin{cases} 0 & \mathbf{p} \in (\mathbf{S} - \mathbf{D}_0) \cup (\partial \mathbf{B}_0 \cap (\mathbf{D}_0 - \mathbf{C}_0)) \\ d_0 - \mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p}) & \mathbf{p} \in \partial \mathbf{C}_0 - \mathbf{B}_0 \end{cases} \tag{4.3}$$

If we choose for o_1 a continuous function satisfying the above conditions, then the resultant area $\mathbf{B}_1$ will have a continuous boundary. We can consider the new shaded area $\mathbf{B}_1$, to have a "generalized" intensity distribution given by $\mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p}) + o_1(\mathbf{p})$, instead of $\mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p})$. The above procedure can be repeated for each stroke, building upon the offset function created by the previous stroke. The user's kth stroke provides $\mathbf{C}_k$ and $\mathbf{D}_k$. From this new input, the resulting light area can be defined recursively as:

$$\mathbf{B}_{k+1} := \{\mathbf{p} \in \mathbf{S} \mid \mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p}) + o_{k+1}(\mathbf{p}) \geq d_0\} = \mathbf{B}_k \cup \mathbf{C}_k, \tag{4.4}$$

where we assume that $o_{k+1}(\mathbf{p})$ is a continuous function satisfying the constraints:

$$o_{k+1}(\mathbf{p}) = \begin{cases} o_k(\mathbf{p}) & \mathbf{p} \in (\mathbf{S} - \mathbf{D}_k) \cup (\partial \mathbf{B}_k \cap (\mathbf{D}_k - \mathbf{C}_k)) \\ d_0 - \mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p}) & \mathbf{p} \in \partial \mathbf{C}_k - \mathbf{B}_k \end{cases} . \tag{4.5}$$

$\mathbf{D}_k$ includes $\mathbf{C}_k$ and serves the same role for $\mathbf{C}_k$ as $\mathbf{D}_0$ does for $\mathbf{C}_0$. The conditions in Equation 4.3 can be seen to be a special case of (Equation 4.5) if we define $o_0 = 0$. Again we note that, outside of $\mathbf{D}_k$, no modifications will be made to the lighting (i.e., $o_{k+1}(\mathbf{p}) = o_k(\mathbf{p})$). In the $\mathbf{D}_k - \mathbf{C}_k$ region, no modification will be visible under the current lighting conditions, but some modification may be visible when either the lights or the model are moved. Having a $\mathbf{D}_k - \mathbf{C}_k$ band allows for smooth transition from modified $o_k(\mathbf{p})$ values to the original values.

To make the above strategy computationally tractable at interactive rates, we represent the offset function $o_k(\mathbf{p})$ with a sum of Radial Basis Functions (RBF), denoted by $\hat{o}_k(\mathbf{p})$. Thus in practice we use:

$$\hat{\mathbf{B}}_k := \{\mathbf{p} \in \mathbf{S} \mid \mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p}) + \hat{o}_k(\mathbf{p}) \geq d_0\} \tag{4.6}$$

in place of $\mathbf{B}_k$, and the boundary constraint (Equation 4.5) is only discretely enforced at a finite number of points. The RBF approximation $\hat{\mathbf{B}}_k$ is made from the shaded area obtained by the paint operation. Rigorously, the boundary of $\hat{\mathbf{B}}_k$ may not exactly match that of the original painted area. To allow fine adjustment, we provide two additional types of brushes: an *intensity brush* and a *smoothing brush*, which will be described in Section 4.5.4.

Keyframing: Modifications made according to the above algorithm integrate smoothly with standard lighting equations, and for many animations a single offset function o_k may suffice. However, in order to create more elaborate modifications, it is possible to create several keyframes, with a unique offset function $o_{k,f}$ at each frame f, leading to more complex animation of light and shade. Lighting of the animation as a whole can then be accomplished by interpolating the offset functions $o_{k,f}$ (see Figure 4.5). In our prototype we have used simple linear blending for this purpose, though more complicated blending functions are possible and worth exploring.

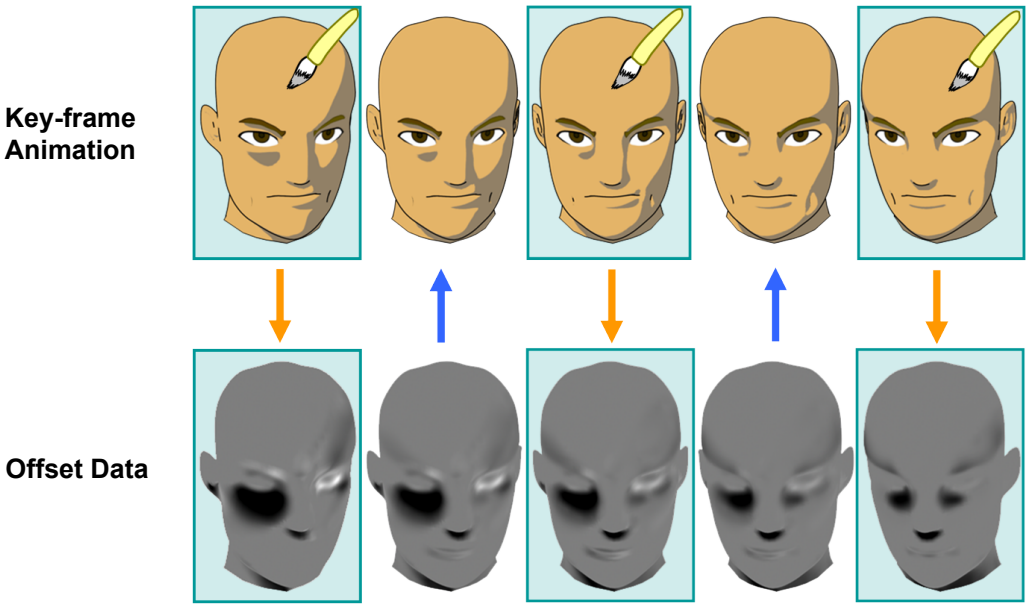


Figure 4.5: *Creating key-frame animation (top row) using lighting offset data (bottom row). (orange arrow) In order to allow the user to modify shaded area at several key-frames, we construct and store unique lighting offset data at each painted key-frame in the process as described previous. (blue arrow) Lighting offset distributions between key-frames are interpolated from key-frame offset data used simple linear functions. Then the final cartoon shading result can be obtained based on these offset distributions.*

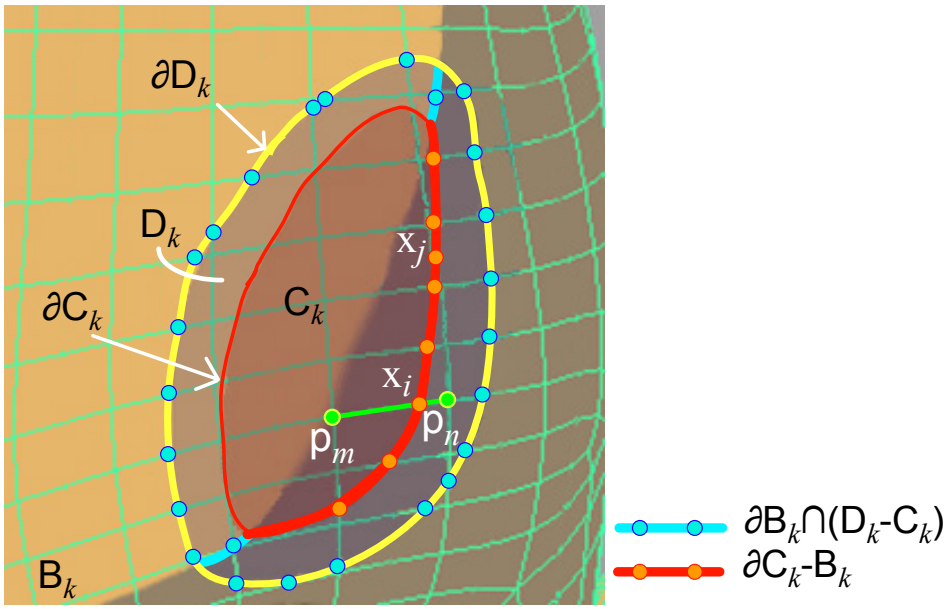


Figure 4.6: *The boundary constraint points used in finding the new offset function* $\hat{o}_{k+1}(\mathbf{p})$. *The orange points* $\{\mathbf{x}_i\}$ *take the value* $d_0 - \mathbf{L} \cdot \mathbf{N}$, *while the blue points are constrained to* $o_k(\mathbf{p})$.

4.5.3 RBF Approximation of The Lighting Offset Function

Suppose that $\mathbf{S}$ consists of polygon meshes, as shown in Figure 4.6. We will assume for simplicity that $\hat{\mathbf{B}}_k = \mathbf{B}_k$. After obtaining $\hat{o}_k(\mathbf{p})$ and $\mathbf{B}_k$ in Equation 4.6, we want to find $\hat{o}_{k+1}(\mathbf{p})$, which satisfies the boundary conditions (Equation 4.5) at a finite number of discrete points. We find a set of such points $\{\mathbf{x}_i\} \in \partial\mathbf{C}_k - \mathbf{B}_k$ by the following procedure. For each vertex $\mathbf{p}_m$ inside $\mathbf{C}_k$, we check adjacent edges for intersection with the boundary $\partial\mathbf{C}_k - \mathbf{B}_k$. For each intersecting edge, linear interpolation between $\mathbf{p}_m$ and the vertex at the other end, $\mathbf{p}_n$, is used to determine the approximate location of the boundary point $\mathbf{x}_i$. Note that we record stroke data per-vertex only and reconstruct the stroke linearly, thus no edge can cross the boundary more than once.

Now let $f \equiv \hat{o}_{k+1}$. We find a continuous f satisfying Equation 4.5 for $\{\mathbf{x}_i\}$ in the following form [33, 95, 101]:

$$f(\mathbf{x}) = \sum_{i=1}^{l} w_i \phi(\mathbf{x} - \mathbf{x}_i) + \mathscr{P}(\mathbf{x}), \tag{4.7}$$

where ϕ is a radial basis function, $\{w_i\}$ are weights, and $\mathscr{P}$ is a polynomial whose degree depends upon the choice of ϕ. In our case, l is the number of the boundary constraint points shown in Figure 4.6.

We employ $\phi(\mathbf{x}) = \|\mathbf{x}\|$ as the basis function after experimenting with various options. This corresponds to the solution of a generalized thin-plate spline problem on $\mathbb{R}^3$ [33, 101], and the curvature minimizing properties of this basis function seem to be well suited to this task. Satisfying a discretized version of Equation 4.5 reduces to solving a linear system of equations for the unknown weights $\{w_i\}$, and the four coefficients of the linear polynomial $\mathscr{P}$ on $\mathbb{R}^3$.

4.5.4 Additional Brushes

The previous sections described how we enable users to add and edit *light* areas using a paint-brush metaphor. In a similar way we can add and edit *dark* areas. In that case the only difference is the selection of boundary points used in Equation 4.5. Instead of using $\partial\mathbf{C}_k - \mathbf{B}_k$, we use the opposite half of $\partial\mathbf{C}_k$, that is, $\partial\mathbf{C}_k \cap \mathbf{B}_k$. The user simply switches the editing mode from *light* to *dark*. In both cases, the paint brush is used for roughly specifying the shading boundary. We call this type of brush a *boundary brush*.

The boundary brush works well to get a desired shape, but the intensity distribution may not change as smoothly as desired. This can be due to the radial basis function we select or due to too many conflicting constraints. For example, we have seen in our experiments that even a smooth radial basis function may result in a rapidly changing intensity distribution in the area where the distribution contours are very close to one another. This may cause the resulting keyframe animation to look unnatural. For this case, we have created a *smoothing brush*. By painting on the surface with the smoothing brush, the lighting offset values are filtered, while preserving the original value of $\mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p})$. In our implementation, the offset values stored per vertex are updated using a simple weighted average of values at connected vertices for each stroke operation. In this way we achieve shading effects that fade in and out more gradually and have smoother boundaries (see Figure 4.7).

In some cases it is useful to be able simply to add or remove an isolated light or dark area. For these situations we provide a simpler alternative to the boundary brush, which we call the *intensity brush*. This brush simply adds to or subtracts from the lighting

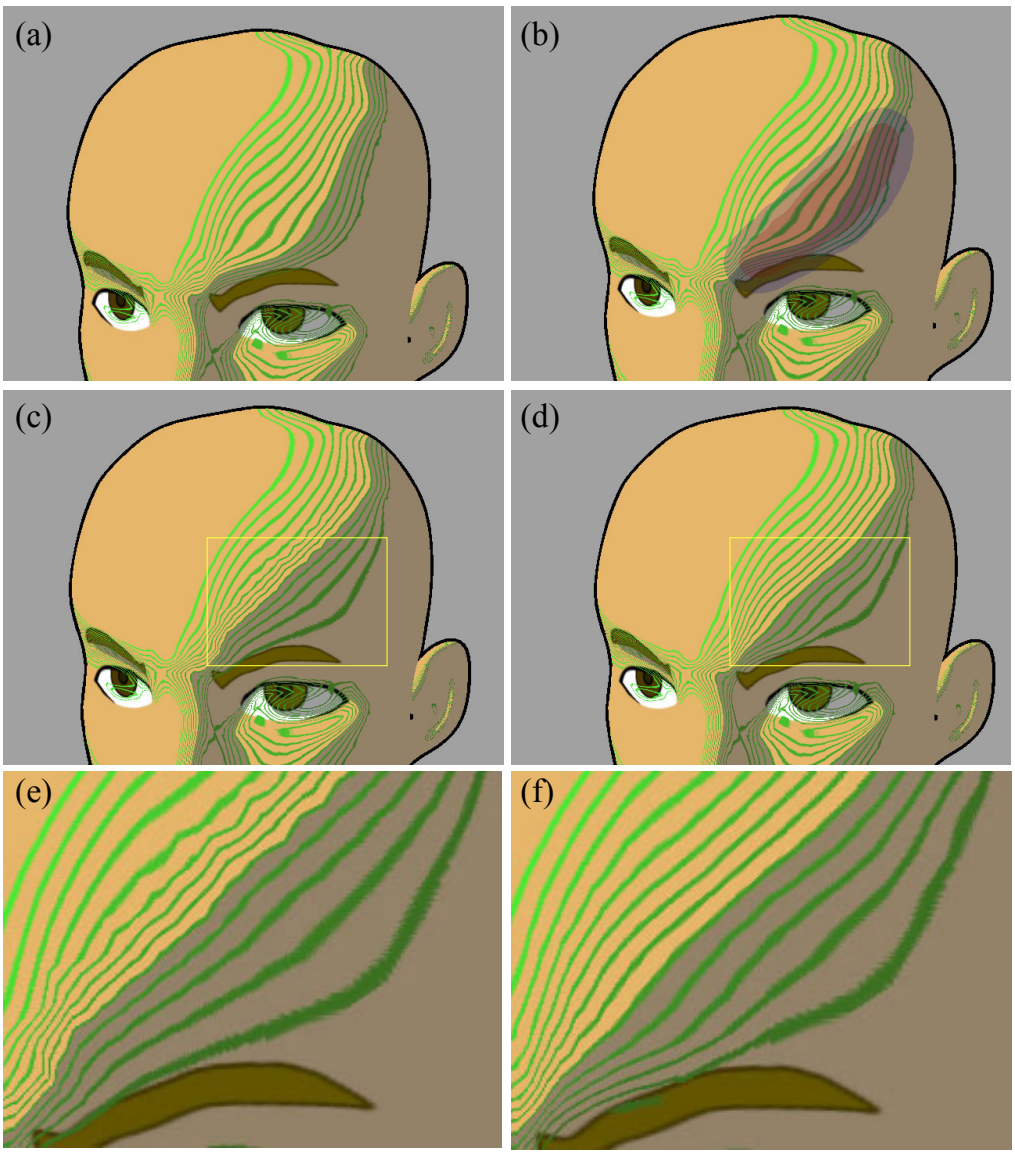


Figure 4.7: *Contours of the intensity distribution,* $\mathbf{L} \cdot \mathbf{N}$*, as influenced by our brush operations. (a) Initial distribution. (b) A boundary brush specifies a region which should become dark. (c) The new distribution with the lighting offset function prescribed by the region. (d) The distribution modified by a smoothing brush. (e–f) Details from (c–d).*

offset function o_k. The amount added is determined by a magnitude parameter and the radius of the brush. The magnitude is the amount to add to o_k along the centerline of the stroke. We fade the added intensity smoothly to zero at the edges of the stroke using a "smooth-step" cubic polynomial falloff.

Figure 4.7 shows a simple example of how to use these brushes. In Figure 4.7(a), an initial intensity distribution on the character is displayed using green contour lines. The boundary brush is then applied in (b). After getting the lighting offset function in Equation 4.6, we have the new intensity distribution as shown in (c). Using the smoothing brush, it is made smoother, as shown in (d).

4.5.5 Extensions

In order to get more variations of stylized light and shade, we add a few simple, but useful, extensions of the main algorithms above.

Specular Highlight: We can deal with stylized highlights in the same framework as the shaded area. In our system we simply need to replace the Lambertian term (the dot product, $\mathbf{L} \cdot \mathbf{N}$) in Equation 4.6 with $\mathbf{H} \cdot \mathbf{N}$ from Blinn's specular highlight model [15], where $\mathbf{H}$ is the normalized half-way vector between the light and the eye. The user can easily edit the highlights by the brushes in the same manner as the shaded area.

Continuous tone control: The threshold d_0 in Equation 4.1 is a global constant which controls the shaded area in accordance with Equation 4.6, but this is not an essential assumption. Similarly, we can use the paint-brush metaphors to locally control and edit continuous tone on a surface by dispensing with the threshold and defining the lightness at a given point to be simply $\mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p}) + o_k(\mathbf{p})$, or any continuous function thereof.

4.5.6 Lighting Offset Function Interpolation Based on Light Parameters

The methods presented so far assumes the key-framing control for lighting offset functions. In order to apply our method to more interactive applications, we extend our methods to a light-dependent framework. The user interface is almost same: the artist modifies shaded area at each state of light parameters. Similar to the key-framing case, it is possible to create several key-offset functions, with a unique offset function $o_{k,P_L i}$ for each state of light parameters $\mathbf{P}_{\mathbf{L}i}$. With the necessary set of offset functions, we can interactively animate designed lighting by interpolating offset functions $\{o_{k,P_L i}\}$ based on the input light parameters $\mathbf{P}_{\mathbf{L}}$.

Here we also use the RBF interpolation for this purpose. Now let $f \equiv o_{k,P_L}$ representing lighting offset function for arbitrary input light parameter set $\mathbf{P}_{\mathbf{L}}$. In this case, we design a continuous function f satisfying the following conditions for $\{\mathbf{P}_{\mathbf{L}i}\}$:

$$f(\mathbf{P}_{\mathbf{L}i}) = o_{k,P_L i}, \tag{4.8}$$

where the states of light parameters $\{\mathbf{P}_{\mathbf{L}i}\}$ is used for the conditions of Equation 4.7, by replacing the constraint points $\{\mathbf{x}_i\}$ with them. This function f lets the user change the light setting interactively with the desired shading designed using our system.

However, computing the RBF for each vertex is time-consuming due to the large number of vertices. To reduce the heavy computation, we approximate the RBF interpolation f by pose space interpolation. We first replace the RBF conditions in Equation 4.8 with:

$$f'(\mathbf{P}_{\mathbf{L}i}) = \mathbf{t}_i, \tag{4.9}$$

where $\mathbf{t}_i$ is the pose weight vector for each state of light parameters $\mathbf{P}_{\mathbf{L}i}$. In interpolation process, we first compute the pose weight vector $\mathbf{t}$ for the input light parameters $\mathbf{P}_{\mathbf{L}}$. With the weight vector, the final lighting offset function is obtained by simple blending of offset functions $\{o_{k,P_L i}\}$.

4.6 Implementation

We implemented our prototype system as a Maya plug in. As we described in Section 3, our system is based on the highly customizable feature of Maya.

GPU Implementation

To greatly reduce CPU's load, we have implemented the rendering algorithms using Maya's hardware shader functionality that allows shader code to be written using standard OpenGL and GLSL. In our rendering algorithms, for each vertex i with position $\mathbf{v}_i$ on surface meshes, the lighting offset function value $o_{k,f}(\mathbf{v}_i)$ at each keyframe is assigned and stored as a vertex color data in Maya. To transfer the offset function value to GPU as a varying parameter, the vertex color data is interpolated and decoded in CPU process. In our GPU programs, the intensity value is efficiently updated in the vertex shader, then the conventional cartoon shading process is computed in the pixel shader using the modified intensity value. Our rendering algorithms are quite simple, but effective and efficient for our use.

Paint-brush Metaphor

We need to find all of the vertices inside the brush stroke region and calculate their distances from the stroke centerline. This information is used to determine the locations of the points on the boundary in Figure 4.6, as well as to implement the smooth falloff of the intensity brush. We accomplish this using a depth first search from seed points along the brush centerline. From each seed point, we find all the vertices with distance less than the brush radius, and set their distance values using the minimum of their current value and their distance from the current seed point. This data is needed only for the duration of a single stroke operation and can be discarded immediately afterward. After the locations of points on boundary are computed, we can use an RBF interpolation technique to obtain lighting offset values as described in Section 4.5.3. The obtained offset values are encoded as the vertex color data in Maya for rendering process as we described above.

4.7 Results and Discussion

We have applied our prototype system to making various stylistic animations. Our system currently runs at interactive rates on a 2.16GHz Intel P4 Core Duo CPU with an NVIDIA GeForce QuadroFX 350M GPU. In editing and previewing the animations, the frame rate ranges from 6 to 20 fps for all the examples in this chapter.

In making facial animation, controlling light and shade on the face is crucial. Figure 4.1 illustrates how effectively and efficiently our algorithms work for this important case. As shown in the figure, even for making a simple facial animation, a 3D head model often creates many unnecessary dark areas, and it is very hard to remove them selectively using conventional lighting control. On the other hand, our approach can eliminate them easily and interactively. Moreover it allows the user to successfully add a variety of effects, each of which dramatically changes the character's impression.

Figure 4.8 demonstrates a typical case where an artist uses our system to make the animation less realistic, but more expressive. Comparing with the animation under conventional lighting (left of Figure 4.8), we note several effects that have been added to the animation. Most obvious is the smoothing and simplification of the moving highlight on the protruding forehead. But also for example, the artist has added a light area to accentuate the jawline; a bright, firm line above the left eye; and delayed emergence of the face into the light, as shown in the right of Figure 4.8. Some of these effects might be

achieved by conventional lighting techniques. However, it is almost impossible to add all of them into the same shot without resorting to frame-by-frame modifications.

Figure 4.10 shows the use of our techniques on an animated character with a highly deforming cape using a moving point light and a fixed directional light. This type of situation can result in light and shade areas that are distracting because they change too rapidly. The animation in the figure demonstrates that our techniques are effective in eliminating such unnecessary shading and in simplifying light and shade to make it suitable for cartoon animation.

Our method also enables local controllability of continuous tone with our intensity brush described in Section 4.5.4. Even when adjusting the continuous tone on this object, our approach allows local tone control, adding a back-light effect around the character's shoulder (see Figure 4.9). We were able to create this animation without modifying the initial lighting setup. However, in cases where the viewpoint and/or lights are moving more dynamically, it may be more difficult to achieve the same effect using our technique.

#Verts	$\lvert\{w_i\}\rvert$	**RBF(solve)**	**RBF(dist)**	**Transfer**	**Total**
2011	68	0.63	5.0	38.8	44.4
8001	114	3.96	19.5	154	178.
31921	311	27.3	88	630	745.

Table 4.1: *Algorithm performance for strokes of various sizes. (All times in milliseconds). #Verts is the number of vertices in the stroke region.* $\lvert\{w_i\}\rvert$ *is the number of unknown weights in the RBF system being solved for, while RBF(solve) is the time taken to solve the linear system. RBF(dist) is the time taken to compute the RBF distance function for calculating* $o_k(\mathbf{p})$*. Transfer is the time taken to transfer vertex data to and from Maya in our plug in.*

In making these animations, we used either of boundary brush or the intensity brush, depending on the type of modification desired. The boundary brush is appropriate when the user wants to specify exactly where the new boundary should lie. If the goal is just to generally make a light or dark shape bigger or smaller, then the intensity brush is more effective. In the examples we determined the size of the paint brushes by experimentation. For example, we chose the width of the boundary brush so that one stroke of the brush includes at least two adjacent vertices of the surface mesh. Similarly, the distance between $\partial\mathbf{C}_0$ and $\partial\mathbf{D}_0$ in Figure 4.4, it is also set to include at least two adjacent vertices of the mesh, which can be accomplished using a slider. The small value of the lighting offset function specified by the intensity brush in Section 4.5.4 is also set empirically. Given the interactivity of our system, results of a particular parameter setting can be seen immediately, so we have not found it burdensome to search for these values via trial and error.

Table 4.1 shows the performance of our current implementation. The computation cost, however, depends on the number of vertices contained in $\mathbf{D}_k$. Since we do not paint very large regions $\mathbf{D}_k$ in practice, this cost seems not to be a serious bottleneck in our system. The most significant part was the basic cost of transferring vertex data between Maya and our plug in. The performance data in Table 4.1 also makes it clear that the algorithm itself is sufficiently fast for interactive editing.

Our prototype system has been made and tested in close collaboration with professional artists in our workplace since the very early stages of development. Initially, we gave

a 20-minute tutorial to the artists. Since our system is implemented as a Maya plug-in, they were able to try it out on their own models immediately. The reaction has been positive: they do seem to find the system capable of producing the desired results easily and quickly. Most of the animations in this chapter were designed with the artists so as to clearly display the capabilities of the proposed technique. Typically animations such as those shown in this chapter take a few hours to complete, which is a drastic improvement over the previous techniques available to the artists. They also claimed that the conventional tricks such as texture animation or modifications to the character's geometry would make it difficult to maintain consistency between different shots with the same character. Therefore, with such conventional techniques, these kind of edits would simply be infeasible on a production schedule.

We also tried to integrate the experimental system into an animation production pipeline. Some of the results are shown in animated feature films. First example is the Tamagotchi feature film: "Tamagotchi: Happiest Story in the Universe!" [93], produced by OLM Digital, Inc. In this film, our system is used to modify undesired shading of a character's lip. The artist could create a desired shading appearance easily in a way similar to the case of making facial animation (see Figure 4.1). Another example is Takashi Murakami's digital animation film: "Kaikai&Kiki" [43]. In making one of the scenes of this film, the conventional lighting method created distracting shaded areas on a character's face due to the rapid movement of the character's head. Similar to the case of deforming cape, our system is also effective in eliminating the undesired light and shade movements to be suited for cartoon animation. These examples demonstrate that our system is capable of improving quality of the animation in professional use. The modified version of this system is presented as a production tool: "Shade Painter" on OLM Digital R&D web site [65].

We feel that there are considerable applications of our algorithms not only in feature films, but also for television animation and even illustrative visualization. In addition, our extension lets the user animate the designed shading with dynamic lighting, which is suited for interactive video games. In these contexts, our system could be useful for artists to pursuit animation of desired shading, since playback using our technique is lightweight and real-time on any modern GPU.

4.8 Summary

In this chapter, we proposed a system for directalbe stylistic depiction of light and shade in 3D animation. According to our interface design framework, we introduced a shading model for local and interactive edits of light and shade by painting directly on 3D objects. Moreover the local edits integrate seamlessly with the conventional global lighting and animate smoothly regardless of the conventional lighting setup used. The animation examples illustrate these advantages over previous methods.

These algorithms, however, are exploratory. There are several things left to accomplish. In our approach, the RBF-based algorithm is used to obtain the rough boundary of the painted shaded area. In addition, we make the assumption that the vertices defining the object will not be added or removed during animation. We do not handle objects that change topology during an animation. We may need a more sophisticated algorithm to obtain a more precise approximation of the painted area. When applying this method to cartoon animation, highlights with very sharp edges are sometimes desired. But our per-vertex offsetting cannot give such a sharp highlight directly. Providing boolean operations as in [4] may be of use here (see Figure 4.11).

©YOUN IN-WAN, YANG KYUNG-IL/Shin Angyo Project 2004

Figure 4.8: *Editing shade and highlights. The animation* (top row) *created using a standard toon shader was modified* (bottom row) *using the techniques described in Section 4.5.4. First the excessive highlight on the forehead was removed using the intensity brush, and then the boundary brush was used to create a light region around the chin, which was otherwise invisible.*

Our method allows us to add locally controllable light and shade, but at the same time conventional lighting control cannot be replaced by our approach. For example, as a very simple case, suppose that we want to move a small rounded highlight on an apple from one location to another. This could be easily accomplished by moving the light source. However, with the approach presented in this chapter, the highlight would not move, but fade off at the original point, and fade in at the destination (see Figure 4.12). This clearly demonstrates a difference between our approach and the conventional one. We believe that these approaches are complementary. Our approach is local, which means not only that it enables local editing, but also that the movement of light and shade is local.

We are currently investigating how to make cast shadows also locally controllable. We believe that a modified version of the approach described here has promise for achieving this. Another challenging avenue of future work would be to transfer designed local shading to different 3D objects. In the case, we need to obtain a good vertex correspondence between different topologies. In particular, such an approach would be essential for reusing the local shading of human faces. In this chapter we have focused on the area of 3D stylized animation. However, this is an important practical area where there is a clear need for new techniques to help bridge the gap between artistic direction and the animator's heavy load. We hope our approach indicates a promising direction to serving such a practical need.

Figure 4.9: *Modifying shading with gradations. Here ShadePainter has been used to make a directional lighting setup appear to be a more dramatic back-lit situation.*

Figure 4.10: *Editing light and shade on a highly deforming object.* (top row) *original frame.* (bottom row) *edited frame. Using the intensity brush, we edited the light and/or dark areas on the deforming cape under rapidly changing lighting conditions.*

Figure 4.11: *Limitation: our method cannot give sharp features. Our per-vertex offsetting results in soft edges. We may use boolean operation [4] to create sharp features.*

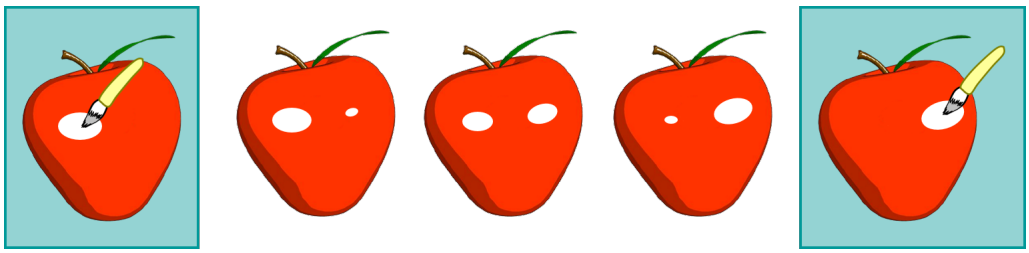


Figure 4.12: *Limitation: our method cannot move a highlight. Edits were made at the two key frames indicated. Highlight would fades off at the original point, and fade in at the destination.*

Chapter 5

Shading Stylization Based on Model Features

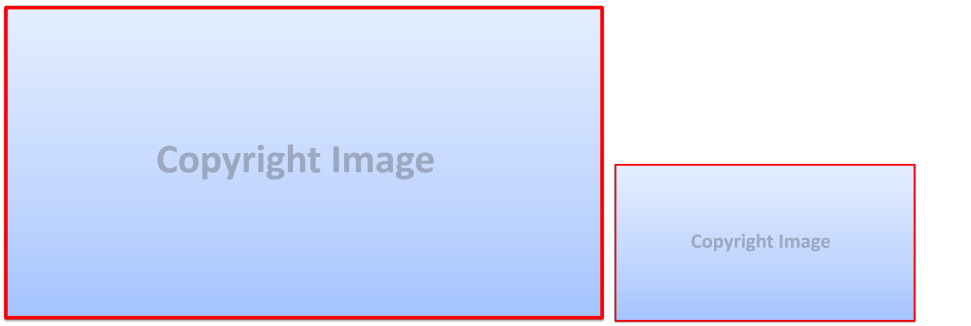
5.1 Overview

The second experiment is to apply our framework to an artist-friendly user interface for shading stylizations based on model features such as surface normals and edges. In this chapter, we focus on how to control feature-dependent lighting effects, rather than the local shading effects described in Chapter 4. In the context of stylized shading design for mechanical objects, it is helpful for the artist to have a method for designing the shading appearance derived from the geometric properties or artistic directions. We present an interactive system that enables straight lighting, edge enhancement, and detailed lighting effects, all of which are commonly used in 2D hand-drawn cartoon animations. In accordance with **Principle 1** (directable shading model for artistic control), we design the shading model which allows the artist to interactively design the these stylized lighting effects for mechanical objects using simple 3D light UIs and appearance-based parameters. The key idea for this directable shading model is to introduce simple lighting transforms and lighting offset based on the model features, such as surface flatness and edge distance field. This complies with **Principle 2** (seamless integration with 3D lighting) in that the proposed stylized shading effects can be manipulated by multiple point light sources. Besides, our system also enables dynamic control over these lighting effects base on a familiar key-framing technique. Thanks to the simple formulations of our algorithms, shading process can be implemented on GPU for real-time preview. Finally, we demonstrate our system by presenting several stylized shading animation results that are effectively designed using our method.

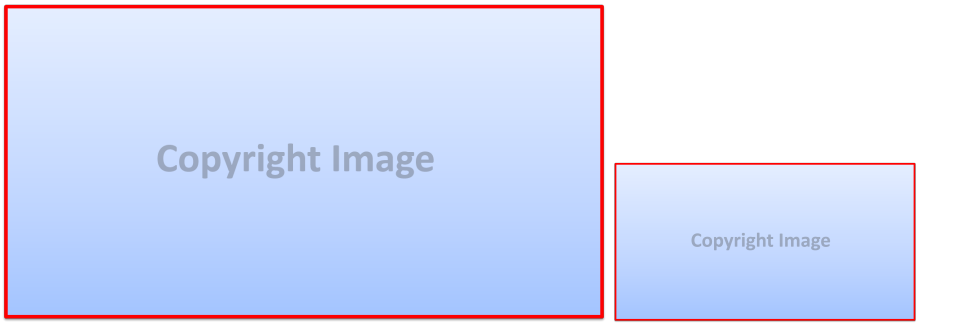
5.2 Introduction

Here we consider the problem of how to provide artists with intuitive and useful control over shading stylization based on model features. First we show 2D hand-drawn examples, where an artist draws typical stylized lighting effects. Figure 5.1 (a) shows straight lighting on windows, which is used to emphasize the flatness of the objects. In Figure 5.1(b), the artist combines the lighting effects derived from the model features. Around the edge of the aircraft, a sharp lighting effect was used to enhance the edge. Another example is a detailed lighting effect, which was used to show that a surface is bumpy. Because of the recent increase in the use of hybrid 2D and 3D models, it is desirable to achieve these lighting effects in a 3D system.

As described in previous Chapter 4, cartoon shading [50] is a standard approach used to design 3D shading in a cartoon style. In this process, the final shading color is obtained



(a)



(b)

Figure 5.1: *Hand-drawn stylized lighting effects. (a) Straight lights on windows. The straight light portrays the flatness and shininess of the windows. (b) Edge enhancement and detailed lighting effects on the aircraft. The sharp lighting is drawn to enhance the edge features. The detailed lighting effect shows that the surface is bumpy.*

using simple 3D lighting processes and 1D color mapping processes. First brightness terms (diffuse and specular) are computed from pre-designed 3D scenes, and then 1D texture maps are used to convert these brightness terms into multi-tone colors. The mechanism is quite simple but effective enough to reproduce a cartoon shading style in a 3D system.

However, the conventional cartoon shading process often creates undesirable shading results. These can be caused by the physical lighting part and the multi-tone color representation in the shading model. Figure 5.2 (a) shows such an example, where the flat polygons were illuminated using a directional light source. The movement of the shaded area (bright area) is discontinuous while the directional light smoothly changes its direction. We refer to this problem as the *discontinuous light appearance problem*, which arises from the constant intensity distribution across the flat surface. To create smooth animation of light and shade, the artist can use point light sources. However, the shaded area will always result in a rounded shape on a flat surface (see Figure 5.2 (b)). While smooth animation is achieved, the rounded shape is not suitable for flat surfaces. As shown in Figure 5.1 (a), it is more helpful if the artist can use straight shapes for lighting on a flat surface. Here we will call this problem: *straight lighting problem*. To solve these issues, artists sometimes use conventional 3D tricks: changing the geometry of the model, animating textures, bump maps, or light maps. However, these require indirect and time-consuming tasks, which are not practical for production work.

Some recent approaches may work well in reducing or partially solving the above prob-

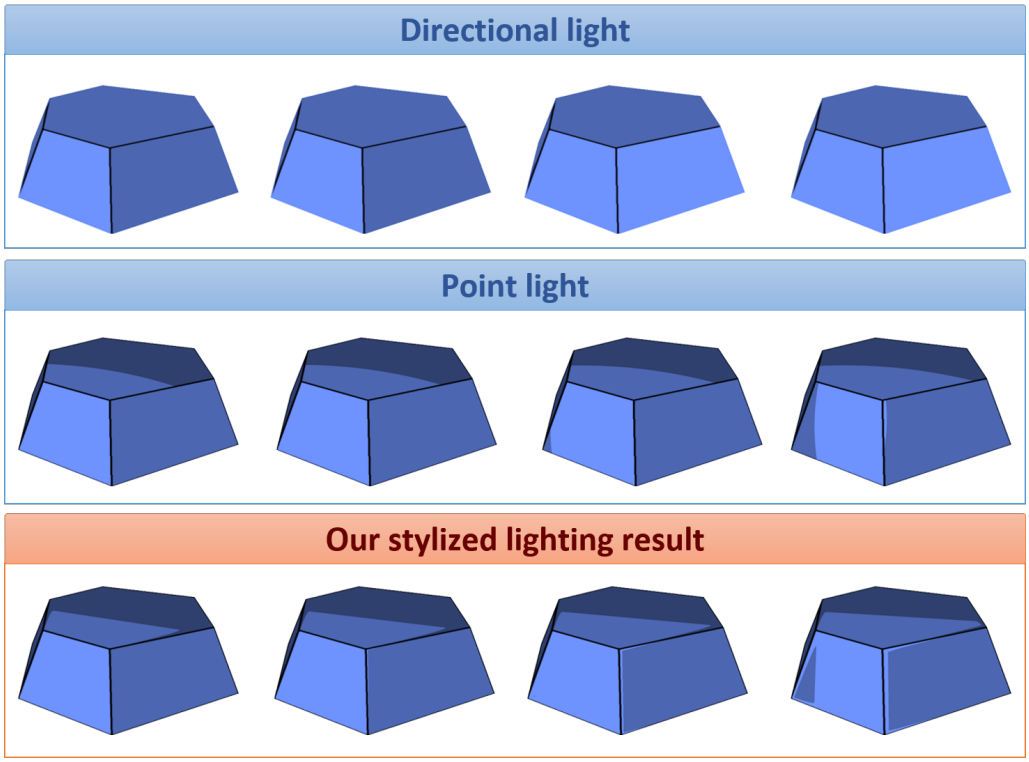


Figure 5.2: *Cartoon shading results with different lighting. (Top row) Directional light. The* discontinuous light appearance problem *occurs from the second frame to the third frame when the directional light smoothly changes its direction. (Middle row) Point light. This type of shading results in smooth animation; however, the* straight lighting problem *is not addressed. (Bottom row) Our stylized lighting effect. Straight shaded areas and edge lighting effects are achieved in smooth light and shade animation.*

lems. Our method for local light and shade described in Chapter 4 and the cartoon highlight shader [4] are helpful for designing the arbitrary shaped lights. However, with these methods it is difficult to enhance the model features, which may become a time-consuming task of adding edge enhancement or emphasizing a bumpiness feature. To enhance the model features, the multi-scale shading method [75] or XToon [11] can be used to control the shading appearance based on geometric features, such as edge enhancement, depth of field, or back lighting. However, these methods support only one additional defined stylization of the original shading result.

Our goal is to develop new 3D lighting methods that can achieve stylized lighting control based on multiple model features. As an initial step toward this goal, we focus on the integration of straight lighting, edge enhancement, and detailed lighting for reproducing the important elements of 2D manual stylized shading styles. To achieve this, we extend the lighting part and texture mapping part of a typical cartoon shading process as follows:

- We introduce a light coordinate system to produce a smooth animation of straight lighting effects. The central idea for straight lighting is to apply a lighting transform to the incoming light vectors based on the surface normal. Our light coordinate system defines the local transformation space according to the designed 3D lighting.
- We extend the conventional texture mapping part to enhance multiple features. In our extension, the threshold values of multi-tone texture are deformed by scalar lighting offset functions. We separately design the offset functions for edge en-

hancement and detailed lighting in a manner whereby the artist can control each effect using intuitive, appearance-based parameters.

5.3 Background

As in Chapter 4, our main focus is on cartoon shading [50], which is commonly used in production environments. Based on the computed brightness terms, this approach effectively reproduces multi-tone shading styles in cartoon animation. Similar to this work, several methods have been proposed to reproduce particular stylized shading styles from 3D scenes. Gooch and Gooch [36] proposed the technical illustration shader for cool-to-warm shading effects. The Lit-Sphere method [87] can describe view-independent tone detail, using a painted spherical environment map. Mitchell et al. [58] presented an illustrative shading style for video game applications. However, these fundamental approaches often lack the ability to control the shading appearance to establish symbolic lighting and enhancements of model features.

Several methods have been developed to change the original lighting result, and hence provide symbolic lighting effects in cartoon animation. DeCoro et al. [28] proposed several stylized shadow effects using image space deformations. Our method described in Chapter 4 allows the artist to paint local lighting effects directly onto a 3D model. Anjyo et al. [4] proposed a cartoon highlight shader which can make various symbolic highlight shapes. While these methods provide creative control over the shape of the lighting, artists may desire further control over the smaller scale appearance.

A number of methods have been developed to design the shading based on geometric or scene properties, providing such control. For instance, the multi-scale shading method [75] and the 3D unsharp masking method [73] can accentuate the edges or the silhouettes of a target model. Several methods focus on the geometric deformation with 2D texture inputs, including bump mapping [16], displacement mapping [24], and relief mapping [64]. More complex extension is obtained with 2D tone controls. Barla et al. [11] proposed X-Toon, where a 2D texture is used to integrate additional controls, including depth of field, back lighting effects, and diffuse-dependent specular effects. These techniques allow the artist to control the appearance of a specified model feature, but they do not permit the integration of multiples features.

In contrast to related work, our approach provides a simple interactive method for stylized shading design, where a symbolic lighting effect (straight lighting) is seamlessly integrated with enhancements based on multiple model features (edge enhancement, and detailed lighting).

5.4 User Interaction

In our approach, shading stylizations can be controlled with three proposed lighting effects: straight lighting effect, edge enhancement effect, and detail lighting effect. In the lighting design process, their parameters can be dynamically modified to check results in real-time. User control parameters of our shading model are summarized in Table 5.1.

To provide convenient and familiar methods for artists, we propose the overall process in the following manner. The artist begins by making an initial 3D scene, including the geometries of the models and their animation settings, using a conventional 3D software tool. In the next step, the artist designs the straight lighting effect by manipulating a 3D

Effect	Parameter
Straight lighting (Figure 5.3)	Orientation of the straight lighting · Translation · Rotation
Edge enhancement (Figure 5.4)	Curved edge appearance · Width · Height
Detail lighting (Figure 5.5)	Wavy shape · Strength · Frequency

Table 5.1: *User control parameters of our shading model.*

light UI (see Figure 5.3). This allows the artist to control lighting shape with translation and rotation operations. The designed lighting can be further controlled using edge enhancements and detailed lighting effects. Figure 5.4 shows the user interface for edge enhancement effects. The artist can adjust the curved shape of the edge appearance using the width and the height parameters. Our system also provides a way to control the wavy shape of a detail lighting effect using the strength and the frequency parameters for its wave form (see Figure 5.5). For both small scale stylizations, the artist can adjust the enhancements using a few appearance-based parameters. All the lightning and parameters are designed using key-frame editing, which efficiently produce interpolated sequences. By relying on our simple algorithms, the artist can preview the result in real-time.

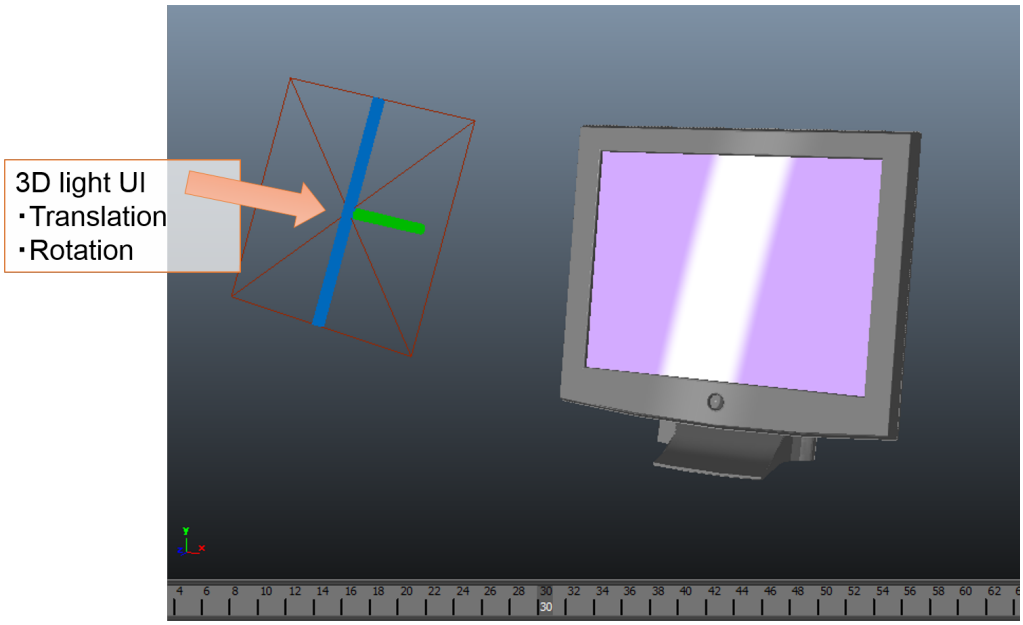


Figure 5.3: *User interface for straight lighting effect. The user controls the lighting shape by manipulating the 3D light UI.*

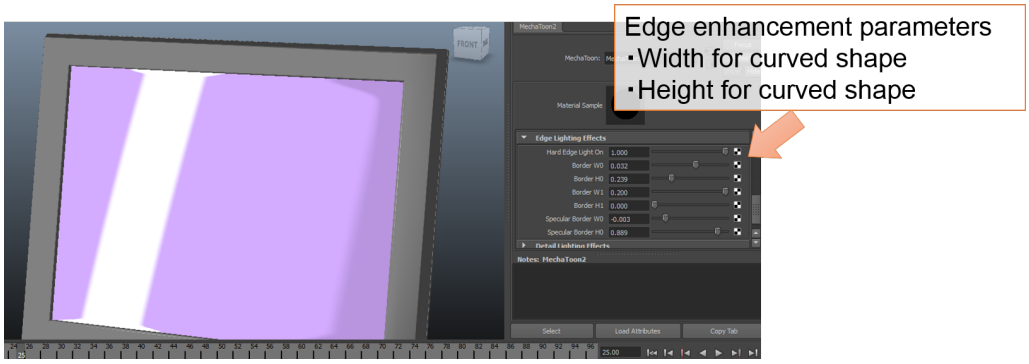


Figure 5.4: *User interface for edge enhancement effect. The curved shape of the edge appearance can be controlled using the width and the height parameters.*

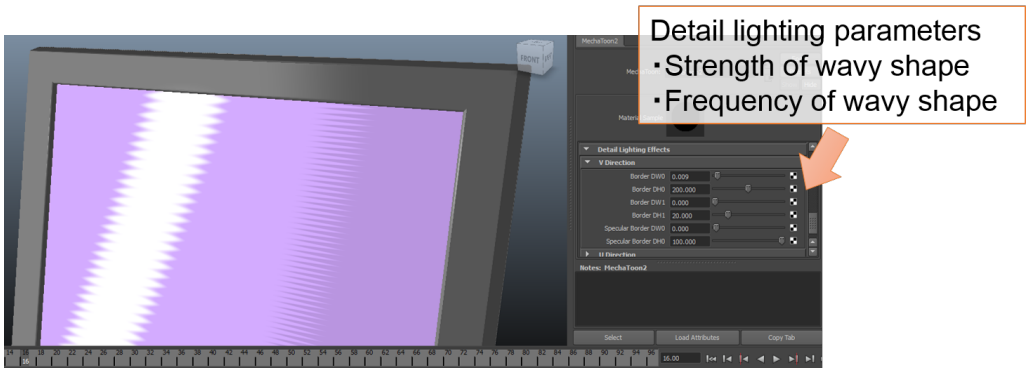


Figure 5.5: *User interface for detail lighting effect. The wavy shape of the detail lighting effect can be controlled using the strength and the frequency parameters.*

5.5 Light Shape Control

In this section, we describe how to design a symbolic lighting effect in the case of the straight lighting in our context. We first consider the conventional cartoon shading process and how to achieve the stylized shape of light and shade. Here we use the thresholded 1D color mapping to define each shaded area $\mathbf{D_i}$ for the given threshold value δ_i:

$$\mathbf{D_i} := \{\mathbf{p} \in \mathbf{S} \mid \delta_{i-1} \leq I_d(\mathbf{p}) < \delta_i\} \quad (i = 1, 2, ..., m), \tag{5.1}$$

where $\mathbf{p} \in \mathbf{S}$ is an arbitrary point of the surface $\mathbf{S}$, $I_d(\mathbf{p}) := \mathbf{L}(\mathbf{p}) \cdot \mathbf{N}(\mathbf{p})$ is the diffuse term computed from the light vector $\mathbf{L}$ and the surface normal vector $\mathbf{N}$, and m is the number of tones, which is typically set to 2 or 3. As described in Section 5.2, this simple mechanism often results in discontinuous light appearance and straight lighting problems on flat surfaces (see Figure 5.2). To solve these problems caused by the physical lighting process, we apply a lighting transform to the light vector $\mathbf{L}$ to achieve the straight lighting effect in a similar manner to the highlight vector transform proposed by Anjyo et al. [4]. The difference is that we introduce a light coordinate system to dynamically define the transformation space, whereas while their method relies on the static tangent space.

5.5.1 Light Coordinate System

In our lighting transform approach, we use a point light for the initial lighting, because the point light source produce a continuous light appearance on a flat surface (see Fig-

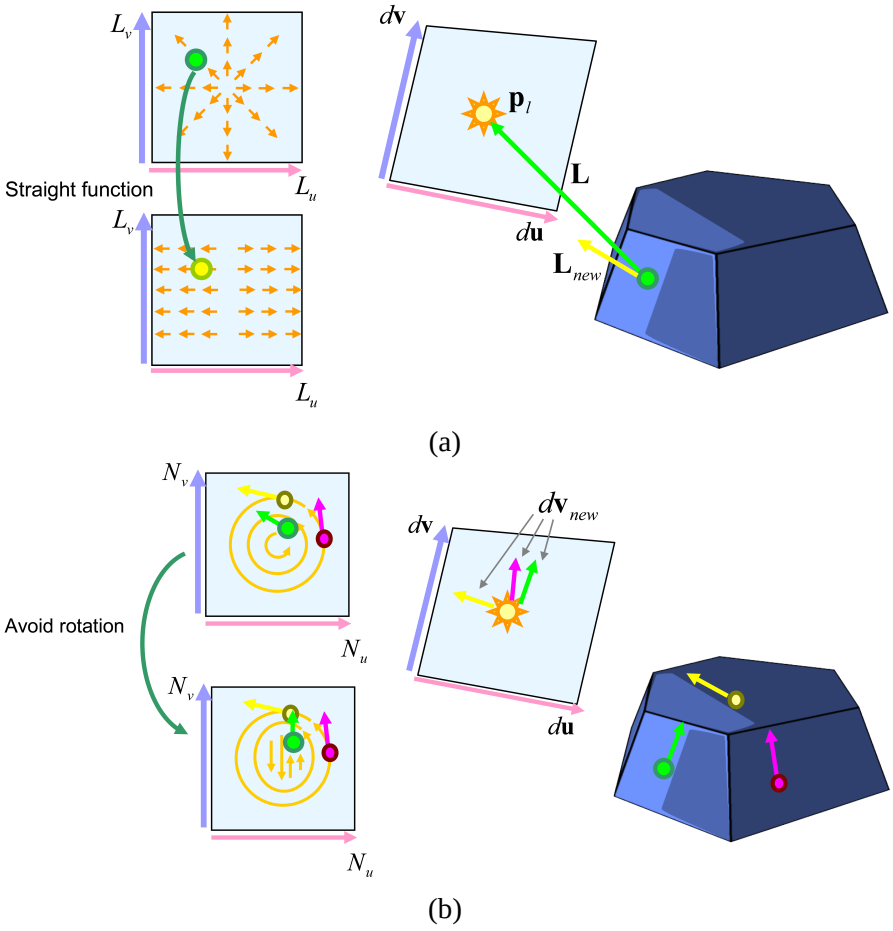


Figure 5.6: *Light coordinate system for the initial lighting design. (a) Light shape control with the light coordinate system. The light vector is transformed through the straight function according to the projection coordinates (L_u, L_v). (b) Automatic transform direction control. The new transformation axis $d\mathbf{v}_{new}$ is assigned depending on the surface normal direction (N_u, N_v).*

ure 5.2 (b)). With the location of point light source $\mathbf{p}_l$, we introduce a light coordinate system to specify the transform orientation (see Figure 5.6). We use additional coordinate axes ($d\mathbf{u}$, $d\mathbf{v}$, $d\mathbf{w}$) for light shape control. With these elements, we can introduce a different representation of the light vector $\mathbf{L}$:

$$\mathbf{L}(\mathbf{p}) = L_u(\mathbf{p})d\mathbf{u} + L_v(\mathbf{p})d\mathbf{v} + L_w(\mathbf{p})d\mathbf{w}, \tag{5.2}$$

where L_u, L_v, and L_w are the projection coordinates which are computed from $L_u := \mathbf{L} \cdot d\mathbf{u}$, $L_v := \mathbf{L} \cdot d\mathbf{v}$, and $L_w := \mathbf{L} \cdot d\mathbf{v}$. Based on this representation of the light vector, we can apply a vector transform to control the light shape. To create a straight lighting effect on a flat surface, we define a straight function f_{st} as:

$$f_{st}(\mathbf{L}) := L_u(\mathbf{p})d\mathbf{u} + (1-\alpha)L_v(\mathbf{p})d\mathbf{v} + L_w(\mathbf{p})d\mathbf{w}, \tag{5.3}$$

where the directional scaling term $1-\alpha$ is applied to the projection coordinate $L_v(\mathbf{p})$. The straight function f_{st} makes the light vector straight along the $d\mathbf{v}$ direction if α approaches 1. As shown in Figure 5.2, the point light vector transformed by the straight function successfully produces a straight lighting shape. In our system, the artist can control the transform direction by manipulating the position and rotation of a 3D line-shaped light UI, which is easily converted into the elements of the light coordinate system ($\mathbf{p}_l$, $d\mathbf{u}$, $d\mathbf{v}$, $d\mathbf{w}$).

5.5.2 Transform Orientation Control

In the previous section, we described a simple case where the transform orientation was constant $d\mathbf{v}$. Here we consider a transform orientation control for the more general case of multiple polygons. From our observations, artists typically use different transform orientations for different polygons. The obvious solution is local control, whereby the animator can set the transform direction for each flat polygon. However, such local controls would require a time consuming key-framing process to create the animation. To reduce the artist's workload, we introduce automatic transform orientation control. Here we try to satisfy the following requirements in our transform orientation control:

- The transform orientations are perpendicular to the surface normal vector (Requirement 1).
- Stable and coherent motion of the transform orientation during the animation (Requirement 2).

For these practical requirements, we provide a method for automatically defining coordinate axes ($d\mathbf{u}$, $d\mathbf{v}$, $d\mathbf{w}$) depending on the surface normal direction $\mathbf{N}$ (see Figure 5.6 (b)). Similar to Equation 5.3, we can introduce a representation of the surface normal vector for the given coordinate axes ($d\mathbf{u}$, $d\mathbf{v}$, $d\mathbf{v}$):

$$\mathbf{N}(\mathbf{p}) = N_u(\mathbf{p})d\mathbf{u} + N_v(\mathbf{p})d\mathbf{v} + N_L(\mathbf{p})d\mathbf{w}, \tag{5.4}$$

where N_u, N_v, and N_w are defined in the same way as L_u, L_v, and L_w in Equation 5.3. For the N_u and N_v, we first consider a new transform orientation $d\mathbf{v}_1$ that satisfies Requirement 1:

$$d\mathbf{v}_1 = normalize(N_v d\mathbf{u} - N_u d\mathbf{v}), \tag{5.5}$$

where $d\mathbf{v}_1$ is the vector perpendicular to $\mathbf{N}$ and $d\mathbf{w}$. This definition satisfies Requirement 1; however, Requirement 2 is not satisfied. For instance, suppose that an animator rotates the light along the $d\mathbf{u}$ direction. This results in an undesirable rotation of shaded area, whereby the angle between $d\mathbf{w}$ and $\mathbf{N}$ approaches 0°. To avoid such undesirable rotations, and to satisfy Requirement 2, we integrate stable behavior into the definition in Equation 5.4 by rewriting the definition of Equation 5.5 as:

$$d\mathbf{v}_{new} = normalize(\phi(N_u, N_v)N_v d\mathbf{u} - N_u d\mathbf{v}), \tag{5.6}$$

where the scaling function $\phi(N_u, N_v)$ is applied to the projection coordinate N_v. We designed the scaling function $\phi(N_u, N_v)$ as follows:

$$\phi(N_u, N_v) := \begin{cases} 0 & \text{if } \|(N_u, N_v)\| < r_1 \\ \frac{\|(N_u, N_v)\| - r_1}{r_2 - r_1} & r_1 \le \|(N_u, N_v)\| < r_2 \\ 1 & \text{otherwise} \end{cases} \tag{5.7}$$

where r_1 and r_2 are the user-given parameters satisfying that $0 < r_1 < r_2 < 1$. With this scaling function, our method can seamlessly blend $d\mathbf{v}_1$ ($\phi(N_u, N_v) = 1$) and $d\mathbf{v}$ ($\phi(N_u, N_v) = 0$).

5.6 Threshold Offset to Enhance Multiple Features

In this section, we consider how to apply multiple enhancements to the designed straight lighting result. Barla et al. [11] used a 2D texture to extend the conventional texture

mapping process to enhance one additional feature. Inspired by this approach, we design an extension of the 1D color mapping process to enhance of multiple features. In our approach, we use procedural scalar lighting offset functions to modify the threshold values of multi-tones. Figure 5.7 shows our lighting offset process. The final threshold value δ_i^{new} is computed according to:

$$\delta_i^{new} := \delta_i + o_i^e(E) + o_i^d(D), \tag{5.8}$$

where $o_i^e(E)$ and $o_i^d(D)$ are the scalar offset functions of the edge lighting effect and the detailed lighting effect, respectively. We use these two offset functions in our method; however, we can easily provide additional enhancements if required. Our system runs at interactive rates with the two detailed features for three shaded areas and one highlight area, as described later.

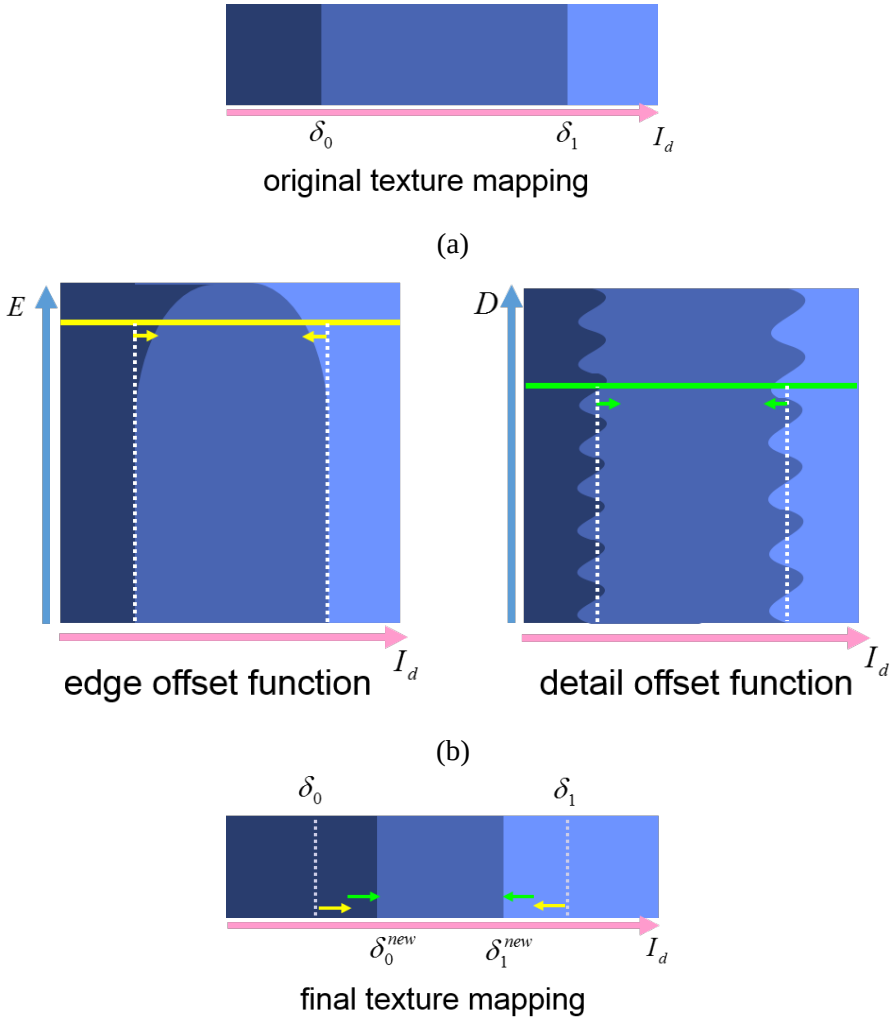


Figure 5.7: *Lighting offset for multiple enhancements. (a) The original texture mapping assigns the three tone colors according to the Lambertian diffuse term I_d. (b) The edge offset function and detailed offset function deform the threshold values of the multi-tones. (c) The final texture mapping enhances the edge features and the detail features through the combined offset functions.*

5.6.1 Edge Enhancement

Here we describe a method for providing intuitive control over the edge appearance with our lighting offset approach. Edge enhancement typically uses a sharp curved lighting effect around the edge of the object (see Figure 5.1 (b)). To achieve this, we design the edge offset function o_i^e that is a function of the edge intensity value and a few appearance-based parameters (width control and height control). The edge intensity value is referenced from the image space edge field computed from the 3D scene. With the input edge intensity value, the final edge appearance can be controlled using the defined appearance-based parameters.

Image Space Edge Field

To specify a deformation space, we need to define an edge field on the target 3D model. While our method is independent of the choice of the edge intensity representation, we chose the image space edge field, because it can be dynamically extracted from screen-space information, which provides an efficient way to control the edge appearance in real-time.

To compute the image space edge field, we use an approach similar to the ray-tracing algorithm for the NPR-Line [21], where the feature lines are extracted based on the parametric distance of sampling points. Figure 5.8 shows an overview of our image space edge detection algorithm. The parametric distance is computed from the image space property vectors:

$$d(\mathbf{x},\mathbf{y}) := ||\mathbf{P}(\mathbf{x}) - \mathbf{P}(\mathbf{y})||, \tag{5.9}$$

where $\mathbf{x}$ and $\mathbf{y}$ are the sampling points, $\mathbf{P}(\mathbf{x})$ is the surface property vector at $\mathbf{x}$, and $d(\mathbf{x},\mathbf{y})$ is the parametric distance between $\mathbf{x}$ and $\mathbf{y}$. Figure 5.9 describes the overall

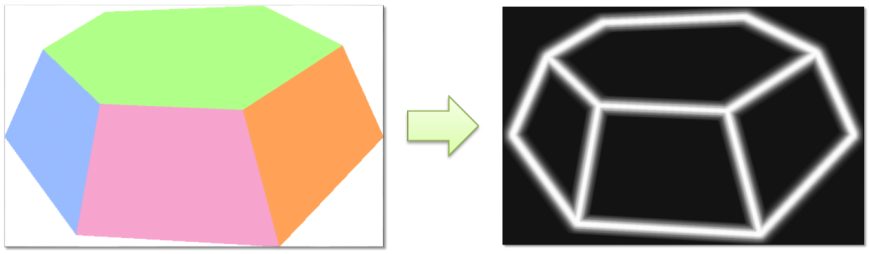


Figure 5.8: *Image space edge detection. (Left) A scene rendered with color mapped surface normals. Each surface normal is used as part of an image space property vector in our edge detection algorithm. (Right) Our image space edge detection algorithm searches for discontinuities in the property vectors to compute the image space edge field.*

process of computing an edge intensity at a sampling pixel $\mathbf{x}$. Based on the parametric distance definition, a nearby pixel $\mathbf{y}$ around $\mathbf{x}$ can be classified as a similar pixel ($d(\mathbf{x},\mathbf{y}) \leq c$) or a dissimilar pixel ($d(\mathbf{x},\mathbf{y}) > c$) using a threshold value c. We use the minimum distance $d_E(\mathbf{x})$ between the sampling pixel $\mathbf{x}$ and the dissimilar pixels to define the edge intensity $E(\mathbf{x})$:

$$E(\mathbf{x}) := \max(0, 1 - t d_E(\mathbf{x})), \tag{5.10}$$

where t is the thickness control parameter for an image space edge field. To reduce the computational expense, we approximate the edge intensity using sparse sampling.

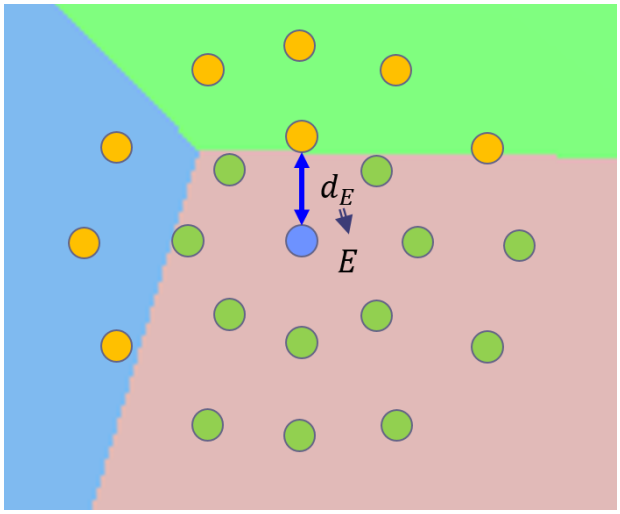


Figure 5.9: *Edge intensity at a sampling pixel (the blue point). Based on the parametric distance, each nearby pixel is classified as a similar pixel (the green points) or a dissimilar pixel (the red points). The minimum distance d_E between the sampling pixel and the dissimilar pixels is used to compute the edge intensity E.*

The precision of the edge intensity depends on the number of sampling points M. We use $16 \leq M \leq 32$, which we have found provides a trade-off between precision and efficiency.

Edge Offset Function

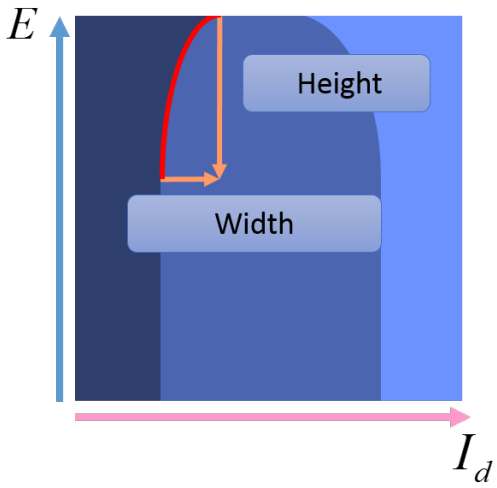


Figure 5.10: *Lighting offset with edge offset functions. Our edge offset functions deform the threshold values with appearance-based parameters (width control and height control).*

With the definition of the image space edge field, we design the edge offset function to deform the threshold values using a few simple appearance-based parameters (see Figure 5.10). For the computed the edge intensity value E, the edge offset function $o_i^e(E)$ is defined as:

$$o_i^e(E) := \beta_i^w(1 - \sin(\text{acos}(\frac{E - \beta_i^h}{1.0 - \beta_i^h})), \tag{5.11}$$

where β_i^w (for width control) and β_i^h (for height control) are the user-specified parameters for controlling the curved shape of the edge appearance.

5.6.2 Detailed Lighting Effect

Another application of our lighting offset function approach is detailed lighting effect. Here we describe how to design effective jagged highlights and shaded areas using a few intuitive parameters. As shown in Figure 5.1 (b), typical jagged lighting has a wavy shape used to depict a bumpy surface. With this observation in mind, we empirically design the following offset function to achieve a detail lighting effect.

Detail Offset Function

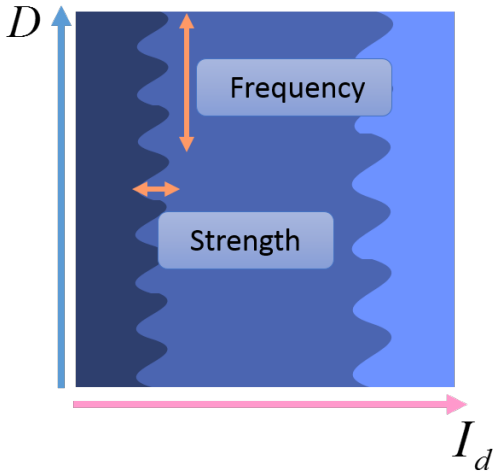


Figure 5.11: *Lighting offset with detail offset functions. Our detail offset functions deform threshold values using appearance-based parameters (strength and frequency).*

Figure 5.11 shows how our detail offset function deforms threshold values. Let D be the detail feature parameter (vertical axis in the figure) to specify a deformation space. For the given detail feature parameter D, we define the detail offset function as:

$$o_i^d(D) := \gamma_i^s \sin(\gamma_i^f D), \tag{5.12}$$

where γ_i^s and γ_i^f are the user-specified parameters to control the strength and frequency of the wavy shape. While any kinds of parameter can be used for D, we use one of the object-space texture coordinates v, because the artist is familiar with the layout design of (u,v) coordinates. By changing γ_i^s and γ_i^f, we can easily adjust the appearance of the detailed lighting effect.

5.7 Implementation

To integrate the proposed methods into an existing 3D shading design process, we implemented our stylized shading algorithms as a Maya plug-in. Our implementation makes use of the highly customizable features of this plug-in. To reduce the computational expense, we implemented the rendering algorithms using Maya's hardware shader functionality, which allows the shader code to be written using standard OpenGL and GLSL.

Since the tessellation of mechanical objects is generally not suited for per-vertex lighting, we need to calculate the lighting process using a pixel shader.

Straight Lighting Effect

To implement the straight lighting effects, we provided a simple 3D line-shaped light UI to specify the position and the orientation of the light coordinate system. These data and the parameters for the shape control are transferred to the pixel shader, and then the lighting transforms are applied to the per-pixel light vector.

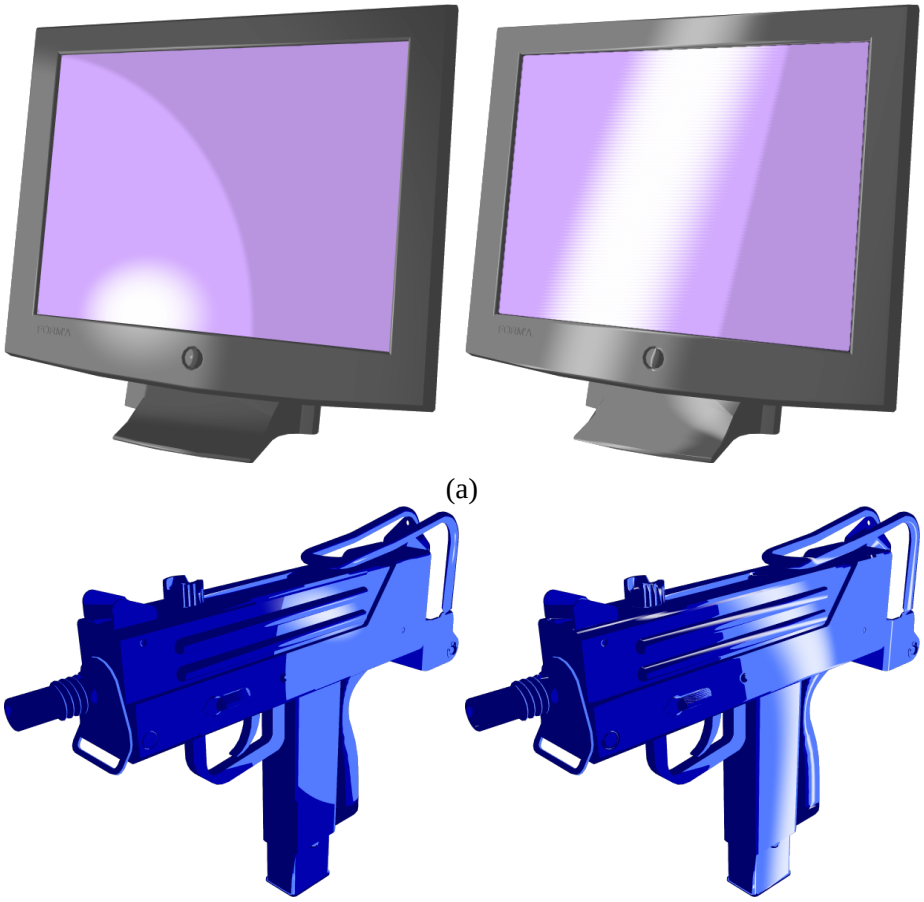
Image Space Edge Field

The most time-consuming part of our rendering algorithms is computing the edge distance field. To perform this process in real-time, we compute the per-pixel edge intensity on the GPU using multiple passes. In the first pass, the target 3D model data are rasterized into three property textures: the surface depth, the surface normal, and the surface material ID. In the second pass, we use the pixel shader to compute the per-pixel edge intensity values of these property textures. Finally, our system merges these edge fields by choosing the maximum edge intensity value. The computed edge field is referenced by the edge offset process, which is implemented as a part of lighting process in the pixel shader. In our prototype system, we can compute an edge field at interactive rates with an image resolution of 512×512 pixels.

5.8 Results and Discussion

We applied our prototype system to create various stylistic animations. Our system runs at interactive rates on a 2.16GHz Intel P4 Core Duo CPU with an NVIDIA GeForce QuadroFX 350M GPU. In editing and previewing the animations, the frame rate was in the range 6 - 20 fps for all of the examples in this chapter.

The continuous appearance of straight lights is crucial for creating 3D animations of mechanical objects composed of multiple flat polygons, and suggest that the surfaces are flat and shiny. Figure 5.12 (a) demonstrates how effective and efficient our algorithms are for such requirements. As described in Section 5.2, even for a simple object such as a computer monitor, conventional lighting results in the discontinuous light appearance problem and the straight lighting problem on flat surfaces. In contrast, our stylized lighting method enables interactive design of the straight lighting effect by manipulating a 3D light source UI. Moreover, the designed lighting can be further controlled using edge enhancement and the detailed lighting effects, which can give the viewer different impressions or nuances of the objects. (see the right of Figure 5.12 (a)).



(a)

(b)

Figure 5.12: *Typical lighting examples. (a) A computer monitor model including multiple flat surfaces. The left image is illuminated using a point light source and the right is designed using our method. Our straight lighting and detailed lighting effects portray the flatness as well as the bumpiness of the surface. (b) A model of a gun. The left image was illuminated using a point light source and the right is designed using our method. Our straight lighting effect produced the straight shaded areas on the flat polygons, while rounded shaded areas were preserved on the smooth surface.*

In creating animations with the gun model shown in Figure 5.12 (b), lighting control is more difficult due to the complexity of the geometry. Compared to the results generated using a point light source (the left of Figure 5.12 (b)), our method allows the artist to design the straight lighting effect easily and interactively. Note that the rounded shaded areas are preserved on the smooth surface. This would be difficult to achieve using conventional lighting control without locally controlling the lighting on a per-polygon basis.

Edge enhancement and detailed lighting are important when drawing mechanical objects, as shown in Figure 5.1 (b). Figure 5.13 demonstrates an example where an artist used our system to design such expressive shading styles for a mechanical object. The conventional lighting was limited to a simple shading appearance, and it was difficult to add detail. In contrast, our approach enabled interactive control of the detailed appearance (using edge enhancement, and detailed lighting effects) by adjusting a few simple appearance-based parameters.

Figure 5.14 shows another example where our tools were used to produce a crystal appearance. In this example, it would be desirable to design a straight lighting effect to

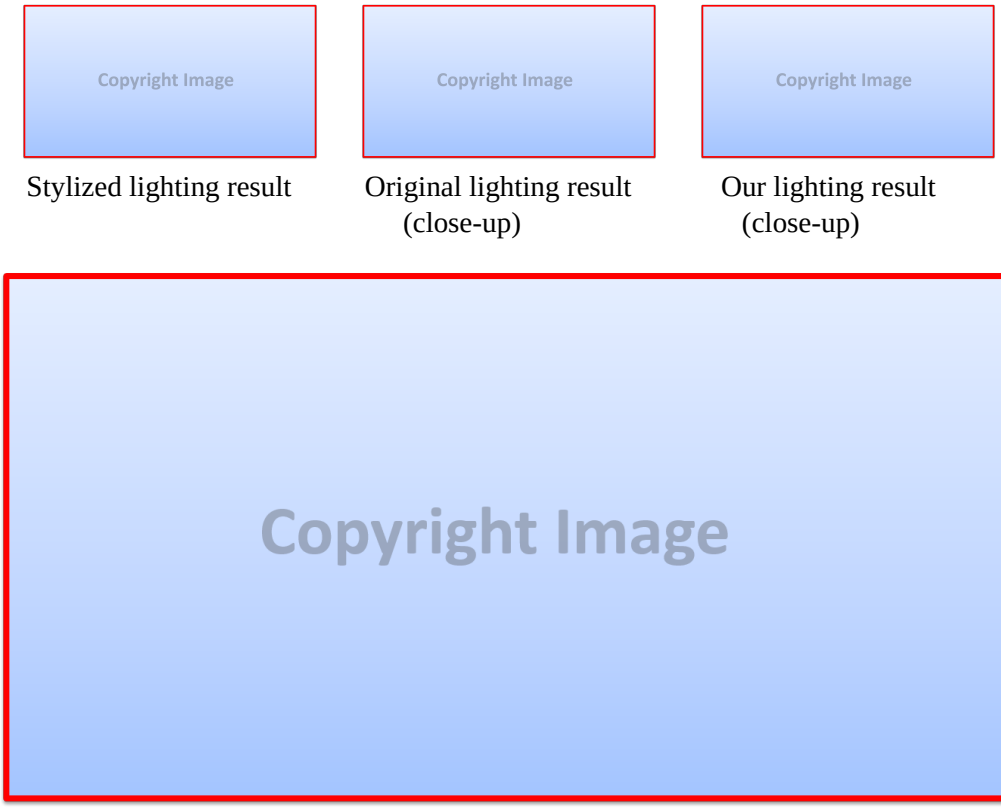


Figure 5.13: *Edge enhancement and detailed lighting effects on an aircraft. Top row: our stylized lighting result (left), close-up of the original lighting result using a point light (middle), and close-up of our stylized lighting result (right). Middle row and bottom row: a comparison between the original lighting result and our lighting result. The sharpness and the bumpiness of the aircraft are emphasized using our edge enhancement and detailed lighting effects.*

emphasize the shiny surface properties of the crystal. However, a discontinuous light appearance would be problematic if using a directional light source, as shown in the top row of Figure 5.14. The bottom row of Figure 5.14 demonstrates that our straight lighting effects allows a continuous light appearance for flat surfaces, making it suitable for the images of crystals. In addition, our edge enhancement effects allows the artist to design sharp lighting on the edges.

While these examples do not include any deforming object, our techniques can be applied to a highly deforming object (see Figure 5.15). In this case, it would be time-consuming to compute the edge field in the object space because the geometry changes rapidly. On the other hand, our image space algorithm extracts the edge features interactively due to the mechanism independence of the complexity of the target geometry.

Our prototype system has been tested by professional artists in our work place. Most of the animations in this chapter were easily and quickly designed, taking only a few hours to complete. In addition, we were able to demonstrate the system and examples to many artists. Their reactions were positive: they felt that the methods were effective for reproducing typical lighting styles that are commonly used in hand-drawn cartoon images. Furthermore, they commented that conventional lighting processes are not capable of producing such stylized lighting results, and that manually drawing the lighting frame by frame is the most reliable way to achieve these effects using existing methods. We

Figure 5.14: *Straight lighting effects and edge enhancements for crystal appearance. (Top row) The original lighting result using a directional light. (Bottom row) Our lighting result using the straight lighting and the edge enhancement effects. First, a discontinuous light appearance was replaced by our smooth straight lighting effects; then, the sharp lighting was designed using the edge enhancement.*

feel that there is considerable potential for our methods to replace these time-consuming tasks for artists.

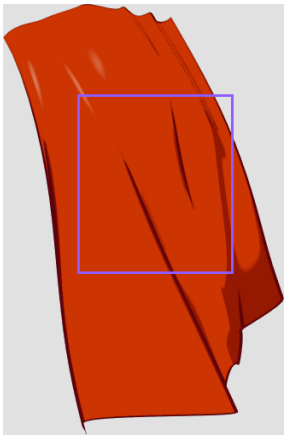
5.9 Summary

In this chapter, we presented a system for shading stylization based on model features, including straight lighting, edge enhancement, and detailed lighting effects. According to our interface design framework, we extended the simple cartoon shading model, providing additional control over specific model features based on the lighting transforms and the lighting offsets. Moreover, our shading stylization can be seamlessly integrated with the conventional lighting techniques that use point light sources. The animation examples illustrate the advantages of our system over existing methods.

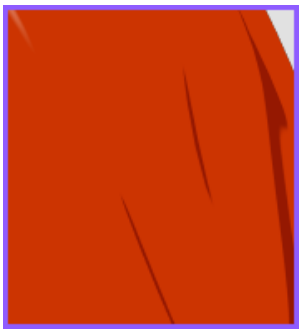
Some additional capabilities are required to realize the full potential of our system in practice. For example, our stylized lighting methods only permit global controls to change the shading appearance. As shown in Section 5.8, our methods allow the artist to quickly design plausible lighting results to generate expressive shading appearances. However, the artist may require local controls over the highlights and shading of each surface patch for fine-tuning (see the left image of Figure 5.16). Besides, small scale controls for complex edge appearance would also be effective for further adjustment of the character's appearance (see the right image of Figure 5.16). Establishing such local controls with suited interactive design process is an important area of future work, and is planned to integrate the experimental system into an animation production pipeline.

Our image space edge field is effective for interactive design of effects; however, for offline rendering, we may require a more precise approach to obtain a temporally coherent edge field. In restricting ourselves to 3D cartoon animation, we feel that the current prototype system has the capability to create good results with high resolution images.

We are also investigating stylized shadow effects. The main challenge is thus to establish a generalized framework and interface that allows the artist to easily create a number of simultaneous light effects using the global lighting, local lighting, and shadows.



Stylized lighting result



Original lighting result
(close-up)

Our lighting result
(close-up)

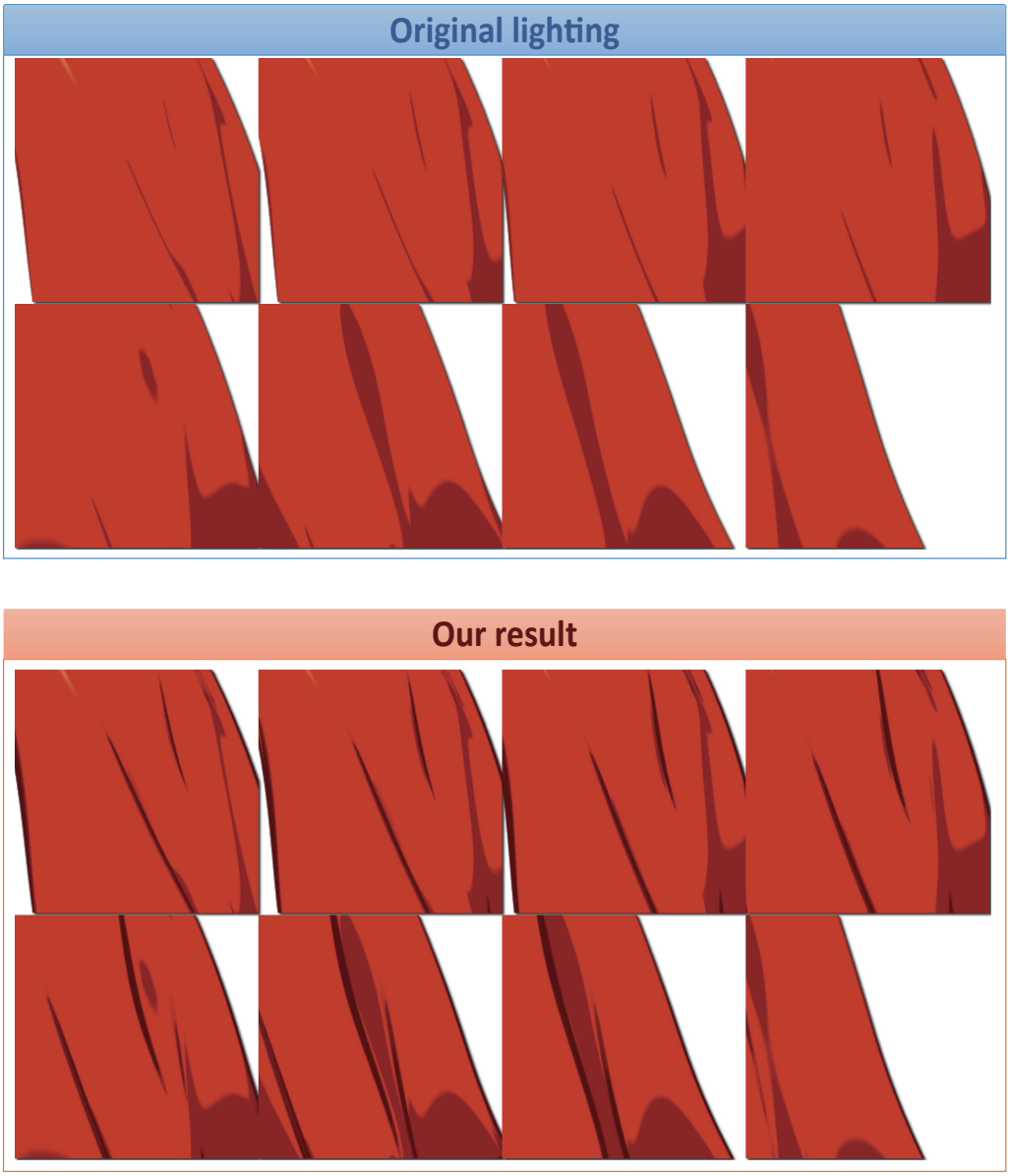


Figure 5.15: *Edge enhancement for a highly deforming object. The silhouettes of the deforming cape were emphasized using our edge enhancement.*

Figure 5.16: *Limitations of our method. (Left) Our stylized methods only permit global controls. (Right) Our edge enhancement cannot handle a complex shape beyond our simple formulation.*

Chapter 6

Practical Shading Model for Expressive Shading Styles

6.1 Overview

The third experiment is to apply our framework to artist-friendly user interface for practical shading model for stylized shading styles. In this chapter, we focus on how to design overall appearance with expressive shading styles whereas the first and second methods are limited to simple shading tones. For global control of stylized shading appearance, the Lit-Sphere shading model proposed by Sloan et al. [87] is state-of-art method for emulating expressive stylized shading styles. Assuming that stylized shading styles are described by view space normals, this model produces a variety of stylized shading scenes beyond traditional 3D lighting control. However, it is limited to the static lighting case: the shading effect is only dependent on the camera view. In addition, it cannot support small-scale brush stroke styles. To address these issues, we propose an extension of the Lit-Sphere shading model that allows the artist to design expressive shading styles for dynamic lighting. In accordance with **Principle 1** (directable shading model for artistic control), we design the directable shading model which provides intuitive painting process for the Lit-Sphere shading and appearance-based controls for prominent features. The key idea for the directable shading model is to reformulate the Lit-Sphere shading model using light space surface normals. Thanks to the light space representation, our shading model addresses the issues of the original Lit-Sphere approach, and allows artists to use a light source to obtain dynamic diffuse and specular shading. Besides, the shading appearance can be refined using stylization effects including highlight shape control, sub lighting effects, and brush stroke styles. Our extension complies with **Principle 2** (seamless integration with 3D lighting) in that all the proposed shading effects can be controlled by a single global directional or point light source. Besides, the designed shading style results in coherent animation, to which 3D object deformation can be applied. Finally, our algorithms are easy to implement on GPU, so that our system allows interactive shading design.

6.2 Introduction

Stylized rendering techniques in computer graphics have been widely used to emulate shading styles of artists. Among them, cartoon shading is popular in a variety of production software, including Autodesk® Maya® and 3ds Max®. This approach is based on computed illumination, and effectively reproduces the abstracted shading styles of

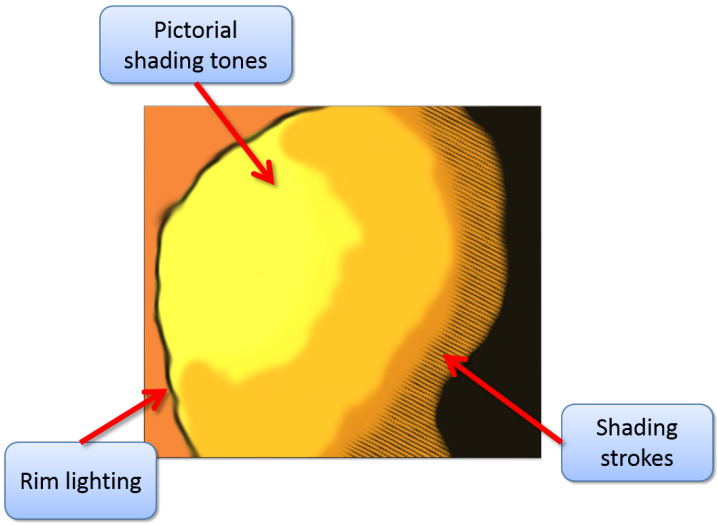


Figure 6.1: *Typical hand-drawn shading style. Pictorial shading tones are enhanced with rim lighting effects and shading strokes.*

comics or cartoons. However, the shading appearance is limited to simple shading tones, whereas hand-drawn shading styles may have rich variations as follows.

Figure 6.1 shows a typical hand-drawn shading style. In this scheme, the artist designs complicated shading tones in the pictorial space, which cannot be simply described by the typical diffuse and specular terms. We term such shading tones *pictorial shading tones*. These shading tones can be enhanced using the following stylization effects. The character silhouette is accentuated by a sharp lighting effect, which is commonly called *rim lighting* (also known as back lighting). In addition, the boundary of the shading tones are often drawn with brush strokes, which we will refer to as *shading strokes*. Here we will call these kinds of effects: *secondary stylizations*.

In this chapter, we consider how to design such stylized shading with dynamic 3D lighting. As with the static lighting case, the Lit-Sphere shading model [87] is attractive because it can deal with *pictorial shading tones* (see Figure 6.2). Versions of this approach have been successfully used in commercially available software tools such as MudBox® and ZBrush®. However, the Lit-Sphere model is limited to this static lighting appearance; the resulting shading will not be dynamically affected by the lighting since the shading effect is totally dependent on the view space normals (see Figure 6.3). In addition, the generation of small-scale stylizations such as *shading strokes* using this method may result in unwanted visual artifacts (see Figure 6.4).

We propose to extend the Lit-Sphere model with dynamic lighting and shading stylizations while preserving pictorial shading tones. To achieve this, we introduce the concept of the light space normals, which enhance the Lit-Sphere model by including the following new features:

- Light-dependent diffuse and specular behavior.
- Secondary stylizations including highlight shape control, rim lighting, and shading strokes.
- Coherent animation of the shading styles, to which 3D object deformation can be applied.

The remainder of this chapter is organized as follows. First, we briefly summarize re-

lated work in Section 6.3 and then we explain how to extend the Lit-Sphere approach for dynamic diffuse and specular behavior in Section 6.5. In Section 6.6, we describe how the secondary stylizations can be combined with the original shading tones. Combining these techniques, we demonstrate a variety of shading appearances in Section 6.8. Finally, we discuss the limitations and possible extensions of our approach in Section 6.9.

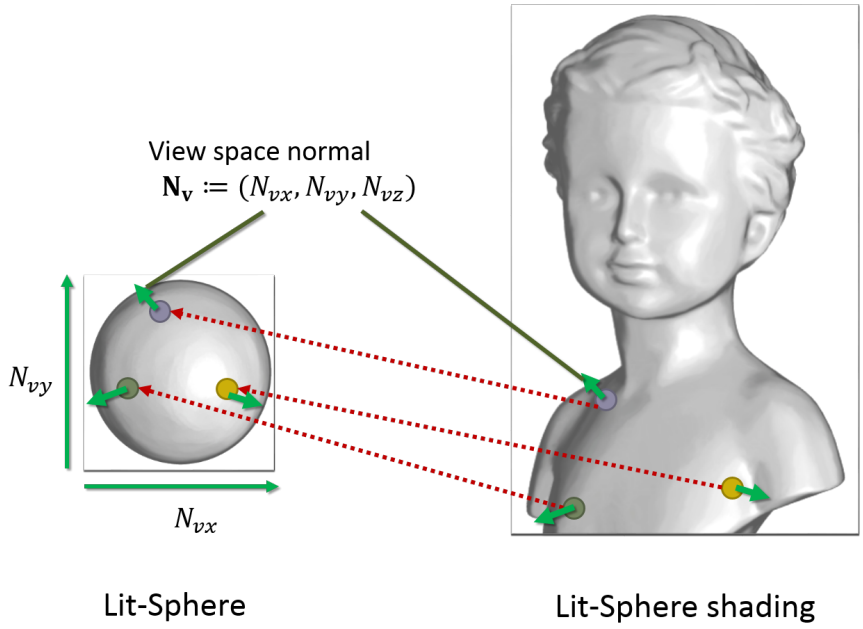


Figure 6.2: *Lit-Sphere shading. The pictorial shading tones are captured using the Lit-Sphere model.*

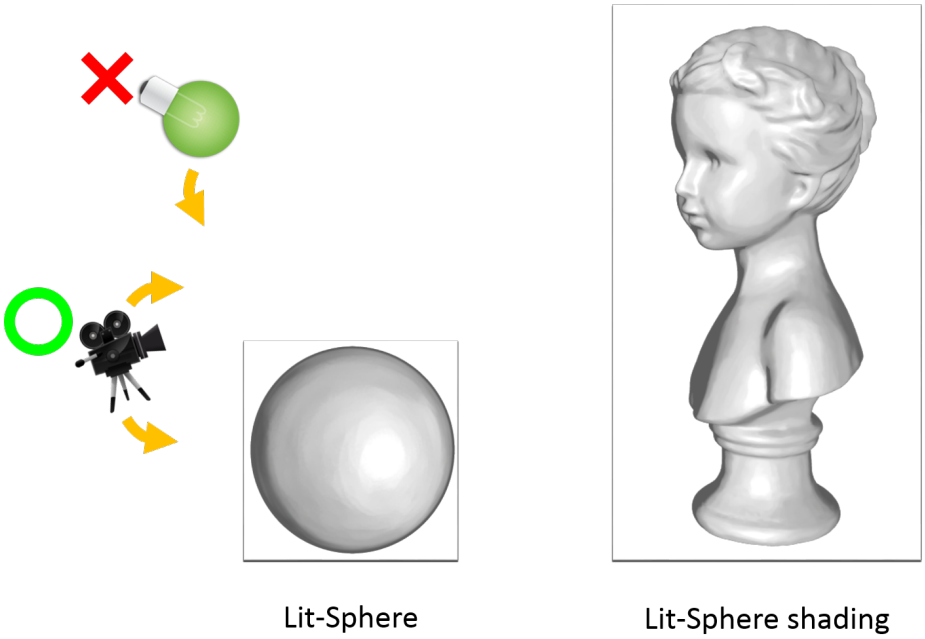


Figure 6.3: *Lit-Sphere issue 1: static lighting appearance. Manipulations of the light do not affect the shading result.*

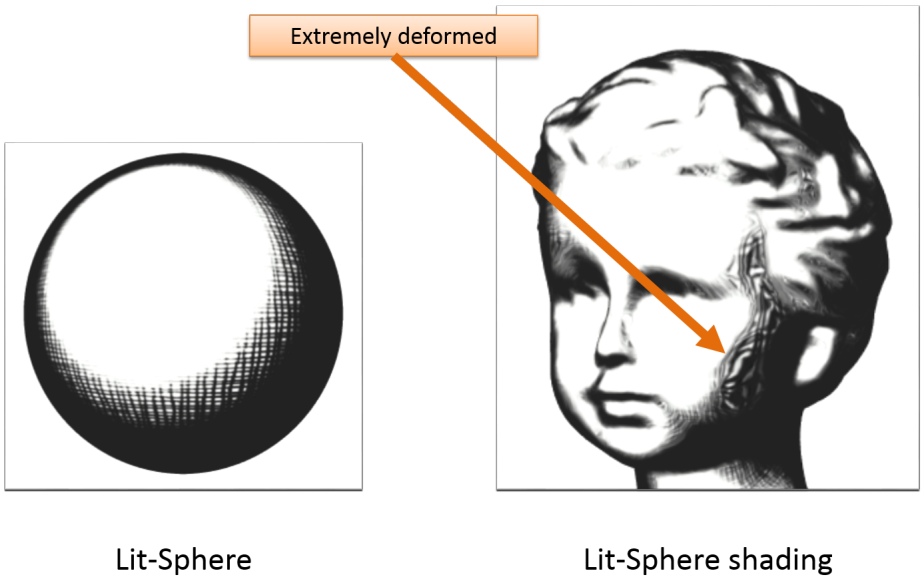


Figure 6.4: *Lit-Sphere issue 2: artifacts of small-scale stylizations. The regular pattern of shading strokes is extremely deformed in Lit-Sphere shading result.*

6.3 Background

Early stylized rendering techniques are typically described by simplified shading tones computed using diffuse and specular terms. Gooch and Gooch [36] used cool-to-warm shading tones for their technical illustration shader. Alternative 1D texture representation has been used to emulate illustrative shading styles for video games [58]. For more complex shading styles, X-Toon [11] extends 1D shading tones to a 2D function, stored in a 2D texture. The additional dimension is used to convey specular, depth or surface orientation. However, this kind of simple mapping of the computed diffuse and specular shading tones permits less control over the shading appearance.

To give the artists more control on the top of stylized rendering results, several techniques have been developed that modify the shape of the shading effect. The cartoon highlights of [4] [96] deal with shape transformation by dragging operations. The stylized highlight shape is adjusted via translation, rotation and scaling operators achieved using vector-field transforms. Ritschel et al. [74] describe a shading deformation technique that is based on a virtual piece of cloth. By modifying sample points of the shading components, reflections and shadows can be dragged on the surface. Todo et al. [90] and Pacanowski et al. [68] give more direct control on the highlight shapes by providing intuitive painting methods. Although these methods allow additional flexibility to achieve the desired shape of the shading effects, the final shading appearance is limited to the specification of the stylized materials.

For further shading stylizations, several good approaches have been developed that allow artists to design custom textures to achieve finer controls over the shading appearance. The Lit-Sphere model [87] allows an artist to design shading tones in pictorial space using a 2D texture of a shaded sphere; however, this approach does not support dynamic lighting. As for hatching styles, Kulla et al. [49] and Yen et al. [109] proposed procedural methods for generating coherent stroke animation using structured brush stroke textures. These methods focus on variation in shading styles, but shape control is not considered.

The Lit-Sphere model [87] has the advantage that complex shading tones can be de-

signed in the pictorial space. For this reason, we chose to extend it to provide additional controls that will result in a scheme where both highlight shape control and shading stylizations are seamlessly combined.

6.4 User Interaction

Our prototype system allows the artist to design a shading style with three shading design processes: Lit-Sphere map design, highlight shape design, and small scale stylization design. In these shading design processes, editing results are interactively updated through user operations. The operations of our system are summarized in Table 6.1.

Design process	**User operation**
Lit-Sphere map design (Figure 6.5)	Paint shading on a reference sphere · Diffuse component · Specular component
Highlight shape design (Figure 6.6)	Adjust transform parameters · Directional scaling · Rotations · Translations
Small scale stylization design (Figure 6.7)	Adjust stylization parameters · Rim lighting · Shading strokes

Table 6.1: *User operations of our system.*

We now proceed to describe these user operations in detail. The artist starts with Lit-Sphere map design on a reference sphere (see Figure 6.5) by painting a diffuse component first and then adding a specular component. In our implementation the designed diffuse and specular components are stored in the 2D textures, which are then transferred to a target 3D object. The artist can animate these shading components by manipulating a single directional or point light source. After the initial design of shading tones, lighting shape can be adjusted using a few simple transform parameters including directional scaling, rotations, and translations (see Figure 6.6). The artist can further control and enhance the small scale features using the rim lighting effects and the shading strokes, if desired (see Figure 6.7). Except painted Lit-Sphere textures, the lightning and parameters are designed using key-frame editing, which enable dynamic control over the shading appearance. Besides, our system also provides real-time feedback to the editing operations, which are essential for interactive shading design.

6.5 Dynamic Lit-Sphere: Defining The Light Space Normals

In this section, we describe how to extend the Lit-Sphere model to deal with dynamic 3D lighting. By introducing the concept of light space normals, we can animate the shading using common diffuse and specular behavior.

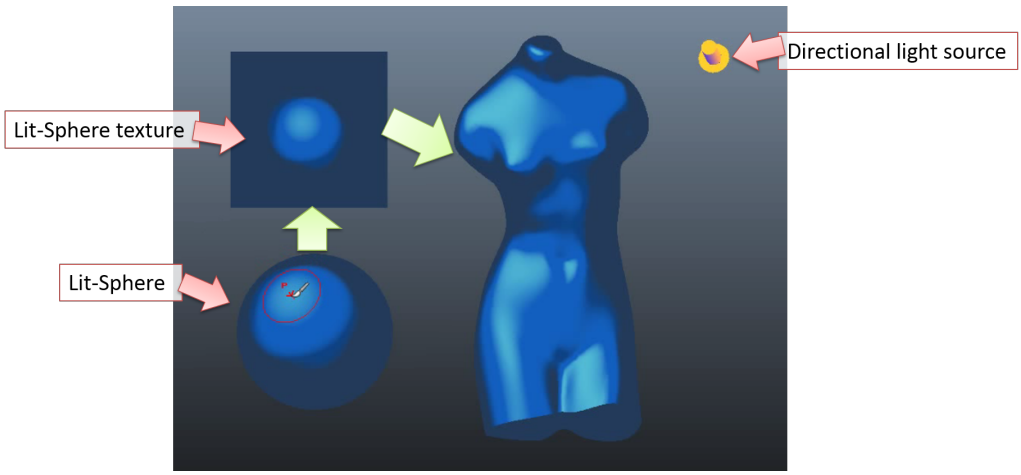


Figure 6.5: *Lit-Sphere design for shading tones. The artist designs Lit-Sphere maps by intuitive painting process. The designed shading tones are stored into 2D texture, and then transferred to the target 3D model. The artist can manipulate the shading tones using a single directional or point light source.*

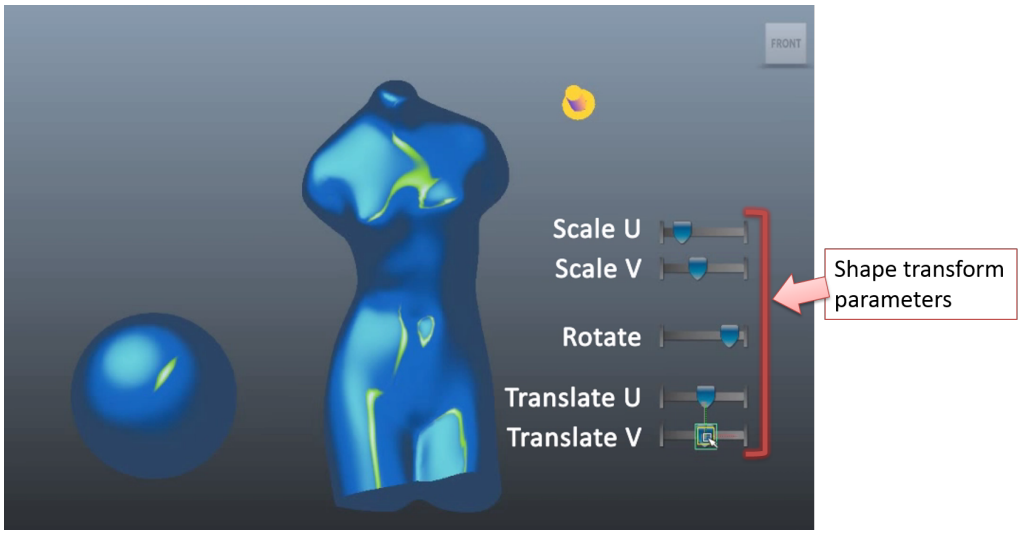


Figure 6.6: *Highlight shape design. These simple transform parameters including directional scaling, rotation, and translations are used to adjust the highlight shape.*

6.5.1 Original Lit-Sphere Model

Sloan et al. [87] described an effective method for generating stylized shading using a reference sphere map. The essence of the technique is to capture the shading style of an object as a function of view space normals (see Figure 6.8). This function is stored in a 2D texture, which is then transferred to a target 3D object. We compute the texture coordinates $(u,v) \in [-1,1] \times [-1,1]$ at a point $\mathbf{p}$ on a surface $\mathbf{S}$ as follows:

$$(u(\mathbf{p}), v(\mathbf{p})) = (N_{vx}(\mathbf{p}), N_{vy}(\mathbf{p})). \tag{6.1}$$

We use the texture coordinates to sample color from the texture, and $N_{vx}(\mathbf{p})$ and $N_{vy}(\mathbf{p})$ are the components of the view space normal $\mathbf{N_v}(\mathbf{p}) := (N_{vx}(\mathbf{p}), N_{vy}(\mathbf{p}), N_{vz}(\mathbf{p}))$, where $N_{vx} := (\mathbf{N} \cdot \mathbf{V_x})$ and $N_{vy} := (\mathbf{N} \cdot \mathbf{V_y})$ are obtained from the surface normal vector $\mathbf{N}$ and view plane vectors $\mathbf{V_x}$, $\mathbf{V_y}$. This approach is effective for designing pictorial shading tones for static scenes. Here, we want to obtain such expressiveness with dynamic lighting.

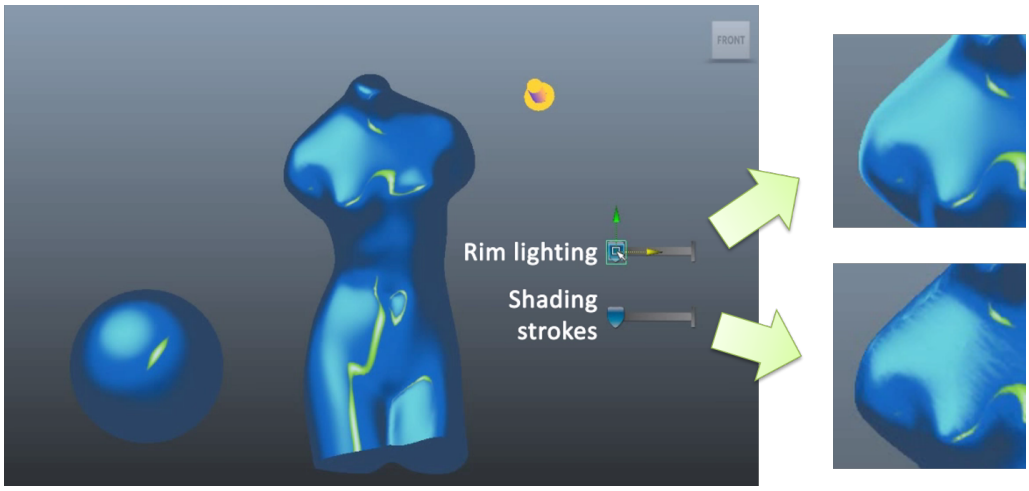


Figure 6.7: *Rim lighting effects and shading stokes. These stylization effects are used to design the small scale shading appearance.*

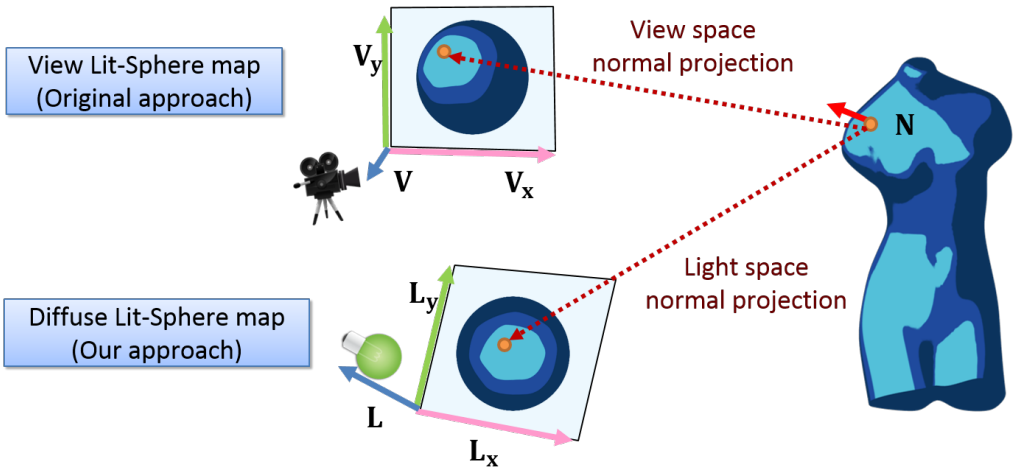


Figure 6.8: *The original view Lit-Sphere shading model compared to the dynamic diffuse Lit-Sphere (our approach). The original view Lit-Sphere uses view space normals to represent the shading of an object. The dynamic diffuse Lit-Sphere uses light space normals, enabling dynamic lighting environments.*

6.5.2 Dynamic Diffuse Behavior

To animate the Lit-Sphere model with dynamic lighting control, we introduce a new space normal representation: light space normals. For a given light direction $\mathbf{L}(\mathbf{p})$, we define the light space using three orthogonal vectors $\mathbf{L}(\mathbf{p})$, $\mathbf{L_x}(\mathbf{p})$ and $\mathbf{L_y}(\mathbf{p})$ (see Figure 6.8). The precise definition of two other vectors $\mathbf{L_x}$ and $\mathbf{L_y}$ will be provided in Section 6.5.4. We refer to the plane spanned by $\mathbf{L_x}$ and $\mathbf{L_y}$ as the light view throughout the rest of this chapter. Because the space is light dependent it can handle dynamic lighting. While our shading model is capable of dealing with specular behavior (see Section 6.5.3), we first describe the simpler diffuse behavior. For a given light space, we define a light space normal $\mathbf{N_l} := (N_{lx}, N_{ly}, N_{lz})$ as follows:

$$\begin{aligned} N_{lx}(\mathbf{p}) &:= (\mathbf{N}(\mathbf{p}) \cdot \mathbf{L_x}(\mathbf{p})), \\ N_{ly}(\mathbf{p}) &:= (\mathbf{N}(\mathbf{p}) \cdot \mathbf{L_y}(\mathbf{p})), \\ N_{lz}(\mathbf{p}) &:= (\mathbf{N}(\mathbf{p}) \cdot \mathbf{L}(\mathbf{p})), \end{aligned} \tag{6.2}$$

where the surface normal $\mathbf{N}$ is transformed to the light space normal $\mathbf{N_l}$. With this light space normal definition, we obtain the texture coordinates $(r,\theta) \in [0,1] \times [0,2\pi]$ for a diffuse Lit-Sphere map using the following relations:

$$\begin{aligned} r(\mathbf{p}) &= arccos(N_{lz}(\mathbf{p}))/\pi, \\ \theta(\mathbf{p}) &= arctan(N_{ly}(\mathbf{p})/N_{lx}(\mathbf{p})), \end{aligned} \tag{6.3}$$

where θ is the angle in the light view and r denotes the radial coordinate derived from the brightness term $N_{lz} = \mathbf{N} \cdot \mathbf{L}$. The final shading color is sampled from the Cartesian coordinates $(u,v) = (rcos\theta, rsin\theta)$, which is readily transformed from the polar coordinates (r,θ).

These texture coordinates are directly related to lighting information, i.e., brightness (r) and light view angle (θ). We will show how these can be used efficiently to add various shading stylizations in Section 6.6.

6.5.3 Dynamic Specular Behavior

Unlike diffuse, specular is dependent on the view direction. Using our light space normal definition, we implemented two common specular models: the Phong model and the Blinn-Phong model. Figure 6.9 illustrates our specular map in the case of the Blinn-Phong model. In practice, the specular layer is composited over the diffuse layer, which provides more plausible shading appearance.

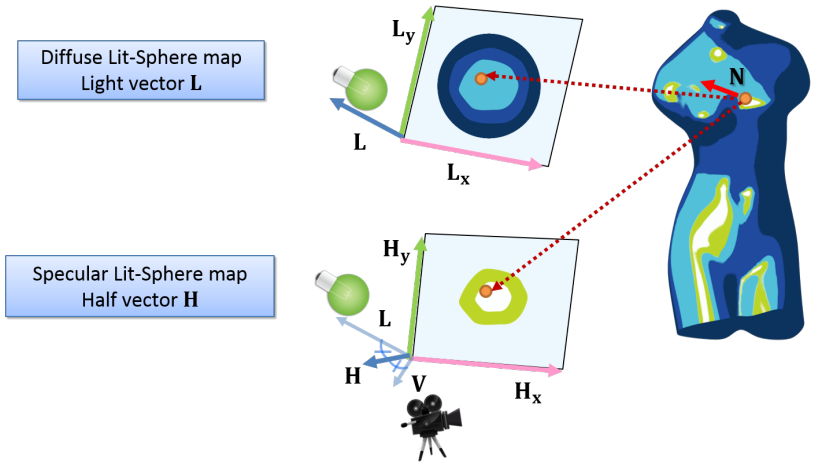


Figure 6.9: *The specular Lit-Sphere map based on the Blinn-Phong model. The specular layer is composited over the diffuse layer. The half vector* **H** *is used to integrate specular behavior into our shading process.*

The Phong model produces a stretched highlight shape described by the specular term $\mathbf{L} \cdot \mathbf{V}'$ where $\mathbf{V}' := 2(\mathbf{V} \cdot \mathbf{N})\mathbf{N} - \mathbf{V}$ denotes the reflected view vector. We integrate the Phong model with the light space normal approach using the reflected view vector $\mathbf{V}'$ rather than the surface normal vector $\mathbf{N}$ in Equation 6.2. The top images of Figure 6.10 show the reflectance properties in Phong: the highlight shape is more elongated near the silhouettes.

The Blinn-Phong model preserves the original highlight shape. The behavior is described by the specular term $\mathbf{H} \cdot \mathbf{N}$ where $\mathbf{H} := (\mathbf{L}+\mathbf{V})/\|\mathbf{L}+\mathbf{V}\|$ denotes the half vector. We integrate the Blinn-Phong model using the half vector $\mathbf{H}$ rather than the light vector $\mathbf{L}$ in Equation 6.2. While this modification effectively provides the Blinn-Phong behavior,

it may result in an animation artifact when the light comes from the back of the object. We address this issue by interpolating the specular and diffuse behavior according to the angle of the light and the view vectors.

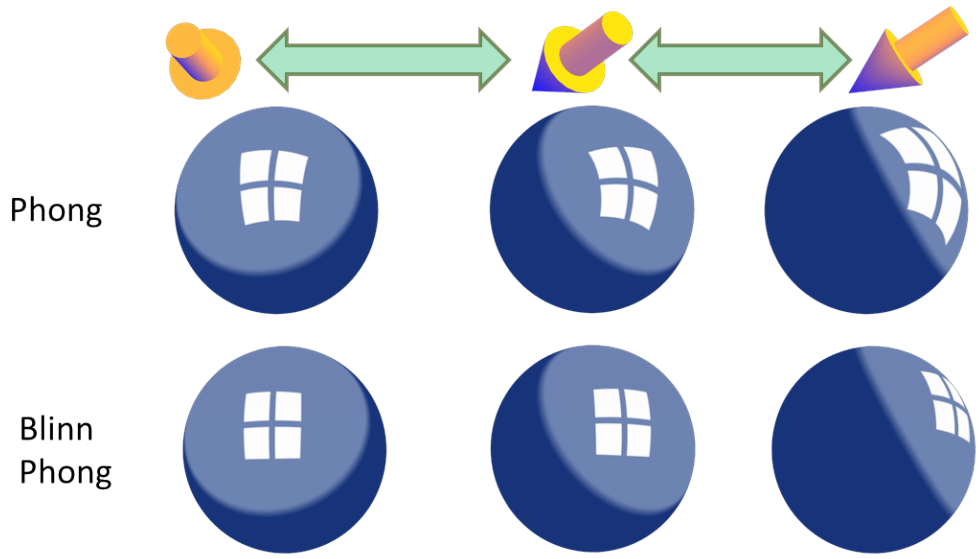


Figure 6.10: *Comparison between Phong and Blinn-Phong models. The Phong model produces a stretched highlight shape. The Blinn-Phong model preserves the highlight shape.*

Figure 6.10 shows a comparison between the visual appearance generated using the two models. Our system allows an artist to use both specular models as the situation demands: the Phong model is typically used to emphasize the reflection properties of objects whereas the Blinn-Phong model is more effective for preserving the shape of the highlight.

While the Blinn-Phong offers a good property for preserving lighting shape, it leads to animation artifacts without multiplying geometric term $(\mathbf{N}\cdot\mathbf{L})$. An undesirable rotation of highlight is observed when the light direction comes from the back of the object (see Figure 6.11). Incorporating the geometric term is not straight-forward, since our shading model is not only determined by the brightness term $(\mathbf{H}\cdot\mathbf{N})$, but also the *light view* $(\mathbf{L_x},\mathbf{L_y})$. As an alternative approach, we reduce the artifacts by modifying half vector $\mathbf{H}$ to $\mathbf{H}'$:

$$\mathbf{H}' = R_t(\mathbf{H},\mathbf{L},t), \tag{6.4}$$

where R_t computes spherical interpolation of vector $\mathbf{H}$ and $\mathbf{L}$, and its interpolation factor is t. We define t heuristically as $2(\pi - arccos(\mathbf{L}\cdot\mathbf{V}))/\pi$ clamped to $[0,1]$. Intuitively, the interpolation starts when the light starts coming from the back-side of the view ($\mathbf{L}\cdot\mathbf{V} = 0$), ends when the light and the view are opposite($\mathbf{L}\cdot\mathbf{V} = -1$). As shown in Figure 6.11, this simple modification works well for reducing the animation artifacts.

6.5.4 Light Space Definition

In our light space approach, lighting orientation is specified by the light view $(\mathbf{L_x},\mathbf{L_y})$. Although the user could manually specify the orientation of this light view, we provide a method to automatically define it from the given view and light settings.

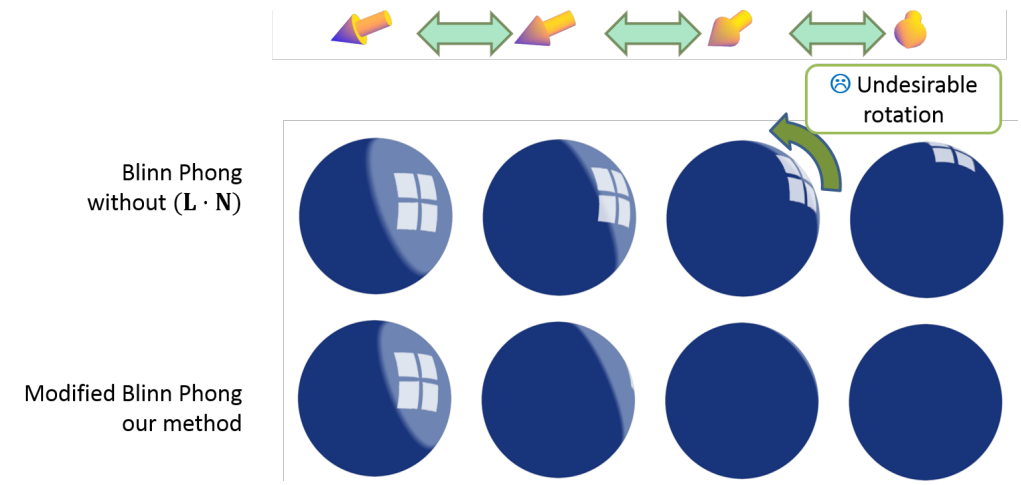


Figure 6.11: *Comparison between original Blinn-Phong and modified Blinn-Phong (our method). The original Blinn-Phong model results in the undesirable rotation of highlight. Our modified Blinn-Phong model reduces the animation artifact.*

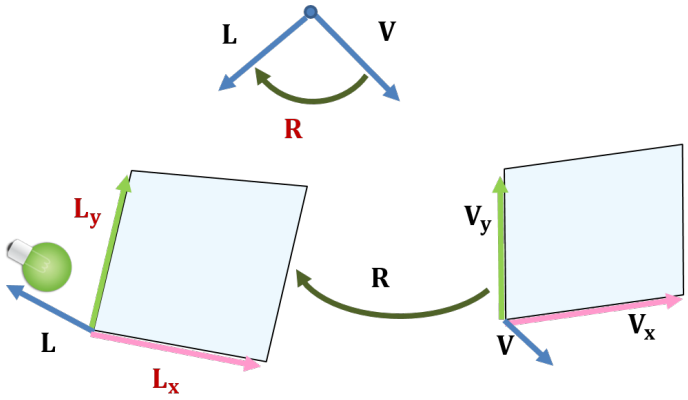


Figure 6.12: *Rotation of the camera view to light view. The spherical rotation* **R** *is obtained from* **V** *and* **L**. *The camera view* $(\mathbf{V_x}, \mathbf{V_y})$ *is transformed to the light view* $(\mathbf{L_x}, \mathbf{L_y})$ *with* **R**.

An overview of our methodology is illustrated in Figure 6.12, where the camera view $(\mathbf{V_x}, \mathbf{V_y})$ is transformed to the light view $(\mathbf{L_x}, \mathbf{L_y})$ as a function of the camera view and the light direction. The transform is given by the minimum angle spherical rotation **R** between **V** and **L**. Since the rotation angle is minimum, the light view $(\mathbf{L_x}, \mathbf{L_y})$ will be similar to the camera view $(\mathbf{V_x}, \mathbf{V_y})$.

One approach is to use the static tangent space instead of the light space we defined. Figure 6.13, 6.14 shows a comparison between our method and the static tangent space given by the tangent vector **t** and the binormal vector **b**. The lighting orientation based on the tangent space is well-suited for depicting the surface flows. However, a minor drawback is that the lighting orientation is strongly constrained by the given tangent field. The result is that the highlight becomes distorted at the singular point (see Figure 6.13). Another issue is that if the anisotropic highlight orientation varies along the static tangents, then this orientation will not be coherent throughout the model (see Figure 6.14). Our light space approach preserves the highlight shape regardless of the tangent space definition and results in a highlight orientation that is coherent along the view orientation.

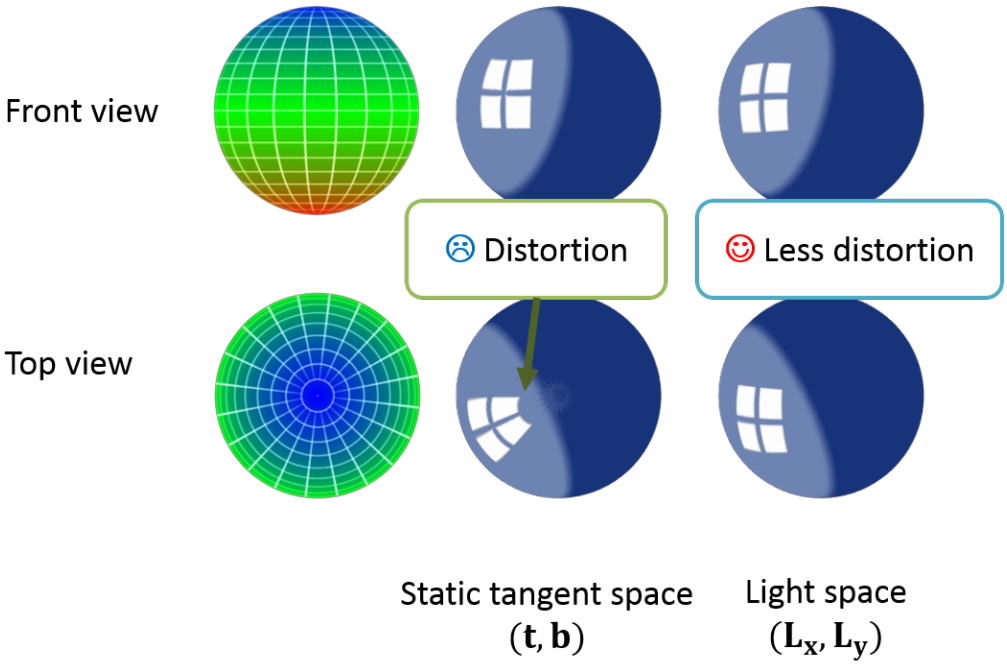


Figure 6.13: *Lighting orientation comparisons for symbolic highlight. Static tangent space causes distortion near to the singular point at the pole. The light space preserves the highlight shape regardless of the tangent space definition.*

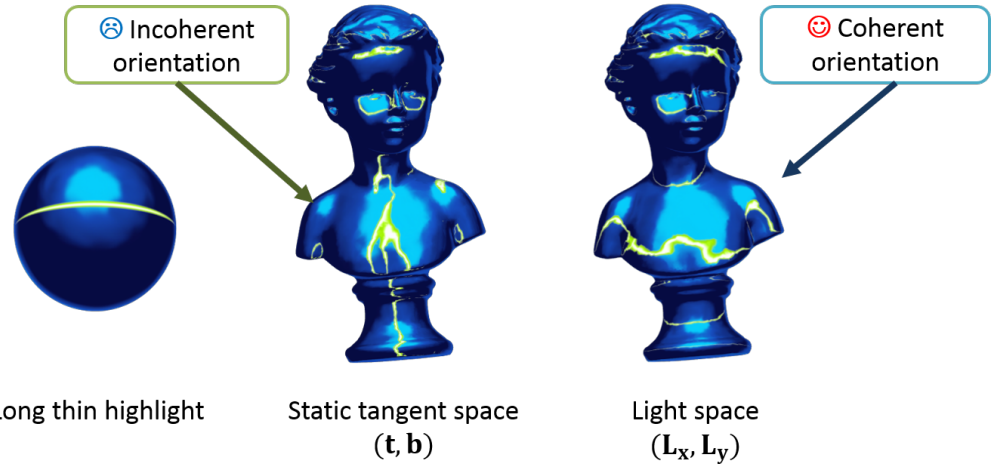


Figure 6.14: *Lighting orientation comparisons for a long thin highlight. The highlight orientation varies along the static tangent direction. Employing the light space results in the coherent orientation along the view orientation.*

6.6 Shading Stylizations: Transforming The Light Space Normals

In this section, we describe how to apply secondary stylizations to the designed shading tones. Thanks to the light space representation of our Lit-Sphere extension, transformation of the light space results in shading transformations. In the following sections, we describe how we can use such transformations to implement secondary stylizations.

6.6.1 Highlight Shape Transforms

To give users finer control over the highlight shape, we apply lighting transforms similar to those described in Section 5.5 and [4]. While these methods used a vector-field to deform the highlight shape, we can use simple texture transforms to achieve the same effects.

The texture transforms are designed as a composite transform function $A : [-1,1] \times [-1,1] \rightarrow [-1,1] \times [-1,1]$ for the texture coordinates (u,v) such that

$$A(u,v) := A_d(A_r(A_t(u,v))), \tag{6.5}$$

where A_t is the translation operator, A_r is the rotation operator, and A_d is the directional scaling operator.

The translation operator A_t is defined by two parameters α and β as follows:

$$A_t(u,v) := (u-\alpha, v-\beta). \tag{6.6}$$

The rotation operator A_r is defined by one parameter ϕ as follows:

$$A_r(u,v) := (u,v) \begin{bmatrix} \cos(-\phi) & \sin(-\phi) \\ -\sin(-\phi) & \cos(-\phi) \end{bmatrix}. \tag{6.7}$$

The directional scaling operator A_d is defined by two parameters γ and δ as follows:

$$A_d(u,v) := (u/\gamma, v/\delta). \tag{6.8}$$

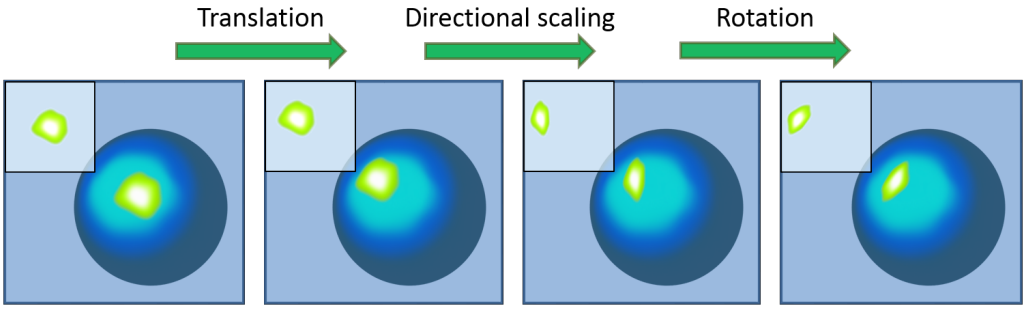


Figure 6.15: *Highlight shape transforms. Starting from the initial state in the leftmost image, the highlight shape is deformed by the transform operations.*

Figure 6.15 demonstrates how these operations deform the highlight shape. The parameters are simple and straightforward so that the artist can adjust the highlight shape intuitively.

6.6.2 Lighting Offset for Feature Enhancements

An artist can control and enhance the small-scale features using our lighting offset technique. This process is illustrated in Figure 6.16. In a similar manner to the traditional bump mapping process, the attribute value $h \in [-1,1]$ is used to modify the brightness value (specific examples of h will be described later). When h is a maximum ($h = 1$) shading will be bright whereas at the minimum ($h = -1$) shading will be dark.

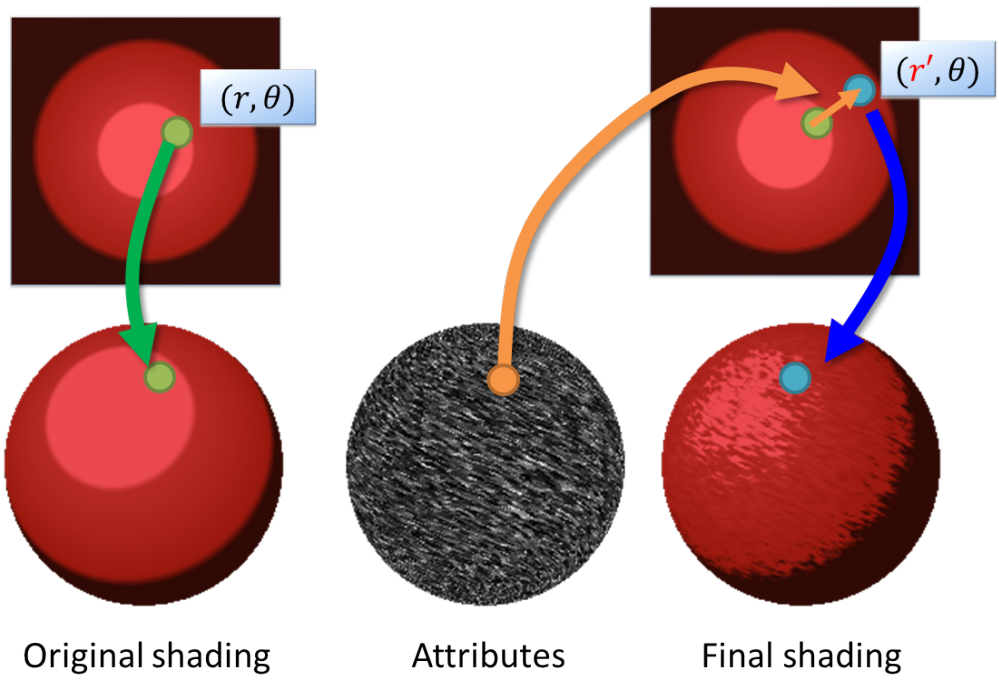


Figure 6.16: *Lighting offset for feature enhancements. The original shading color is sampled from the polar coordinates* (r,θ)*. With the user-defined attribute value h, the coordinate r is deformed to* r'*. Final shading is enhanced by the attribute with the deformed polar coordinates* (r',θ)*.*

In contrast to bump mapping, our approach provides more direct control over the brightness value without transforming the surface normal vectors. Thanks to the light space representation of the texture coordinates (r,θ), we can modify the brightness value by deforming the radial coordinate r to r'. For $h = 0$ (no offset), we use the coordinate $r' = r$. For the brightest values ($h = 1$), we use the coordinate $r' = 0$ corresponding to the brightest point in the texture map. For the darkest values ($h = -1$), we use the coordinate $r' = 1$ corresponding to the darkest point in the texture map. For intermediate values of h, r' is interpolated according to

$$r' = \begin{cases} C(h,-1,0)(r-1)+1 & (-1 \leq h \leq 0) \\ (1-C(h,0,1))r & (0 \leq h \leq 1) \end{cases}, \tag{6.9}$$

where $C(x,a,b) \in [0,1]$ computes the interpolation term of $x \in \mathbb{R}$ between $a \in \mathbb{R}$ and $b \in \mathbb{R}$. The function C was a clamped cubic Hermite interpolation via the common *smoothstep* function, which is available as a built-in feature of most GPU shading languages. Then the term r' in Equation 6.9 is used to sample the final shading color.

Using various choices of h in Equation 6.9, we can easily specify different stylizations. In the following, we will present typical usages of h to achieve rim lighting and shading strokes.

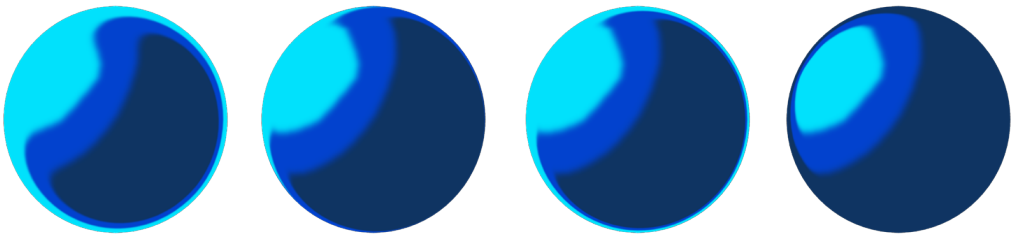
Rim lighting:

The rim lighting effect is the result of a light behind the objects. It results in a bright glow effect on the shading around silhouettes. We can implement such effects using the

facing ratio $(\mathbf{N} \cdot \mathbf{V})$ and defining the offset $h_r : S^2 \times S^2 \rightarrow [-1,1]$:

$$h_r(\mathbf{N},\mathbf{V}) := \mu C(|arccos(\mathbf{N} \cdot \mathbf{V})|, \eta, 0), \tag{6.10}$$

where $\mu \in [-1,1]$ controls the brightness of the glow, $\eta \in [0,\pi]$ is the size of the glow and C is the same cubic Hermite interpolation function as in Equation 6.9. Figure 6.17 shows how different values of μ and η allow an artist to design rim lighting appearances. With a larger η, the rim lighting effect becomes sharper. Note that μ can be negative, which corresponds to darkening.



$\mu = 0.5$ $\eta = 0.5$ | $\mu = 0.7$ $\eta = 0.5$ | $\mu = 0.7$ $\eta = 0.8$ | $\mu = -0.4$ $\eta = 0.5$

Figure 6.17: *Rim lighting effects. By varying μ and η, different rim lighting effects are obtained. The designed effects are seamlessly integrated with the original shading tones.*

Shading strokes:

As shown in Figure 6.4, specifying the shading strokes directly in the Lit-Sphere map may result in shading artifacts. Therefore, we separate the shading strokes from the shading tones. We use a structured brush stroke texture map $h_s(s,t) \in [-1,1]$ for the shading stroke attribute, where (s,t) are the object space texture coordinates attached to the model (see the middle image of Figure 6.16). By changing the structured texture maps, we can combine various stroke styles with the original shading tones (see Figure 6.18). In contrast to the original Lit-Sphere approach, our shading strokes maintain coherent motion using the object space textures as demonstrated in Section 6.8.

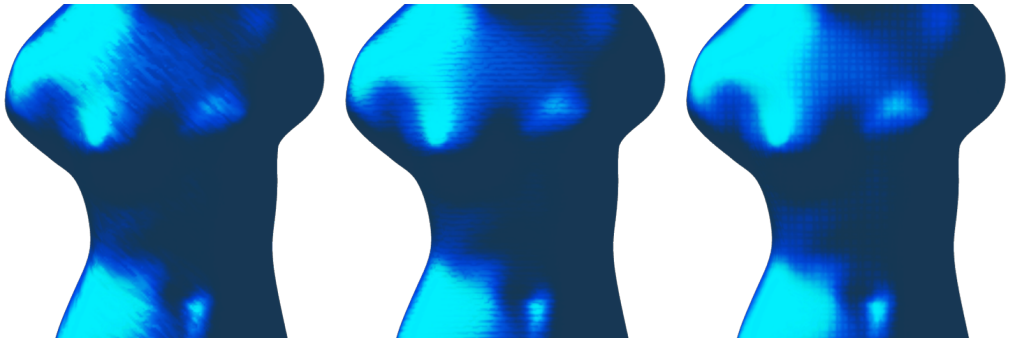


Figure 6.18: *Shading stroke variation. By changing the structured brush stroke texture maps, various shading stroke styles can be designed.*

6.7 Implementation

To integrate the proposed methods into existing 3D shading design process, we implemented our Lit-Sphere extension as a Maya plug-in.

GPU Shading Process

To perform the rendering process at interactive rate, our rendering algorithms were implemented using a CgFX shader that allows to design a custom GPU shading process. The diffuse and specular Lit-Sphere maps are stored into 2D textures, which are referenced in the shading process. In our GPU program, most of the shading process are computed in the pixel shader, including the secondary stylization (highlight shape transforms and lighting offset). We use additional 2D brush stroke textures for shading strokes, which is assigned to the object space texture coordinates.

Lit-Sphere Painting on A Reference Sphere

Separately from the GPU shading process, we use a CPU-based Maya painting mechanism for our painting process for Lit-Sphere maps. Our system dynamically updates the per-vertex texture coordinates of a reference sphere when the light and the view change. The computational cost depends on the number of vertices of the reference sphere. Since we do not need a highly tesselated sphere in practice, this computation is not a serious bottleneck in our system. In making examples, we used the reference sphere with 1000 vertices for our Lit-Sphere painting.

6.8 Results

We have applied our prototype system to designing various artistic shading styles. In making the following examples, our system enables interactive shading design with NVIDIA Quadro 600 (more than 30fps for our examples). A variety of shading styles are achievable by combining all of the system features (see Figure 6.19). These shading styles can be dynamically controlled by a light source with coherent shading behavior over time. In the following, we describe in more detail how we can achieve the typical shading styles.

Typical artwork shading styles can be obtained by combining a few stepped shading tones and simple stylization effects. In Figure 6.20, we demonstrate the use of our shading model to achieve a minimal shading style similar to the illustration in [40]. We used simple black and white colors for the shading tones, and then applied shading strokes to enhance the surface features.

A more complex illustrative shading style is shown in Figure 6.21, which is obtained by layering diffuse and specular components. To create this style, we applied highlight shape transforms to adjust the size and location of the highlight areas. As shown in the figure, our shading model results in coherent transitions while maintaining the secondary stylizations.

A stylized metallic appearance can be designed using complex reflection patterns. This is illustrated in Figure 6.22, where we designed multiple long thin highlights to generate a gold appearance. Our shading model can animate anisotropic diffuse and specular shading, whereas this type of shading property would be hardly designed with existing stylized rendering techniques.

Figure 6.23 shows the use of our technique with an animated object with a deforming cape. Strong deformations of an object can often result in a lack of coherence of the stylization. The animation in the figure shows that our techniques effectively create coherent motion using the object space structured texture.

Table 6.2 shows the performance of our current implementation. The time for shading process depends on the complexity of the geometry of the target model (the number of vertices and triangles). Even for the highly tesselated model used in the example of minimal shading style, the rendering cost seems not to be serious bottleneck in our system. The performance data in Table 6.2 makes it clear that our GPU-based implementation is sufficiently fast for interactive editing.

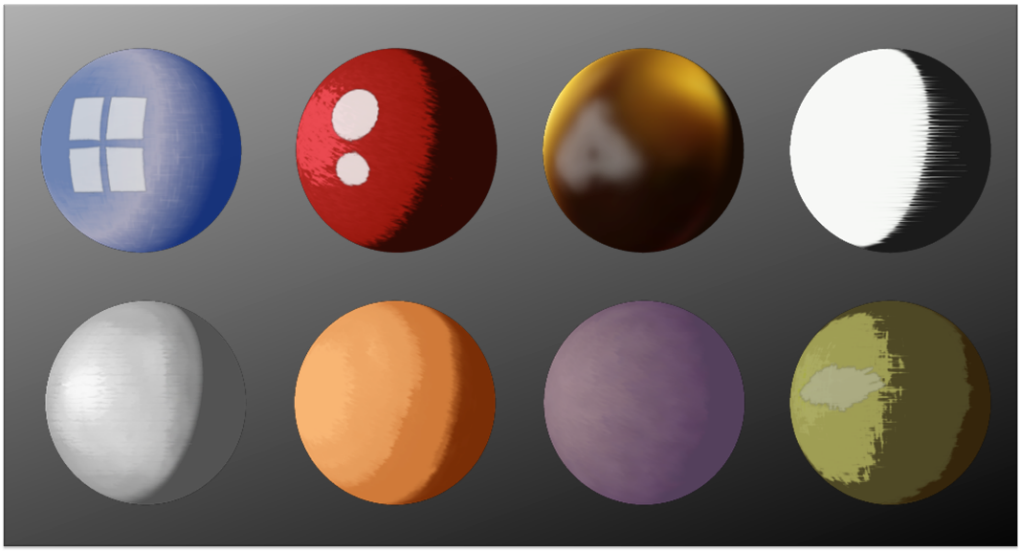


Figure 6.19: *Material variation. Various shading styles are obtained using our system.*

Title	#Verts	#Tris	Shading process[fps]
Minimal shading	26850	53696	32.4
Illustrative shading	19985	39856	78.8
Stylized metallic	8001	15920	99.8
Highly deformed cape	3400	6422	120.1

Table 6.2: *Performance of our shading process. Column describe (from left to right): title, number of vertices of the target model, number of triangles of the target model, time for the shading process.*

6.9 Summary

In this chapter, we have demonstrated an extension for the Lit-Sphere model that enalbes dynamic controls of pictorial shading tones with correlated stylization effects. Thanks to our light space representation, the shading appearance can be controlled with highlight shape transforms and lighting offsets while preserving the original shading tones. The current prototype system allows an artist to design many commonly used artistic shading styles; however, many additional capabilities are needed to meet the growing requirements of artists.

For example, the current implementation of our shading model only allows for a single light source (see Figure 6.24). However, our method can produce multiple lighting appearances designed into a single specular Lit-Sphere map (see Figure 6.22). Work is underway to implement a layered method that blends the Lit-Sphere shading effects with

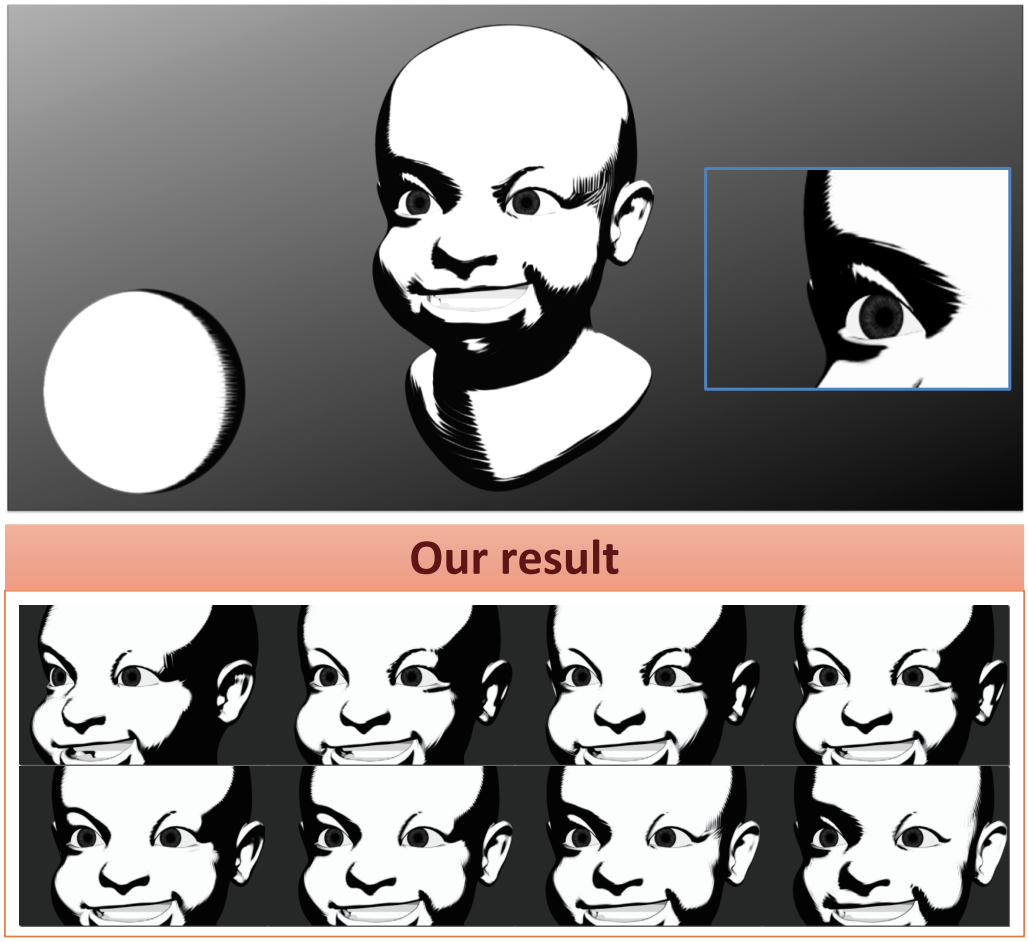


Figure 6.20: *Minimal shading style.*

the brightness term of each light source to give artists more direct control with multiple light sources.

Another limitation of the current implementation is that our shading strokes only provide indirect control whereas an artist may want to directly design brush strokes over the shading tones (see Figure 6.25). Therefore, integrating more direct inverse stylization methods [49] [109] into our system is an area for future direction.

A promising avenue would be dynamic control of brush stroke styles, taking into account the temporal coherence. In our approach, we can obtain coherent shading strokes due to the fixed stroke placement using the object space texture coordinates. Therefore, the stroke placement itself is static during animation. Integrating the recent temporally coherent Image Analogy technique [13] might be helpful for establishing an artist-friendly brush stroke control.

Since these demands depend on the range of shading styles and control that artists may want, we plan to conduct extensive user feedback investigations to target areas where additional functionality is desired, as well as ways to improve both effectiveness and usability of our work.

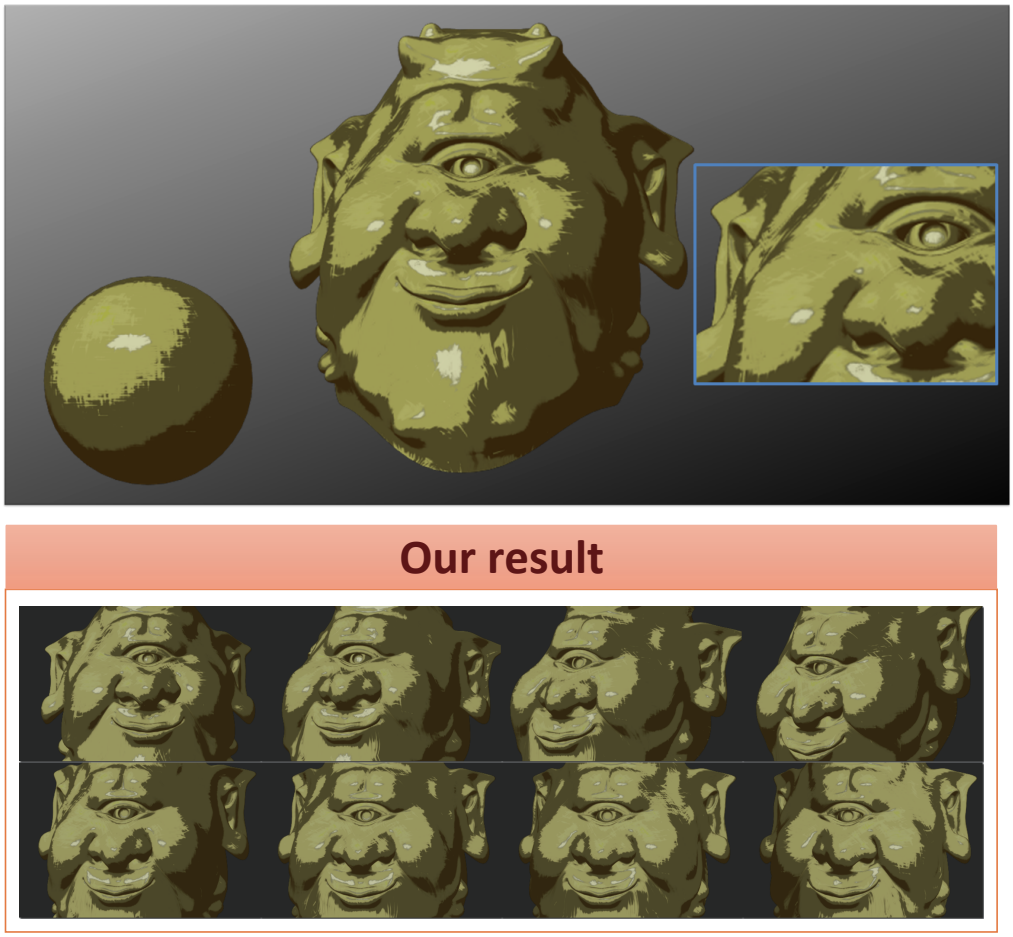


Figure 6.21: *Illustrative shading style.*

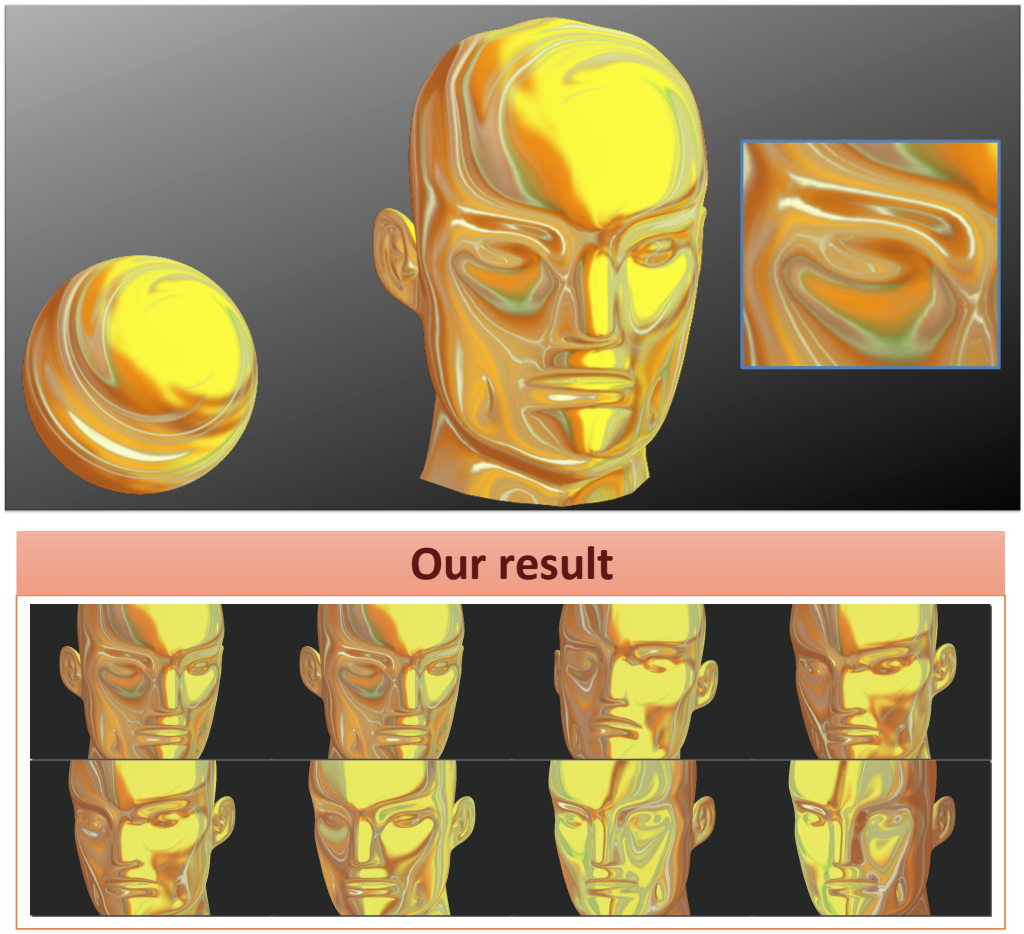


Figure 6.22: *Stylized metallic appearance produced with our system.*

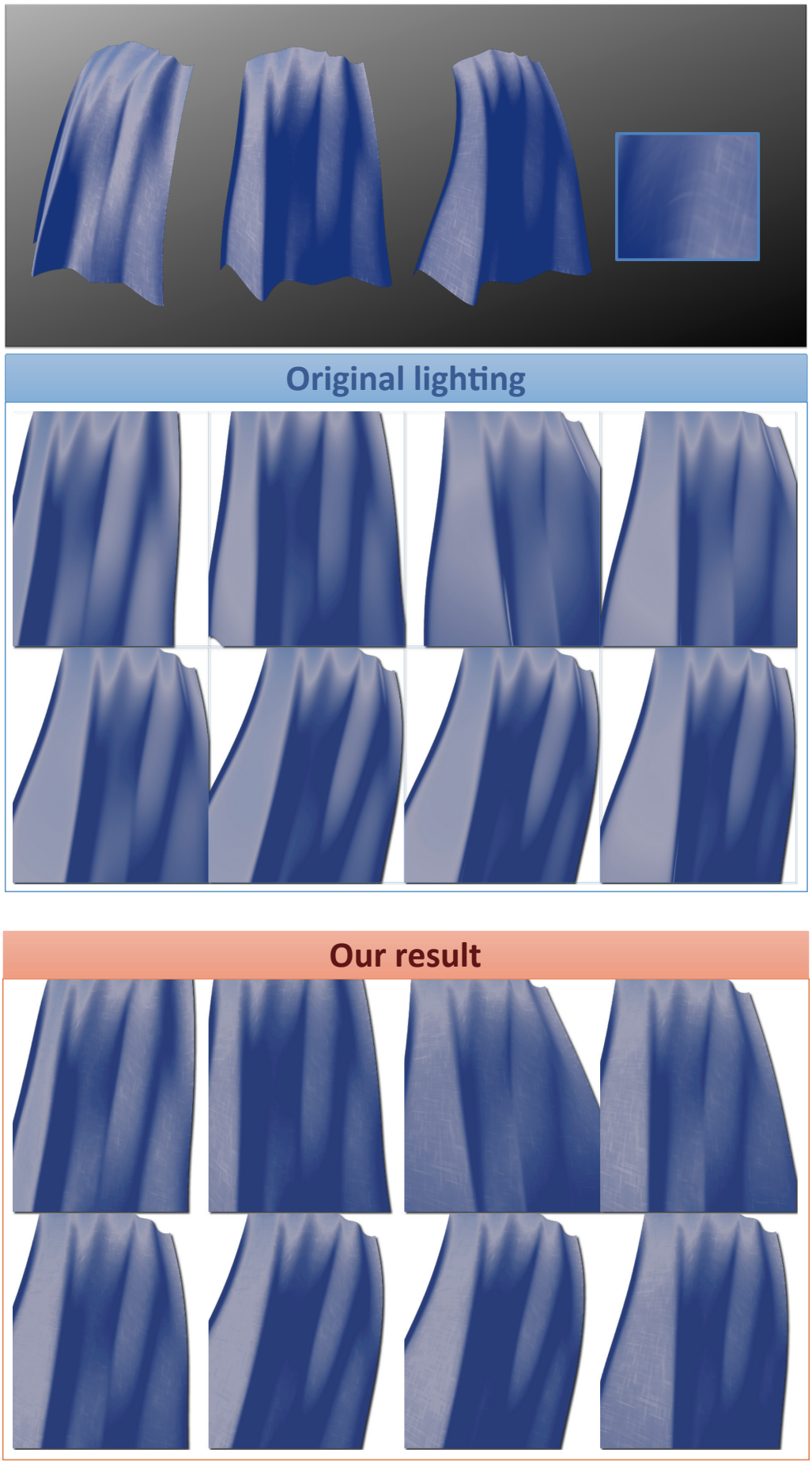


Figure 6.23: *The shading tones and stylizations are coherently animated on the highly deformed cape.*

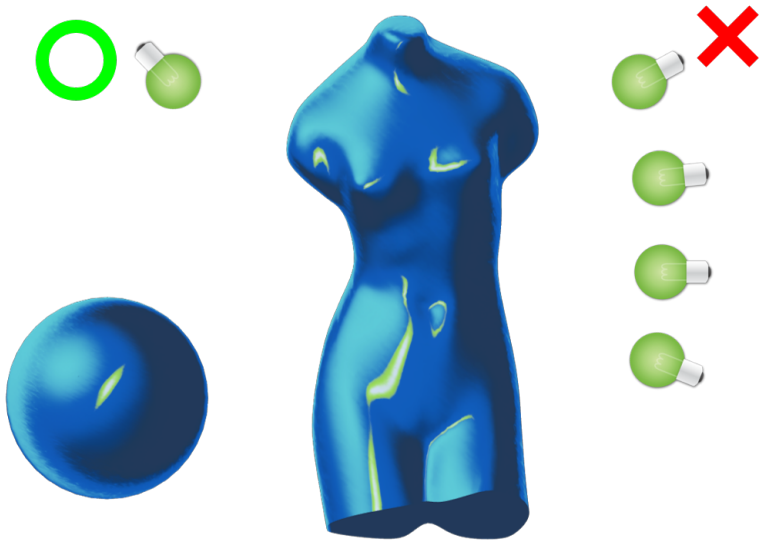


Figure 6.24: *Limitation 1: our shading model is limited to single light source.*

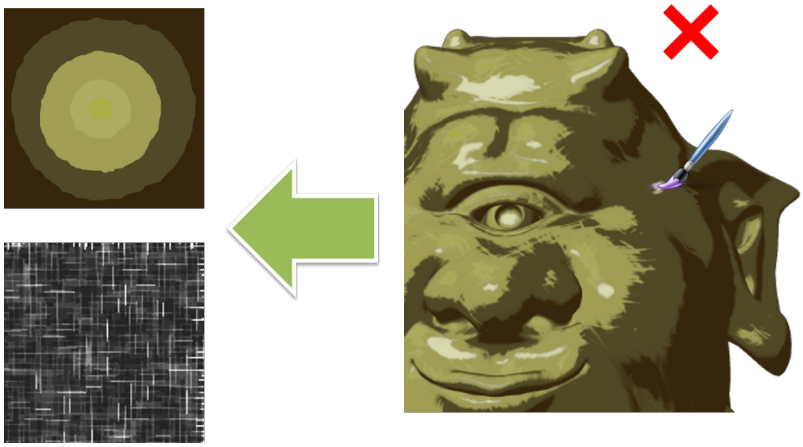


Figure 6.25: *Limitation 2: our shading model does not permit direct shading design on a target model.*

Chapter 7

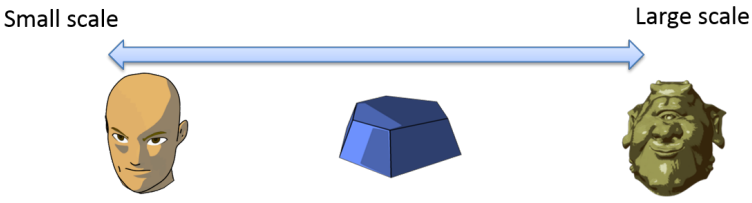
Discussions

In this chapter, we examine the capabilities of the three methods proposed in this thesis, clarifying their strengths and weaknesses. Figure 7.1 summarizes our methods from the perspective of the proposed framework. Depending on the level of the design process, we introduced appropriate directable shading models that are seamlessly integrated into existing 3D lighting controls.

Principle 1: For the directable shading models, we use three kinds of mechanisms: lighting offset, lighting transform, and 2D color mapping. The lighting offset offers the smallest scale control by directly modifying the brightness term. The lighting transform provides larger scale control over the shape of the lights. The 2D color mapping provides the largest scale control of the overall shading than the previous two mechanisms.

Principle 2: Our systems make use of these directable mechanisms with existing 3D lighting controls. Multiple light sources can be used with both the lighting offset and the lighting transform since the both methods are applied to the variables used for lighting computation. On the other hand, we reformulated the entire lighting computation in the case of 2D color mapping. Our reformulation provides expressive shading appearance at the expense of limiting the control to a single light source.

In the following sections, we make detailed comparisons of three directable mechanisms.



Principle 1: Directable shading model	• Lighting offset	• Lighting transform • Lighting offset	• 2D color mapping • Lighting transform • Lighting offset
Principle 2: Seamless integration with 3D lighting	○ Multiple light sources	○ Multiple light sources	△ Single light source

Figure 7.1: *Summary of our methods for an artist-friendly shading design system.*

7.1 Comparison of 1D Color Mapping and 2D Color Mapping

We have demonstrated how our methods achieve expressive shading appearance. We use two types of color mapping for our shading models. The first is 1D color mapping used in Chapters 4 and 5. The second is 2D color mapping used in Chapter 6.

Figure 7.2 shows a comparison of these methods. With 1D color mapping, we simply use the computed brightness as the input of the color mapping function. Since the brightness distribution is determined by the angle between the surface normal vector and the light vector (or the half vector for specular effects), the final shading colors are distributed in a circular shape (top row of Figure 7.2).

With 2D color mapping, the 2D Lit-Sphere maps are projected onto the target 3D model based on light space normals. Since the projection coordinates are defined by light space normals, this shading model allows the artist to design normal-dependent shading styles. The bottom row of Figure 7.2 illustrates how 2D color mapping is effective for emulating multiple lighting appearance.

In summary, 1D color mapping is limited to simple shading appearance, according to the brightness distribution. On the other hand, 2D color mapping enables a normal-dependent color distribution, which produces more expressive shading styles.

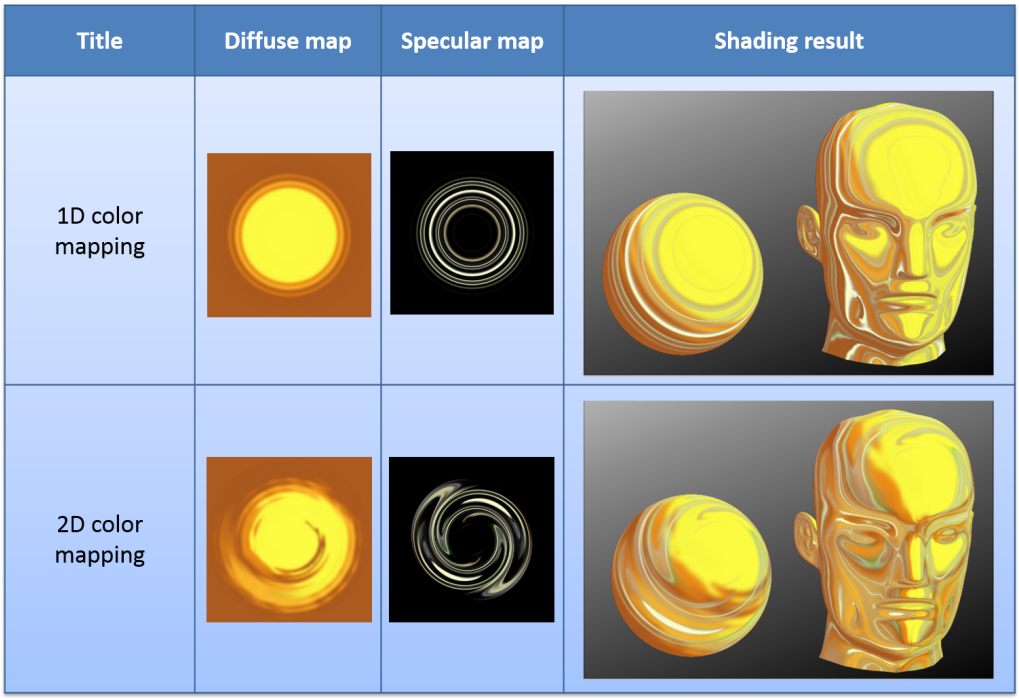


Figure 7.2: *Comparison of 1D and 2D color mapping. (Top row) 1D color mapping produces circular tone distributions. (Bottom row) 2D color mapping allows the artist to design more complex normal-dependent tone distributions.*

7.2 Comparison of Lighting Transform and Lighting Offset

To change the shape of lighting, we use two types of directable lighting mechanisms: lighting transform and lighting offset. Figure 7.3 shows operation examples edited by these lighting shape control mechanisms. In the second system (Chapter 5), lighting

transform functions are used to change the round shape of lighting. Edge lighting offsets provide a more local control over the shape of lighting based on specific model features. In the first system (Chapter 4), local lighting offsets are calculated based on user painting operations, which enables fine-tuning of the lighting shape. In the following, we compare these lighting shape control mechanisms with simple operation examples.

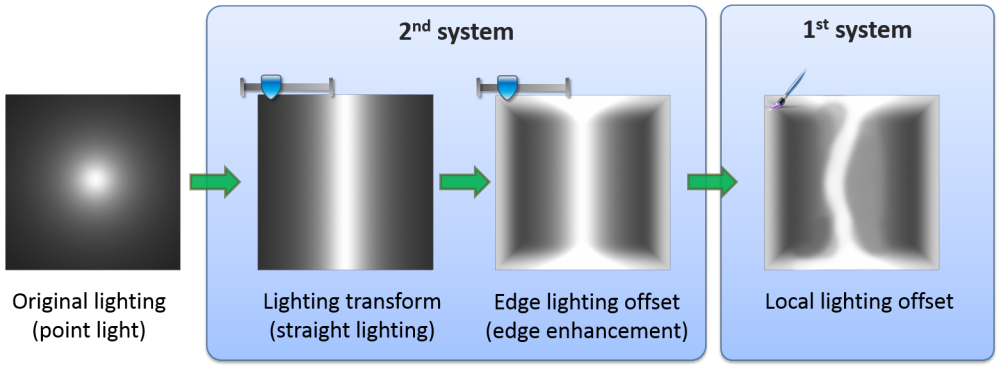


Figure 7.3: *Operation example of lighting shape controls. Starting from the initial state obtained with a point light source in the leftmost image, the lighting shape is deformed by our lighting shape control mechanisms.*

Figure 7.4 compares lighting transform and lighting offset for the case where a straight lighting effect (lighting transform) is approximated by lighting offsets. For the comparison, we begin with a simple flat surface illuminated by a point light source (first row). The second row shows the ground truth for a straight lighting effect, where the lighting transform function is applied to the light vector of original lighting result. The third row shows an approximation of the straight lighting result by the offset function, using 10 key offset data calculated from the differences between the original lighting and straight lighting results. 5 in-between frames shown in the third row were interpolated from the key offset data. The fourth row of illustrates the regions where the lighting offset errors are larger than 0.04. The red regions show positive errors and the blue region show negative errors. We can observe that the red regions are caused by insufficient blended offset data, where the brightness values are less than those of the lighting transform result. The blue regions are caused by the original lighting distributions, where the strong circular lights are placed near the blue regions.

Table 7.1 summarizes the maximum lighting offset errors of in-between frames with various numbers of key offset data. We observed that the quality of the lighting offsets depended heavily on the number of key offset data used. Even for this simple model, 10 key offset data were insufficient to approximate the straight lighting result. To obtain a smooth flow of the straight lighting effect, we needed at least 40 key offset data. This means that use of the offset function requires numerous time-consuming paint operations to obtain a straight lighting effect. Therefore, the lighting transform approach is more suitable for a straight lighting effect.

On the other hand, the lighting offsets cannot be fully replaced by lighting transforms. In designing the shading for cartoon animation, it is often necessary to adjust the lighting in local areas in addition to adjusting the overall lighting shape. Figure 7.3 illustrates such an example where we used the edge and local lighting offsets for fine-tuning. This is easily designed by using the lighting offset approach which enables local lighting effects. However, it is difficult to design such local lighting effects using the lighting transform approach since it is limited to simple controls using its pre-defined transforms. Therefore, integration of these two approaches are desirable.

In summary, we have shown that the shape of lights can be easily deformed by the use of lighting transforms. On the other hand, the lighting offset approach is optimized for local lighting effects that cannot be achieved using lighting transforms. Therefore, we believe these two approaches are complementary.

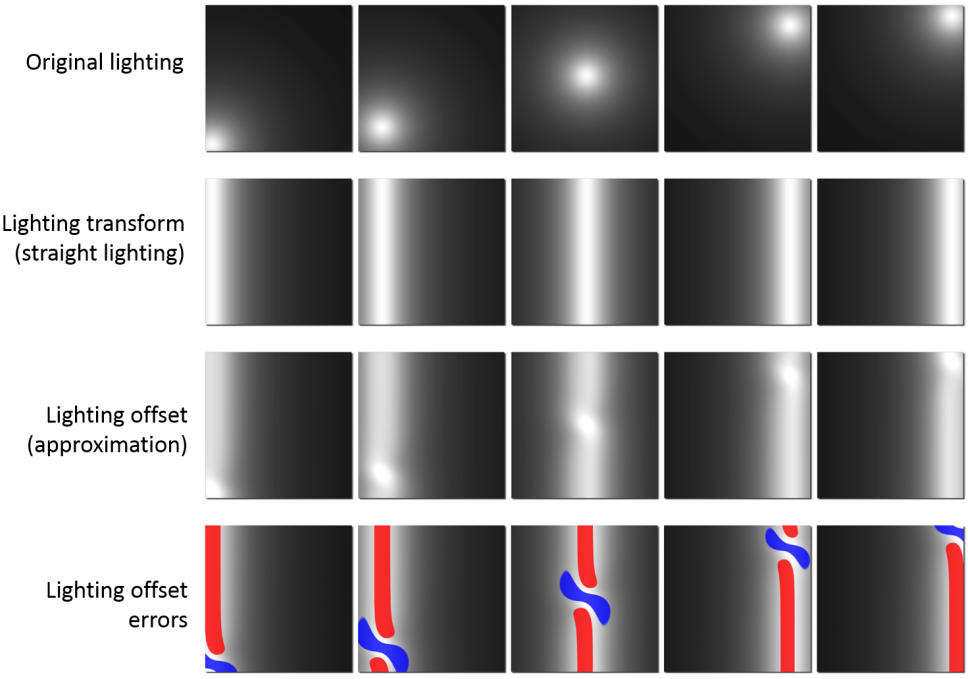


Figure 7.4: *Comparison of the lighting transform and lighting offset. (First row) Original lighting result obtained with a point light source. 5 frames were chosen from 48 frames. (Second row) Straight lighting result. The lighting transform was applied to the light vector of the original lighting result. (Third row) Approximation of the straight lighting result by use of the lighting offset. 10 key offset data, calculated from differences between the original lighting and straight lighting results, were used to approximate the vector transform. (Fourth row) Lighting offset errors. The red regions illustrate where the brightness differences are greater than* 0.04. *The blue regions illustrate where the brightness differences are less than* -0.04.

#Key offset data	Brightness error
5	0.386
10	0.188
20	0.0679
40	0.0221

Table 7.1: *Lighting offset errors as a function of the number of key offsets used to approximate the straight lighting effect. #Key offset data is the number of key offset data to approximate the straight lighting effect. The brightness errors are the maximum errors of in-between frames.*

7.3 Comparison of Lighting Offset Spaces

In our shading models, the lighting offset functions are designed for different spaces. In this section, we compare two representative examples: a local lighting offset defined on a local area of a surface (Chapter 4) and an edge lighting offset defined in an edge feature space (Chapter 5).

Figure 7.5 compares an edge enhancement that was controlled by using these two approaches. Similar to the previous section, we first introduced a ground truth result and then we approximated it. The first row shows the original lighting result obtained using the lighting transform to simulate a straight lighting effect. The second row shows the ground truth result, in which the edge enhancement was controlled by the edge lighting offset function defined in the edge distance field. This animation was quite simple, created from three key-frames using the edge lighting offset parameters. We then approximated this operation by using local lighting offsets defined on a local area of a surface. The third row shows 5 in-between frames interpolated from the key offset data. The fourth row of Figure 7.5 illustrates the regions where the local offset function errors are greater than 0.02.

Table 7.2 lists the local lighting offset errors as a function of the number of key offset data. Compared to the result in the previous section, the approximation errors are relatively small. We observed that 10 key offset data provided a visually sufficient approximation. However, we still needed a greater number of key offset data than the number of key-frames for the edge function. Also, the continuous change of the edge appearance would not be possible with a paint operation. Therefore, a specific model feature is better enhanced by a suitable lighting offset function defined in the model feature space.

On the other hand, local lighting offsets are still required for designing an arbitrary shape. Figure 7.3 illustrates such an example, where we used the local lighting offsets to adjust a small portion of the lighting shape. This could be easily achieved by a local lighting offset function constructed from a paint operation, whereas the offset function based on model features would be insufficient for representing the arbitrary shape beyond the limited feature space. Therefore, local lighting offsets cannot be replaced by a lighting offset based on a specific model feature.

In summary, enhancing an edge by using an offset function benefits by defining the offset function in a suitable model feature space. On the other hand, a local lighting offset function is still the most effective for designing an arbitrary shape. Both types of lighting offsets are desirable as directable controls.

#Key offset data	Brightness error
5	0.3743
10	0.1304
20	0.0386
40	0.0104

Table 7.2: *Local lighting offset errors as a function of the number of key offset data used to approximate the edge enhancement. #Key offset data is the number of key offset data to approximate the ground truth data. The brightness errors are the maximum errors of in-between frames.*

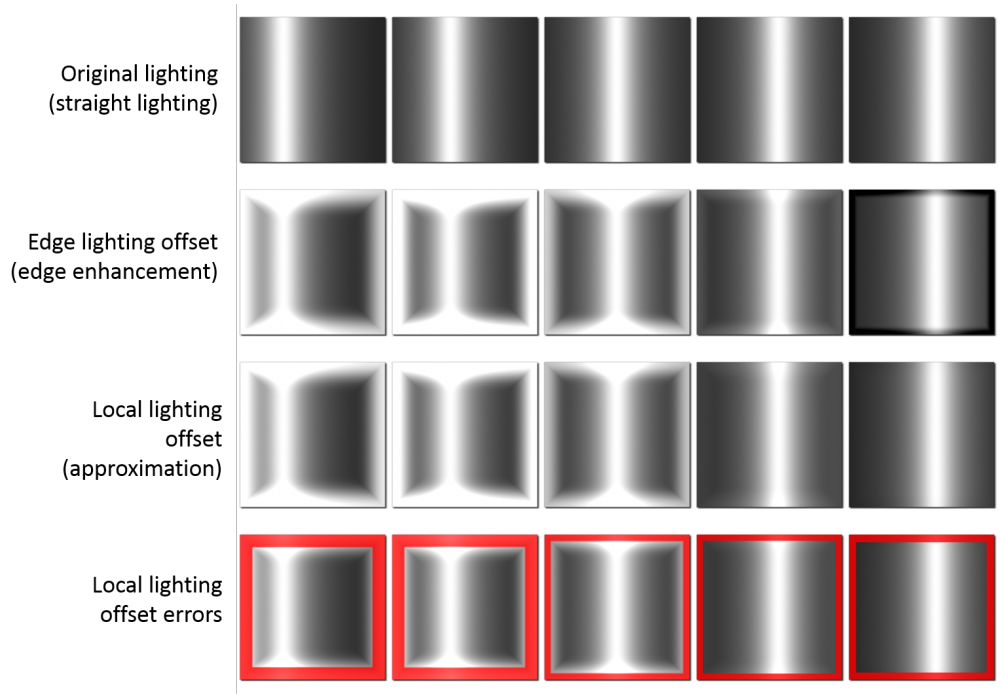


Figure 7.5: *Comparison of different lighting offset definitions. (First row) Original lighting result obtained using a straight lighting effect. 5 frames were chosen from 48 frames. (Second row) Edge enhancement result. Edge lighting offset function applied to the original lighting result. (Third row) Approximation of the edge enhancement result by use of the local lighting offset. 10 key offset data, calculated from differences between the original lighting and edge enhancement results, were used to approximate the edge lighting offset function. (Fourth row) Local lighting offset errors. The red regions illustrate where the brightness errors are greater than* 0.02.

7.4 Summary

In this chapter, we examined the capabilities of three directable mechanisms: 2D color mapping, lighting transform, and lighting offset. Each directable shading mechanism has a different degree of controllability. The comparisons in this chapter are summarized as follows:

- The 2D color mapping better emulates a complex color distribution better than the 1D color mapping, but it cannot be used with multiple light sources.
- The lighting transform is suitable for changing the overall shape of lighting than is the lighting offset, but it cannot give a local lighting effect like the lighting offset.
- The lighting offset function defined in a model feature space provides better controllability for a specific model feature than a lighting offset function defined on the local area of the surface, but it cannot be used to create an arbitrary lighting shape through a paint operation.

From these comparisons, we found that each directable shading mechanism is effective for a specific design target and complementary to other directable mechanisms. This suggests that our directable shading mechanisms are well-designed with suitable controls for each shading design target.

Chapter 8

Conclusion

In this chapter, we conclude the thesis by first summarizing our contributions, then discussing the limitations of our methods, and finally describing some possible directions for future research.

8.1 Summary of Contributions

In this thesis, we focused on how to improve the design process of stylized shading in 3D animations. We found that existing approaches for shading design are difficult to use intuitively and lack dynamic controls over the shading appearance. In many cases, the existing stylized rendering methods are insufficient to meet the demands of artists. Therefore, the artist must spend substantial time developing work-arounds, which only work in special cases. To address this problem, we proposed a new framework, *integration of artistic depictions with physics-based lighting*, for designing an artist-friendly stylized shading model and interface. The framework is based on the following two principles.

- **Principle 1: Directable shading model for artistic control**. We simplified difficult shading design process by introducing appropriate shading models that allow artistic control. Our shading models provide artists with not only easy and intuitive user interfaces but also interactive and dynamic controls, which are essential for the animation design process.
- **Principle 2: Seamless integration with 3D lighting**. The proposed shading models are carefully designed to be seamlessly integrated into existing 3D lighting controls. This allows artists to efficiently create animation with the desired shading appearances through use of 3D scene elements (models, lights, and cameras), with which they are already familiar.

To verify the effectiveness of this artist-friendly framework for stylized shading design, we provided three interactive systems that are appropriate for different levels of the design process, from small scale to large scale. We explored the capabilities of these systems to improve the stylized shading design in production work. These interactive systems are as follows.

- **Locally controllable shading with intuitive paint interface** (small scale). First, we presented a 3D stylized shading system for adding local light and shade using paint operations. The basic idea behind locally controllable shading is to modify the brightness term directly, by adding a lighting offset function. With the lighting offset, obtained from the painted area, our shading model provides additional

flexibility for changing an animated character's appearance. The proposed offset function mechanism is consistent and seamlessly integrated into the commonly used 3D lighting process, including multiple light sources and different types of lights (directional lights, points lights, and spot lights). We demonstrated with animation examples how our method allows the artists to design the desired shading appearance in making 3D animation.

- **Shading stylization based on model features** (middle scale). Our first system permitted only local control using a paint operation, which is not suitable for designing lighting enhancement of a specific feature of a model. Our second system provided easy and intuitive controls for such practical requirements. This method uses a lighting transforms to design a straight lighting effect. This effect can be enhanced by the edge enhancement and the detail lighting effect, which are provided through lighting offset functions defined in the feature spaces. These methods are seamlessly integrated into 3D lighting, with one exception: point lights must use our straight light functions. Even with this limitation, our animation examples demonstrate that artists can design commonly used feature enhancements.

- **Practical shading model for expressive shading styles** (large scale). This system focuses on the overall shading appearance, whereas the first and second systems are limited to simple shading tones of the conventional cartoon shading process. As an extension of the Lit-Sphere model, our shading model enables more expressive 2D shading tones for diffuse and specular effects. For control of light shape and correlated lighting effects, we reintroduced the lighting transforms and the lighting offsets in a manner that is suitable for a 2D color map representation of the shading model. Although this shading model allows for only a single light source, our animation examples demonstrate that the system is effective for designing many commonly used artistic styles.

We carefully designed these systems to fulfill the requirement of the proposed framework: integration of artistic depictions with physics-based lighting. Our systems allow the artists to design their expressive shading styles using an intuitive editing process, which would be difficult to accomplish by using only existing lighting controls or conventional stylized shading systems. In addition, the designed shading appearance is dynamically controlled, and seamlessly integrated into the 3D lighting. This provides a flexible and efficient shading design process. All of the animation examples designed using our systems indicate the effectiveness of the framework.

8.2 Limitations

While the proposed systems provide directable and efficient shading design mechanisms, our methods are still inadequate to fulfill the growing demands of artists.

For example, most of the stylized rendering using our methods is limited to shading effects. Our shading model cannot handle stroke-based rendering, which is used in many painterly rendering techniques (Figure 8.1). In our shading model, we have focused on providing directable controls for creating commonly desired shading. Our extensions are relatively small because we wanted to maintain the integration with the usual 3D lighting process. The lighting transforms and the lighting offset of our methods can handle only shape controls. Even our Lit-Sphere extension, which enables effective shading strokes for brush stroke styles, is still limited to shading effects.

On the other hand, typical painterly rendering methods can deal with overlapped strokes

Figure 8.1: *Limitation of our brush stroke styles. (Left) Brush stroke style obtained by using our approach. (Right) Overlapped brush stroke styles taken from painterly rendering methods. Our approach, while effective for emulating brush strokes, cannot fully support these kinds of overlapped brush stroke styles.*

to emulate brush-based artistic styles. Each stroke placement is defined by various properties: camera-space positions, user-defined density parameters, and brightness terms. These properties are also used to specify the color and the size of the brush strokes, which adjusts the painterly rendering style. In stroke-based rendering methods, shading is only one element that defines a brush stroke style, therefore it would be difficult to handle such stroke-based rendering styles using only our shading-based approach.

Another limitation is that our methods cannot deal with extreme changes form the initial lighting conditions. As discussed in Chapter 7, each directable shading mechanism has a different degree of controllability; thus, each must be carefully controlled in an appropriate manner that is suitable for each shading design requirement. Our shading control mechanisms, including the initial lighting, cannot be replace one other.

In summary, most styles designed using our systems are limited to shading effects, which are based on the commonly used shading models: cartoon shading and the the Lit-Sphere shading. Stroke-based rendering styles like those in painterly rendering systems would be difficult to model in our system, even with our extension of the Lit-Sphere. Our systems require appropriate controls of each shading mechanism to benefit from the capabilities of the proposed methods. Exploring the shading styles and control that artists want is essential for further improving both the effectiveness and usability of our artist-friendly framework.

8.3 Future Directions

In this thesis, we have focused on establishing an artist-friendly framework that is based on the integration of artistic depictions with physics-based lighting. Exploring a well-designed artist-friendly framework opens up several interesting avenues for future research in stylized rendering and its applications.

8.3.1 Example-based Shading Model from Painted Artwork

In the future, we would like to integrate more advanced example-based techniques into our shading models. In this thesis, we demonstrated that our methods provide controllability to obtain desired shading with suited design process. Although these directable mechanisms are effective for changing the designed shading appearance, artists may want a more rapid prototyping of shading style in the early stage.

Kulla et al. [49] and and Yen et al. [109] used manually painted examples to extract shading models for their painterly rendering styles. Their key idea was to separate brush strokes from shading tones. We could use a similar approach for our Lit-Sphere-based shading representation. For other lighting effects using our shading models, the learning approach used for pen-and-ink illustrations [46] might be appropriate. In their approach, painting examples are analyzed with a focus on line-drawing style. If, in the future, these kinds of example-based techniques are combined with our directable mechanisms, artists would be able to obtain a desired shading style more quickly.

8.3.2 Applying the Framework to Different Stylized Rendering Elements

Another direction for future research is to apply our artist-friendly design framework to other stylized rendering elements. In this thesis, we dealt only with two key stylized rendering elements: diffuse shading and specular highlights. It would be desirable to integrate directable mechanisms into other stylized rendering elements, such as shadows and contours.

One possible idea is to reintroduce models of shadows and contours using the same approach of our lighting transforms and offset functions. However, the models would need to handle discontinuous occlusions. It is very challenging to establish temporally coherent directable mechanisms against discontinuous properties. Our approach described in this thesis will provide a good start point for such further researches.

8.3.3 Stylized Control for Realistic Shading

In this thesis, we focused on designing stylized shading styles, mainly for cartoon shading. We believe that our stylized shading design methods are also useful for more realistic shading styles. One possible application is Hollywood cartoon animation films, where 3D characters are commonly designed with realistic shading styles, that clearly require directable artistic depictions.

For example, Disney explored physically-based shading techniques for their realistic shading styles [54, 62, 77]. It would be an interesting challenge to integrate our directable shading mechanisms into such realistic shading styles for further fine-tuning. We believe it is essential to pursue a well-designed artist-friendly system to make significant contributions to the improvement of the shading design process in production work as well as to the progress of rendering techniques for artists.

References

[1] Miika Aittala, Tim Weyrich, and Jaakko Lehtinen. Practical svbrdf capture in the frequency domain. *ACM Transactions on Graphics.*, 32(4):110:1–110:12, July 2013.

[2] David Akers, Frank Losasso, Jeff Klingner, Maneesh Agrawala, John Rick, and Pat Hanrahan. Conveying shape and features with image-based relighting. In *Proceedings of IEEE Visualization 2003*, pages 349–354, 2003.

[3] Ken Anjyo, Hideki Todo, and J. P. Lewis. A practical approach to direct manipulation blendshapes. *J. Graphics, GPU, & Game Tools*, 16(3):160–176, 2012.

[4] Ken Anjyo, Shuhei Wemler, and William Baxter. Tweakable light and shade for cartoon animation. In *Proceedings of NPAR 2006*, pages 133–139, 2006.

[5] Ken-ichi Anjyo and Katsuaki Hiramitsu. Stylized highlights for cartoon rendering and animation. *IEEE Comput. Graph. Appl.*, 23(4):54–61, July 2003.

[6] Michael Ashikhmin and Peter Shirley. An anisotropic phong brdf model. *J. Graphics. Tools*, 5(2):25–32, February 2000.

[7] Michael Ashikmin, Simon Premože, and Peter Shirley. A microfacet-based brdf generator. In *Proceedings of SIGGRAPH 2000*, pages 65–74, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[8] Autodesk, Inc. Autodesk 3ds max. `http://www.autodesk.com/products/autodesk-3ds-max`.

[9] Autodesk, Inc. Autodesk maya. `http://www.autodesk.com/products/autodesk-maya`.

[10] Autodesk, Inc. Autodesk softimage. `http://www.autodesk.com/products/autodesk-softimage`.

[11] Pascal Barla, Joëlle Thollot, and Lee Markosian. X-Toon: an extended toon shader. In *Proceedings of NPAR 2006*, pages 127–132, 2006.

[12] Pierre Bénard, Adrien Bousseau, and Joëlle Thollot. Dynamic solid textures for real-time coherent stylization. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games (I3D 2009)*, pages 121–127, New York, NY, USA, 2009. ACM.

[13] Pierre Bénard, Forrester Cole, Michael Kass, Igor Mordatch, James Hegarty, Martin Sebastian Senn, Kurt Fleischer, Davide Pesare, and Katherine Breeden. Stylizing animation by example. *ACM Transactions on Graphics. (Proceedings of SIGGRAPH 2013)*, 32(4):119:1–119:12, July 2013.

[14] Pierre Bénard, Ares Lagae, Peter Vangorp, Sylvain Lefebvre, George Drettakis, and Joëlle Thollot. A dynamic noise primitive for coherent stylization. *Com-*

puter Graphics Forum (Proceedings of the Eurographics Symposium on Rendering 2010), 29(4):1497–1506, june 2010.

[15] J. Blinn. Models of light reflection for computer synthesized pictures. *Computer Graphics*, 11(2):192–198, 1977.

[16] James F. Blinn. Simulation of wrinkled surfaces. *Proceedings of SIGGRAPH 1978*, 12(3):286–292, 1978.

[17] Stefan Bruckner and Meister Eduard Gröller. Style transfer functions for illustrative volume rendering. *Computer Graphics Forum. (Proceedings of Eurographics 2007)*, 26(3):715–724, 2007.

[18] Michael Bunnell. Dynamic ambient occlusion and indirect lighting. In *GPU Gems 2*. Addison-Wesley, 2005.

[19] Capcom Co., Ltd. Okami, 2006, 2012. `http://www.capcom.co.jp/o-kami/`.

[20] Jung-Ju Choi and Hwan-Jik Lee. Rendering stylized highlights using projective textures. *Visual Computer*, 22(9):805–813, September 2006.

[21] A.N.M. Imroz Choudhury and Steven G. Parker. Ray tracing npr-style feature lines. In *Proceedings of NPAR 2009*, pages 5–14, New York, NY, USA, 2009. ACM.

[22] Mark Colbert, Sumanta Pattanaik, and Jaroslav Krivanek. Brdf-shop: Creating physically correct bidirectional reflectance distribution functions. *IEEE Comput. Graph. Appl.*, 26:30–36, January 2006.

[23] R. L. Cook and K. E. Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics.*, 1(1):7–24, January 1982.

[24] Robert L. Cook. Shade trees. In *Proceedings of SIGGRAPH 1984*, pages 223–231, New York, NY, USA, 1984. ACM.

[25] Eric Daniels. Deep canvas in disney's tarzan. In *ACM SIGGRAPH 99 Conference abstracts and applications*, pages 200–, New York, NY, USA, 1999.

[26] Paul Debevec, Tim Hawkins, Chris Tchou, Haarm-Pieter Duiker, Westley Sarokin, and Mark Sagar. Acquiring the reflectance field of a human face. In *Proceedings of SIGGRAPH 2000*, pages 145–156, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[27] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. Suggestive contours for conveying shape. *ACM Transactions on Graphics. (Proceedings of SIGGRAPH 2003)*, 22(3):848–855, July 2003.

[28] Christopher DeCoro, Forrester Cole, Adam Finkelstein, and Szymon Rusinkiewicz. Stylized shadows. In *Proceedings of NPAR 2007*, pages 77–83, New York, NY, USA, 2007. ACM.

[29] Oliver Deussen, Stefan Hiller, Cornelius van Overveld, and Thomas Strothotte. Floating points: A method for computing stipple drawings. *Computer Graphics Forum*, 19:40–51, 2000.

[30] Oliver Deussen and Thomas Strothotte. Computer-generated pen-and-ink illustration of trees. In *Proceedings of SIGGRAPH 2000*, pages 13–18, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[31] Yue Dong, Jiaping Wang, Xin Tong, John Snyder, Yanxiang Lan, Moshe Ben-Ezra, and Baining Guo. Manifold bootstrapping for svbrdf capture. *ACM Transactions on Graphics. (Proceedings of SIGGRAPH 2010)*, 29(4):98:1–98:10, July 2010.

[32] Craig Donner, Jason Lawrence, Ravi Ramamoorthi, Toshiya Hachisuka, Henrik Wann Jensen, and Shree Nayar. An empirical bssrdf model. *ACM Transactions on Graphics. (Proceedings of SIGGRAPH 2009)*, 28(3):30:1–30:10, July 2009.

[33] J. Duchon. Splines minimizing rotation-invariant semi-norms in sobolev spaces. In *Constructive Theory of Functions of Several Variables number 571 in Lecture Notes in Mathematics*, pages 85–100. Springer-Verlag, 1977.

[34] Frdo Durand, Victor Ostromoukhov, Mathieu Miller, Francois Duranleau, and Julie Dorsey. Decoupling strokes and high-level attributes for interactive traditional drawing. In *Proceedings of Eurographics Workshop on Rendering Techniques*, pages 71–82, London, 2001. Springer.

[35] Robert W. Floyd and Louis Steinberg. An Adaptive Algorithm for Spatial Greyscale. In *Proceedings of the Society for Information Display*, volume 17, pages 75–77, 1976.

[36] Bruce Gooch and Amy Gooch. *Non-Photorealistic Rendering*. AK Peters Ltd, 2001.

[37] Bruce Gooch, Peter-Pike J. Sloan, Amy Gooch, Peter Shirley, and Richard Riesenfeld. Interactive technical illustration. In *Proceedings of I3D 1999*, pages 31–38, New York, NY, USA, 1999. ACM.

[38] Pat Hanrahan and Wolfgang Krueger. Reflection from layered surfaces due to subsurface scattering. In *Proceedings of SIGGRAPH 1993*, pages 165–174, New York, NY, USA, 1993. ACM.

[39] Aaron Hertzmann and Denis Zorin. Illustrating smooth surfaces. In *Proceedings of SIGGRAPH 2000*, pages 517–526, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[40] Burne Hogarth. *Dynamic Light and Shade*. Watson-Guptill, 1991.

[41] Piti Irawan and Steve Marschner. Specular reflection from woven cloth. *ACM Transactions on Graphics. (Proceedings of SIGGRAPH 2012)*, 31(1):11:1–11:20, February 2012.

[42] Tilke Judd, Frédo Durand, and Edward Adelson. Apparent ridges for line drawing. *ACM Transactions on Graphics. (Proceedings of SIGGRAPH 2007)*, 26(3), July 2007.

[43] Kaikai Kiki Co., Ltd. Kaikai&Kiki, 2008.

[44] R.D. Kalnins, L. Markosian, B.J. Meier, M.A Kowalski, J.C. Lee, P.L. Davivn, M.Webb, J.F. Hughes, and A. Finkelstein. WYSIWYG NPR: drawing strokes directly on 3D models. *ACM Transactions on Graphics. (Proceedings of SIGGRAPH 2002)*, 21(3):755–762, 2002.

[45] Robert D. Kalnins, Philip L. Davidson, Lee Markosian, and Adam Finkelstein. Coherent stylized silhouettes. *ACM Transactions on Graphics. (Proceedings of SIGGRAPH 2003)*, 22(3):856–861, July 2003.

[46] Evangelos Kalogerakis, Derek Nowrouzezahrai, Simon Breslav, and Aaron Hertzmann. Learning hatching for pen-and-ink illustration of surfaces. *ACM Transactions on Graphics.*, 31(1):1:1–1:17, February 2012.

[47] Michael Kass and Davide Pesare. Coherent noise for non-photorealistic rendering. *ACM Transactions on Graphics.*, 30(4):30:1–30:6, July 2011.

[48] John K. Kawai, James S. Painter, and Michael F. Cohen. Radioptimization: goal based rendering. In *Proceedings of SIGGRAPH 1993*, Computer Graphics Proceedings, Annual Conference Series, pages 147–154, 1993.

[49] Christopher D. Kulla, James D. Tucek, Reynold J. Bailey, and Cindy M. Grimm. Using texture synthesis for non-photorealistic shading from paint samples. In *Proceedings of PG 2003*, pages 477–481, Washington, DC, USA, 2003. IEEE Computer Society.

[50] Adam Lake, Carl Marshall, Mark Harris, and Marc Blackstein. Stylized rendering techniques for scalable real-time 3D animation. In *Proceedings of NPAR 2000*, pages 13–20, New York, NY, USA, 2000. ACM Press.

[51] Chang Ha Lee, Xuejun Hao, and Amitabh Varshney. Geometry-dependent lighting. *IEEE Transactions on Visualization and Computer Graphics*, 12(2):197–207, 2006.

[52] Yunjin Lee, Lee Markosian, Seungyong Lee, and John F. Hughes. Line drawings via abstracted shading. *ACM Transactions on Graphics. (Proceedings of SIGGRAPH 2007)*, 26(3), July 2007.

[53] Lee Markosian, Michael A. Kowalski, Daniel Goldstein, Samuel J. Trychin, John F. Hughes, and Lubomir D. Bourdev. Real-time nonphotorealistic rendering. In *Proceedings of SIGGRAPH 1997*, pages 415–420, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[54] Stephen McAuley, Stephen Hill, Naty Hoffman, Yoshiharu Gotanda, Brian Smits, Brent Burley, and Adam Martinez. Practical physically-based shading in film and game production. In *ACM SIGGRAPH 2012 Courses*, pages 10:1–10:7, New York, NY, USA, 2012. ACM.

[55] Michael D. McCool, Jason Ang, and Anis Ahmad. Homomorphic factorization of brdfs for high-performance rendering. In *Proceedings of SIGGRAPH 2001*, pages 171–178, New York, NY, USA, 2001. ACM.

[56] Barbara J. Meier. Painterly rendering for animation. In *Proceedings of SIGGRAPH 1996*, pages 477–484, New York, NY, USA, 1996. ACM.

[57] V. B. Mello, C. R. Jung, and M. Walter. Virtual woodcuts from images. In *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, GRAPHITE '07, pages 103–109, New York, NY, USA, 2007. ACM.

[58] Jason Mitchell, Moby Francke, and Dhabih Eng. Illustrative rendering in Team Fortress 2. In *Proceedings of NPAR 2007*, pages 71–76, New York, NY, USA, 2007. ACM.

[59] Namco Bandai Games Inc. Dragon ball z: Ultimate tenkaichi, 2011. `http://b.bngi-channel.jp/dba/`.

[60] NewTek, Inc. Lightwave 3d. `http://www.lightwave3d.com/`.

[61] J. D. Northrup and Lee Markosian. Artistic silhouettes: a hybrid approach. In *Proceedings of NPAR 2000*, pages 31–37, New York, NY, USA, 2000. ACM.

[62] Derek Nowrouzezahrai, Jared Johnson, Andrew Selle, Dylan Lacewell, Michael Kaschalk, and Wojciech Jarosz. A programmable system for artistic volumetric lighting. *ACM Transactions on Graphics. (Proceedings of SIGGRAPH 2011)*, 30(4):29:1–29:8, July 2011.

[63] Makoto Okabe, Yasuyuki Matsushita, Li Shen, and Takeo Igarashi. Illumination brush: Interactive design of all-frequency lighting. In *Pacific Graphics 2007*, pages 171–180, Washington, DC, USA, 2007. IEEE Computer Society.

[64] Manuel M. Oliveira, Gary Bishop, and David McAllister. Relief texture mapping. In *Proceedings of SIGGRAPH 2000*, pages 359–368, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[65] OLM Digital, Inc. OLM Digital R&D. `http://www.olm.co.jp/rd/`.

[66] Victor Ostromoukhov. Digital facial engraving. In *Proceedings of SIGGRAPH 1999*, pages 417–424, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[67] Victor Ostromoukhov and Roger D. Hersch. Multi-color and artistic dithering. In *Proceedings of SIGGRAPH 1999*, pages 425–432, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[68] Romain Pacanowski, Xavier Granier, Christophe Schlick, and Pierre Poulin. Sketch and Paint-based Interface for Highlight Modeling. In *Proceedings of SBIM 2008*, Annecy, France, 2008.

[69] Fabio Pellacini, Parag Tole, and Donald P. Greenberg. A user interface for interactive cinematic shadow design. *ACM Transactions on Graphics. (Proceedings of SIGGRAPH 2002)*, 21(3):563–566, 2002.

[70] Matt Pharr and Simon Green. Ambient occlusion. In *GPU Gems*. Addison-Wesley, 2004.

[71] Bui Tuong Phong. Illumination for computer generated pictures. *Commun. ACM*, 18(6):311–317, June 1975.

[72] Production I.G., Sanzigen and Ishimori Productions. 009 re:cyborg, 2012. `http://009.ph9.jp/`.

[73] Tobias Ritschel, Kaleigh Smith, Matthias Ihrke, Thorsten Grosch, Karol Myszkowski, and Hans-Peter Seidel. 3D unsharp masking for scene coherent enhancement. In *ACM Transactions on Graphics. (Proceedings of SIGGRAPH 2008)*, pages 1–8, New York, NY, USA, 2008. ACM.

[74] Tobias Ritschel, Thorsten Thormählen, Carsten Dachsbacher, Jan Kautz, and Hans-Peter Seidel. Interactive on-surface signal deformation. *ACM Transactions on Graphics. (Proceedings of SIGGRAPH 2010)*, 29(4):Article 36, 2010.

[75] Szymon Rusinkiewicz, Michael Burns, and Doug DeCarlo. Exaggerated shading for depicting shape and detail. *ACM Transactions on Graphics. (Proceedings of SIGGRAPH 2006)*, 25(3):1199–1205, 2006.

[76] Iman Sadeghi, Oleg Bisker, Joachim De Deken, and Henrik Wann Jensen. A practical microcylinder appearance model for cloth rendering. *ACM Transactions on Graphics.*, 32(2):14:1–14:12, April 2013.

[77] Iman Sadeghi, Heather Pritchett, Henrik Wann Jensen, and Rasmus Tamstorf. An artist friendly hair shading system. *ACM Transactions on Graphics. (Proceedings of SIGGRAPH 2010)*, 29(4):56:1–56:10, July 2010.

[78] Takafumi Saito and Tokiichiro Takahashi. Comprehensible rendering of 3-d shapes. In *Proceedings of SIGGRAPH 1990*, pages 197–206, New York, NY, USA, 1990. ACM.

[79] Michael P. Salisbury, Sean E. Anderson, Ronen Barzel, and David H. Salesin. Interactive pen-and-ink illustration. In *Proceedings of SIGGRAPH 1994*, pages 101–108, New York, NY, USA, 1994. ACM.

[80] Michael P. Salisbury, Michael T. Wong, John F. Hughes, and David H. Salesin. Orientable textures for image-based pen-and-ink illustration. In *Proceedings of SIGGRAPH 1997*, pages 401–406, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[81] Mike Salisbury, Corin Anderson, Dani Lischinski, and David H. Salesin. Scale-dependent reproduction of pen-and-ink illustrations. In *Proceedings of SIGGRAPH 1996*, pages 461–468, New York, NY, USA, 1996. ACM.

[82] Pedro V. Sander, Xianfeng Gu, Steven J. Gortler, Hugues Hoppe, and John Snyder. Silhouette clipping. In *Proceedings of SIGGRAPH 2000*, SIGGRAPH '00, pages 327–334, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[83] Yoichi Sato, Mark D. Wheeler, and Katsushi Ikeuchi. Object shape and reflectance modeling from observation. In *Proceedings of SIGGRAPH 1997*, pages 379–387, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[84] Johannes Schmid, Martin Sebastian Senn, Markus Gross, and Robert W. Sumner. Overcoat: an implicit canvas for 3d painting. *ACM Transactions on Graphics. (Proceedings of SIGGRAPH 2011)*, 30(4):28:1–28:10, July 2011.

[85] Chris Schoeneman, Julie Dorsey, Brian Smits, James Arvo, and Donald Greenburg. Painting with light. In *Proceedings of SIGGRAPH 1993*, Computer Graphics Proceedings, Annual Conference Series, pages 143–146, 1993.

[86] Adrian Secord. Weighted voronoi stippling. In *Proceedings of NPAR 2002*, pages 37–43, New York, NY, USA, 2002. ACM.

[87] Peter-Pike J. Sloan, William Martin, Amy Gooch, and Bruce Gooch. The lit sphere: a model for capturing npr shading from art. In *Proceedings of Graphics Interface 2001*, pages 143–150, 2001.

[88] Mumehiro Tada, Yoshinori Dobashi, and Tsuyoshi Yamamoto. Feature-based interpolation for the interactive editing of shading effects. In *Proceedings of the 11th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry (VRCAI 2012).*, pages 47–50, New York, NY, USA, 2012. ACM.

[89] Hideki Todo and Ken Anjyo. Hybrid framework for blendshape manipulations. In *SIGGRAPH Asia 2011 Posters*, pages 40:1–40:1, New York, NY, USA, 2011. ACM.

[90] Hideki Todo, Ken Anjyo, William Baxter, and Takeo Igarashi. Locally controllable stylized shading. *ACM Transactions on Graphics. (Proceedings of SIGGRAPH 2007)*, 26(3):Article 17, 2007.

[91] Hideki Todo, Ken Anjyo, and Takeo Igarashi. Stylized lighting for cartoon shader. *Comput. Animat. Virtual Worlds*, 20(2- 3):143–152, June 2009.

[92] Hideki Todo, Ken Anjyo, and Shun'ichi Yokoyama. Lit-sphere extension for artistic rendering. *The Visual Computer*, 29(6-8):473–480, 2013.

[93] TOHO (Distributor) and BANDAI, WiZ, Namco Bandai Games, SHOGAKUKAN, ADK, OLM, BANDAI NETWORKS (Production Studios). Tamagotchi: Happiest Story in the Universe!, 2008. `http://tamaeiga.com/`.

[94] Borom Tunwattanapong, Graham Fyffe, Paul Graham, Jay Busch, Xueming Yu, Abhijeet Ghosh, and Paul Debevec. Acquiring reflectance and shape from continuous spherical harmonic illumination. *ACM Transactions on Graphics.*, 32(4):109:1–109:12, July 2013.

[95] Greg Turk and James F. O'Brien. Shape transformation using variational implicit functions. In *Proceedings of SIGGRAPH 1999*, Computer Graphics Proceedings, Annual Conference Series, pages 335–342, 1999.

[96] David Vanderhaeghe, Romain Vergne, Pascal Barla, and William Baxter. Dynamic stylized shading primitives. In *Proceedings of NPAR 2011*, pages 99–104, New York, NY, USA, 2011. ACM.

[97] Romain Vergne, Pascal Barla, Xavier Granier, and Christophe Schlick. Apparent relief: a shape descriptor for stylized shading. In *Proceedings of NPAR 2008*, pages 23–29, New York, NY, USA, 2008. ACM.

[98] Romain Vergne, Romain Pacanowski, Pascal Barla, Xavier Granier, and Christophe Schlick. Light warping for enhanced surface depiction. *ACM Transactions on Graphics. (Proceedings of SIGGRAPH 2009)*, 28(3):25:1–25:8, July 2009.

[99] Romain Vergne, Romain Pacanowski, Pascal Barla, Xavier Granier, and Christophe Schlick. Radiance scaling for versatile surface enhancement. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games (I3D 2010)*, pages 143–150, New York, NY, USA, 2010. ACM.

[100] Romain Vergne, Romain Pacanowski, Pascal Barla, Xavier Granier, and Christophe Schlick. Improving Shape Depiction under Arbitrary Rendering. *IEEE Transactions on Visualization and Computer Graphics*, 17(8):1071 – 1081, June 2011.

[101] Grace Wahba. *Spline Models for Observational Data*. SIAM, 1990.

[102] Walt Disney Animation Studios and Walt Disney Pictures. Paperman, 2012. `http://www.disneyanimation.com/projects/paperman`.

[103] Jiaping Wang, Shuang Zhao, Xin Tong, John Snyder, and Baining Guo. Modeling anisotropic surface reflectance with example-based microfacet synthesis. *ACM Transactions on Graphics. (Proceedings of SIGGRAPH 2008)*, 27(3):41:1–41:9, August 2008.

[104] Gregory J. Ward. Measuring and modeling anisotropic reflection. In *Proceedings of SIGGRAPH 1992*, pages 265–272, New York, NY, USA, 1992. ACM.

[105] Tim Weyrich, Jason Lawrence, Hendrik P. A. Lensch, Szymon Rusinkiewicz, and Todd Zickler. Principles of appearance acquisition and representation. *Found. Trends. Comput. Graph. Vis.*, 4(2):75–191, February 2009.

[106] Brian Whited, Eric Daniels, Michael Kaschalk, Patrick Osborne, and Kyle Odermatt. Computer-assisted animation of line and paint in disney's paperman. In *ACM SIGGRAPH 2012 Talks*, pages 19:1–19:1, New York, NY, USA, 2012. ACM.

[107] Holger Winnemöller and Shaun Bangay. Geometric approximations towards free specular comic shading. *Computer Graphics Forum. (Proceedings of Eurographics 2002)*, 21(3):309–316, 2002.

[108] Holger Winnemöller, Jan Eric Kyprianidis, and Sven C. Olsen. Xdog: An extended difference-of-gaussians compendium including advanced image stylization. *Computers & Graphics*, 36(6):740–753, 2012.

[109] Chung-Ren Yen, Ming-Te Chi, Tong-Yee Lee, and Wen-Chieh Lin. Stylized rendering using samples of a painted image. *IEEE Transactions on Visualization and Computer Graphics*, 14(2):468–480, March 2008.

Appendix A

Additional Examples

In this appendix, we present additional examples where we combined the shading effects of the three independent systems in Chapters 4-6. While we have not yet implemented a single unified system to combine our systems, we can combine some of the proposed shading effects for off-line rendering. In the off-line rendering process, our directable shading mechanisms are implemented as node components, which provide more flexible shading functionalities by making networks of nodes. In making these examples, we first separately design shading effects using each system, and then combine the effects making use of Maya's off-line rendering framework.

A.1 Implementation

As described in Chapters 4-6, our systems are based on Maya's hardware shading functionality that allows GPU shading process. This offers interactive shading process; however, the implemented shading pipelines are processed independent from Maya's off-line rendering process, which is used for final rendering. This off-line rendering process is based on node components, which provide more flexible shading mechanisms by making networks of nodes. To integrate our shading effects into the off-line rendering process, we implemented some of the our directable shading mechanisms as Maya software rendering node plug-ins. In our off-line rendering process, we make use of the following node plug-ins and built-in features.

- *Dynamic Lit-Sphere node*. This node provides dynamic Lit-Sphere shading process. It computes the projection coordinates (u,v) of the Lit-Sphere shading according to the inputs of light and view setting and the geometry of a target model. We also provide a user controlled parameter that interpolate diffuse and specular shading behaviors.
- *Lighting transform node*. This node transforms the lighting shape for the 2D coordinate representation of a dynamic Lit-Sphere node. It allows the artist to control the lighting shape intuitively based on the simple transform functions such as translations, directional scaling, and rotations.
- *Lighting offset node*. This deforms the brightness for the 2D coordinate representation of a dynamic Lit-Sphere node. By changing input attributes, various small scale stylization can be designed.
- *Object space edge field node*. In our current implementation for off-line rendering process, edge field is computed in object space, which is much slower than the GPU-based image space algorithm but more suitable for Maya's off-line rendering

framework. Similar to the image space edge field described in Chapter 5, we provide a parameter to control the thickness. The edge field value is used for a lighting offset node to design an edge enhancement effect.

- *Local lighting offset texture (built-in feature)*. In our off-line rendering process, local lighting effects are stored into 2D textures. These textures are simply constructed from the per-vertex key-frame local lighting offset data used in Chapter 4. Local lighting offset data between key-frames are interpolated from key-frame offset textures used simple linear functions. The final offset data is used for a lighting offset node to add a local lighting effect.

In the following, we experimentally apply these nodes and components to combine some of the proposed effects.

A.2 Results

In making a facial animation, local lighting effects are effective to change the character's impression. Figure A.1 shows typical cases where we designed further small scale stylizations by applying brush strokes styles to the designed local lighting effects. To create these styles, we first designed the local shading effects to obtain desired shading appearances by painting operations as described in Chapter 4. Then we applied brush stroke styles to the designed local lighting effects using lighting offset approach described in Section 6.6.2. The examples in the figure demonstrates that our lighting offset mechanism also works well for combining brush stroke styles with local lighting effects to apply small scale stylizations to the original shading.

In designing a shading style for mechanical objects, controlling edge appearance is crucial. Figure A.2 demonstrates how our off-line rendering enables edge enhancements as in Chapter 5 and the integration of expressive shading styles as in Chapter 6. As shown in the original shading results of the figure, a variety of shading styles can be designed even on flat surfaces using our dynamic Lit-Sphere approach with a single point light source. In addition, our lighting offset mechanism can deal with edge enhancement effects taking the edge field as an input attribute of offsetting process. The examples in the figure suggests that combining these directable mechanisms have a flexibility to control more detailed shading behaviors, which would be difficult to achieve the same effect using a single system.

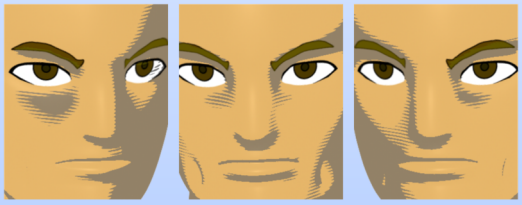
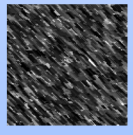
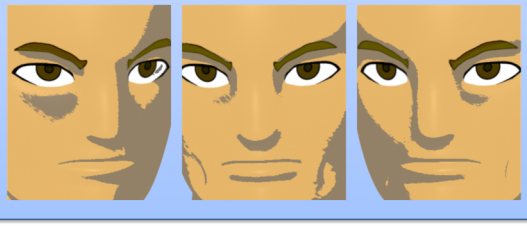
Title	Brush texture	Shading result
Original lighting (local lighting offset)		
Simple hatching style (Shading strokes)		
Simple brush stroke style (Shading strokes)		

Figure A.1: *Brush stroke styles for local lighting effects. Brush texture is a 2D structured texture to specify a brush stroke style. (Top) Original lighting result. The local lighting effects were designed by user paint operations as described in Chapter 4. (Middle and bottom) Brush stroke styles combined with the original lighting result. Using our shading strokes in Section 6.6.2, we applied brush stroke styles to the local lighting effects.*

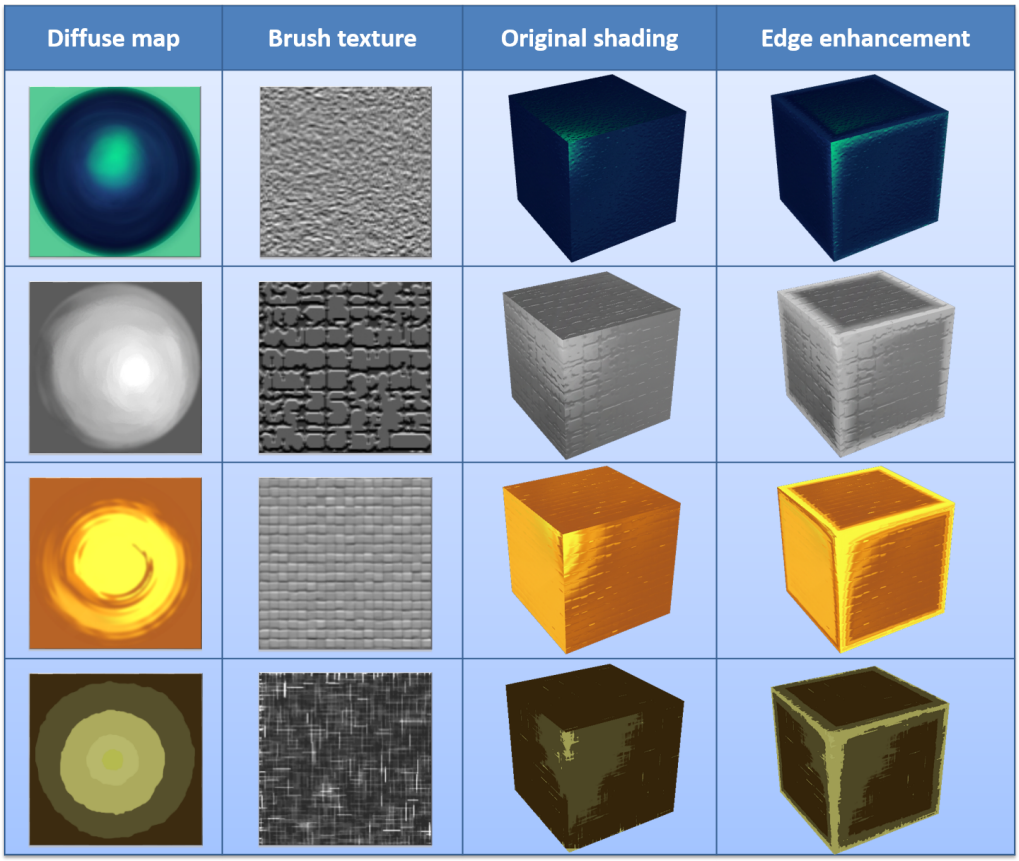


Figure A.2: *Edge enhancements for expressive shading styles. Diffuse map is a Lit-Sphere map to design the diffuse shading effect. Brush texture is a 2D structured texture to specify a brush stroke style. Each diffuse map and brush texture were used to create the original shading result illuminated by a single point light source. We then applied edge enhancement effects to these original lighting results.*