QUALITATIVE GLOBAL ILLUMINATION OF MOCK-3D SCENES

A Dissertation

by

YOUYOU WANG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Ergun Akleman
Co-Chair of Committee,	Dezhen Song
Committee Members,	Scott Schaefer
	John Keyser
Head of Department,	Nancy M. Amato

August 2014

Major Subject: Computer Engineering

ABSTRACT

In this work, we developed a framework to obtain qualitatively acceptable global rendering effects without an explicit geometry. This framework is particularly useful for 2D artists such as painters and illustrators. They will be able to obtain 3D looking images with complete artistic control as if they are using a 2D digital image manipulation system.

The core of this approach is a mock-3D scene representation that allows impossible or stylized shapes as "fuzzy" geometric structures. These fuzzy geometric structures are view dependent shapes that are computed from texture maps which provide normal, thickness and displacement information for all visible points of a shape. The information that is provided by these texture maps, which we call shape maps, do not have to be complete or consistent. Shape maps can be obtaining by (1) converting 3D shapes into 2D images, (2) modeling using a sketch based interface, (3) directly painting a gradient domain image or (4) photographing real objects. The most interesting shape maps are those sketched or painted by an artist, since they can reflect the artist's intention, even if this does not follow the normal rules of perspective.

The major advantage of this approach is the ability to obtain visually acceptable global effects even when shape maps do not correspond to real 3D shapes. We show that computing view dependent fuzzy geometry from shape maps is sufficient to obtain qualitatively convincing global illumination effects even for impossible shapes. The methods we have developed and implemented for global rendering effects include ambient occlusion, local and global shadows, refraction and reflection. Although these methods do not directly correspond to underlying physical phenomena, they

can provide results that are qualitative proportional to 3D realistic rendering.

Our approach is a very 2D artist-friendly representation since the shaders are also defined as images. These images can naturally describe shading parameters and provide a simple 2D control of the shading and rendering processes to intuitively obtain desired visual results. In particular, this representational power helps to easily obtain a wide variety of NPR effects that is still consistent with global illumination.

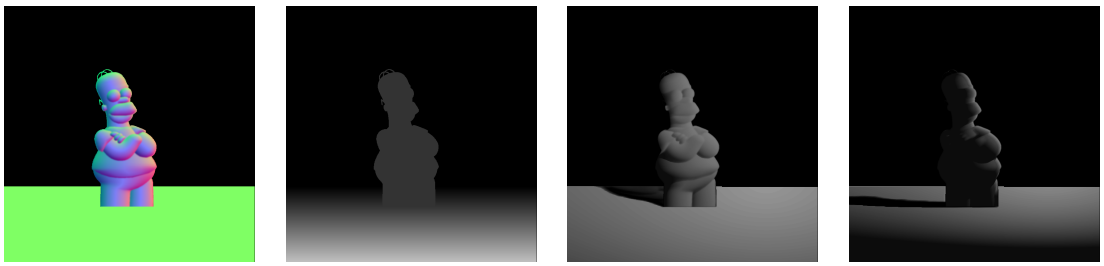
TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

1. INTRODUCTION AND MOTIVATION

Our qualitative rendering approach using mock-3D scenes is inspired by the popularity of 2D image editing tools, such as photoshop and illustrator. Nowadays, the 3D modeling market does not grow as rapidly as 2D painting/editing market[12]. Many people are reluctant to use 3D, since it needs more training and is less intuitive than in 2D. And most importantly, if the fake 3D effects that 2D painting/editing provides are good enough, people will still tend to understand these images as if they are in 3D world. And in order to bridge the gap between 2D painting and 3D rendering, an image that contains 3D information is usually used, such as normal maps and bas-relief. We call these images mock-3D shapes, which are not really 3D but appear 3D.

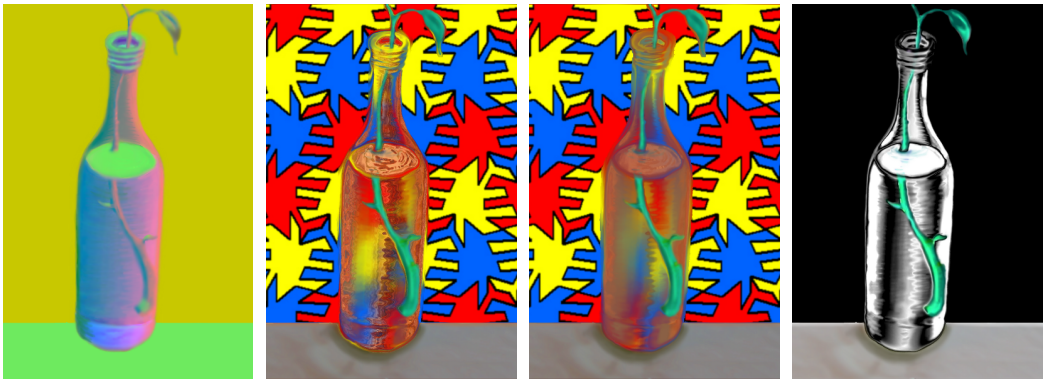


(a) A Mock 3D Scene "Image" (b) Depth Image of Mock 3D scene (c) Shadow casted by a front light (d) Shadow casted by a side light

Figure 1.1: Qualitative rendering with mock-3D shapes - This figure demonstrates obtaining global shadows using two-sided Mock-3D shapes. Since they can cast shadow from both sides, these shapes can provide volumetric shadows as shown in (d)

Figure 1.1 shows an example of qualitative rendering of a mock-3D scene that

consists of mock-3D shapes. As shown in this example, although the shapes are not really 3D, we can obtain realistic looking global shadows. Figure 1.2 shows even a single Mock-3D shape image that is included in a 2D digital painting program can be sufficient to obtain a wide variety of refraction and reflection effects. With multi-layers, we can have reflection of refraction effect, as shown in Figure 1.3.

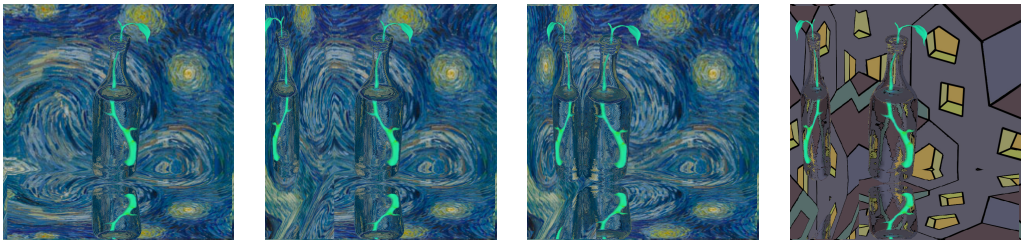


(a) Mock 3D image (b) Refraction & Reflection (c) Translucent & Glossy (d) Fresnel effect only

Figure 1.2: Refraction and reflection with a single mock-3D shape - The particular mock-3D shape in (a) painted by an artist based on a photograph of a real bottle. It is used to reflect and refract a background and an environment image. (b) shows reflection and refraction composited with Fresnel term. (c) shows glossy reflection and translucent refraction combined with a Fresnel and (d) directly shows Fresnel effects using a black background and white environment map

There currently exists two representations for such mock 3D shapes: bas-reliefs and normal maps. In this work, we introduce a fuzzy and view dependent shape representation that is suitable for global illumination and provides the representational powers of both bas-reliefs and normal maps. Our approach can be better explained with a closer look to normal maps.

Normal maps became popular as soon as they are introduced in 1998 [3]. Al-



(a) Reflection of Refraction (b) Two Reflective Planes (c) Moved Object (d) Different Background

Figure 1.3: Refraction and reflection with multiple mock-3D shapes - In (a) there is only one horizontal reflective plane on the ground, (b) has another vertical reflective plane on the left, (c) is a different scene when object moved, (d) is the scene with a different background

though,they are mainly used as texture maps to include details to polygonal meshes, they can directly be used as shape representations as shown by Johnston [15]. He developed a sketch based system, called Lumo, to model normal maps as mock-3D shapes by diffusing 2D normals in a line drawing. Despite the potential power of normal maps as shape representations, only a few groups investigated the potential use of normal maps as mock-3D shapes [19, 2, 31, 24]. All these methods focused on "modeling" of these normal maps. To render these mock-3D shapes, only basic 3D rendering methods are employed. The global rendering effects such as shadow, occlusion, reflection and refractions has never been included. Recently, Sykora et al. [27] developed a user-assisted method to convert normal maps into Bas-Reliefs that can provide correct shadows in a commercial renderer. This approach is useful if the normal maps corresponds to shapes that can have an explicitly meaningful 3D geometry.

Normal maps are just images, therefore, they do not necessarily correspond to any meaningful geometry. This is, in fact, an advantage for 2D artists such as painters or illustrators, who wants to experiment with unusual shapes. Normal maps are

perfect for such experiments since they can be created directly by artists and they can represent impossible or even incoherent shapes. To unleash the true power of this representation for 2D artists, there is a need for a new rendering framework, that is designed to provide artists with full control over the consistent global rendering effects.

In this work, we present such a gradient domain rendering framework: we obtain qualitatively acceptable global rendering effects without an explicit geometry. Our approach is based on construction of a fuzzy geometry that can provide believable global effects such as shadows, reflections and refraction. Our fuzzy geometry are computed from mock-3D scenes that consists of a set of planar quadrilaterals that are textured with a normal map, a thickness map, and an optional displacement map. Using the thickness maps, we obtain fuzzy but still more useful shapes than Bas-Reliefs. Additional thickness information helps us to make fuzzy shapes two-sided. They, therefore, can cast shadow from the invisible side. The resulting shadows looks more realistic (i.e. volumetric) when shadow is casted by a side lights. An example of such volumetric shadows in a mock-3D scene is shown in Figure 1.1(d). Another usage of two-sided shapes to obtain qualitatively correct refractions. The Figure 1.2 demonstrate another usage of thickness maps to control deformations caused by refraction. Using an optional displacement map, we further introduce discontinuities where a sharp change can not be captured by normal maps.

Our rendering framework is extremely robust. Since we allow artists to directly create normal, thickness and displacement maps, normal maps may not necessarily be unit vectors and thickness or displacement information may be missing. Our framework still provide an acceptable rendering in such cases. This approach is in synch with fine art practice. To obtain qualitatively acceptable effects, 2D artists do not require physically accurate "or exact" models for 3D objects and illumina-

tion processes. Instead, they want to freedom to describe final i.e. visual results. By allowing qualitatively acceptable fuzzy shapes and processes, we provide artistic flexibility to painters who may want to directly control final results without considering any constraint.

An important feature of our framework is the compatibility with existing digital image manipulation systems. Even the shader parameters, such as refraction or ambient reflectance, are provided with a set of images. The final results are created by compositing these images using illumination information obtained from the full rendering process. This provides familiarity in addition to efficiency and ability for 2D artists to directly create and manipulate artworks and allow intuitive artistic control over visual results.

The shading parameters we have computed for every pixel guarantee to provide a whole gamut of numbers between 0 and 1. Using this shading parameter c as interpolation parameter for every pixel, we interpolate two artist-provided control images, DI_0 and DI_1, to obtain final image. There is really no requirement for creating DI_0 and DI_1. For any given pixel (u, v) the color of DI_0 is the color the artist wants to see if there is no light reaches to that pixel, i.e. $c(u, v) = 0$. Similarly, for any given pixel (u, v) the color of DI_1 is the color the artist wants to see if this point is completely illuminated by the light, i.e. $c(u, v) = 1$. In other words, this process guarantees to obtain colors exactly like the artist wanted. An example of using DI_0 and DI_1 to control final diffuse shading is shown in Figure 1.4. Transparency and reflection parameters can also be provided by using just images. The Figure 1.5 demonstrate a mixed usage of transparent and diffuse materials.

This approach can be used in a wide variety of 2D applications including digital painting, artistic filtering, re-interpretation of paintings and illustrations. Artists can create artificial, but still believable, versions of the original images as well as original

(1) Original drawing (2) Shape Map: $SM(u, v)$ (3) $DI_0(u, v)$

(4) $DI_1(u, v)$ (5) shading parameter (6) Rendering

Figure 1.4: Using control images as diffuse shading parameters - (1) An artist's original drawing; (2) a mock-3D shape created from the original drawing using an illustration based software like Lumo; (3) a gray-scale image that demonstrates shading parameter for a given point light; (4) and (5) control images images provided by the artist; (6) The rendered image created by interpolating these control images using shading parameter in (3)

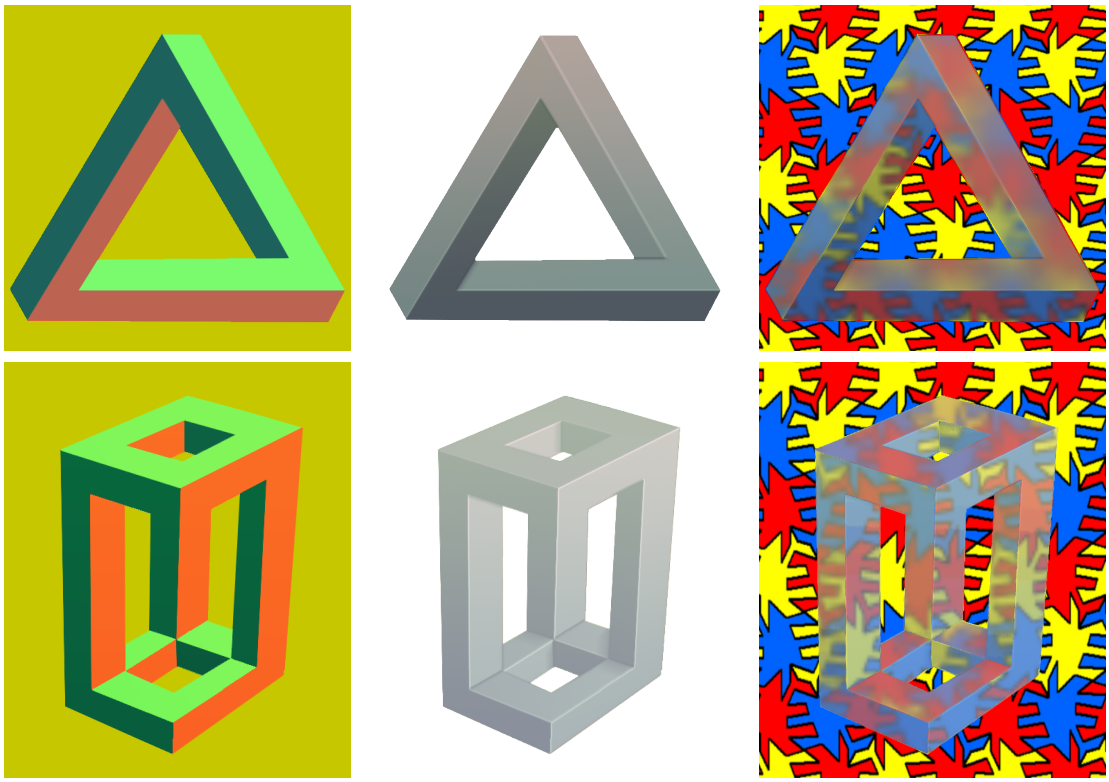
art work that can be dynamically manipulated with complete control over final results or they can reinterpret existing artworks such as the one shown in Figure 1.6 or they can create dynamically rendered images of impossible objects as shown in Figure 1.7.

$DI_0(u,v)$ $DI_1(u,v)$ Mock 3D Shape Rendering

Figure 1.5: Non-photo-realistic rendering with reflection, glossy reflection, refraction and translucence combined with Fresnel using a mock-3D shape - The control images and the normal+thickness map, called shape map, were freely hand-painted by an artist. The white regions in dark and light images are transparent and, therefore, allow refraction and reflection

$DI_0(u,v)$ $DI_1(u,v)$ Mock 3D Shape Renderings

Figure 1.6: Artistic-painted mock-3D shape - This mock-3D shape and control images are painted by an artist based on one of the Pablo Picasso's self portraits



Mock-3D Shape | Diffuse | Refraction

Figure 1.7: Reflection and refraction with impossible objects - Shape maps and foreground images are painted by an artist in a digital painting program. We use global $\alpha_G = 0.5$ to make foreground layer slightly transparent. Note that there exists no continuous height field that can produce these gradient fields

2. RELATED WORK

In this chapter, we will talk about the related work in creating a normal map and rendering and compositings that are directly used on normal map.

Even though the main contribution of this work resides in the rendering and compositing, our mock-3D representation, shape map, resembles normal maps, and most of the previous literatures do not distinguish modeling from rendering, so we provide a survey on both modeling and pure rendering.

2.1 Normal Map Modeling

One popular way of modeling is to define the normals on the strokes from user input, then diffuse the normal from these strokes into the empty area of the shape iteratively. Usually the strokes are for the edges, since edges in 2D drawing typically indicates a sharp change of normals. Johnston [15] proposed to use primitive shapes to define edges, and applied an iterative Laplacian kernel to diffuse the normal vectors from edges. But this method provides very limited controls to the user. Sun et al. [25] introduced Gradient Mesh to semi-automatically and quickly interpolate normals from edges, and Orzan et al. [20] calculate a diffusion from edges by solving the Poisson equation. Sýkora et al. [26] proposed Lazy-Brush, which can propagate scribbles to accelerate the definition of constant color regions. Finch et al. [6] build thin-plate splines which provide smoothness everywhere except at user-specified tears and creases. The underlying splines are used to interpolate normals. Wu et al. [32] proposed shape palette, where user can draw a simple 2D primitive in the 2D view and then specify its 3D orientation by drawing a corresponding primitive. This method also performs diffusion using a thin-plate spline. Shao et al. [24] uses an explicit mathematical formulation of the relationships between cross-section curves

and the geometry they aim to convey. The specified cross-section point is used as an extra control point to control the normals. Vergne et al. [29] introduces surface flow from smooth differential analysis, which can be used to measure smooth variations of luminance. Therefore, the author also propose to drawing the shadows and other shading effects. Sýkora et al. [27] developed a user-assisted method to convert normal maps into Bass-Reliefs that can provide correct shadows in a commercial renderer, but this approach will fail if the normal maps does not correspond to shapes that can have an explicitly meaningful 3D geometry.

2.2 Rendering

The rendering technique we developed follows the branch of non-photorealistic rendering(NPR), where rendering mainly works on normal maps. NPR shading models are often simply functions of the surface normal and light direction that result in effects such as Gooch shading [8], cartographic hill shading[13], or other artist-specified effects. A more complex model includes curvature-based shading [16] and "exaggerated shading" [18]. However, all these techniques are only available in 3D with a normal and corresponding position information. However, shaded appearance may be designed through a painting interface, even though they may not be photorealist, such as tweakable light and shade[14]. In tweakable light and shade[14], Anjyo et al. proposed to control light and shade inside a shape by dragging high-lighted area, and an underlying normal map still needs to be estimated. The rendering technique we developed is most related to the work of [28]. In this work, Toler et al. created non-photorealistic illustrations from a type of data lying between simple 2D images and full 3D models, named RGBN image, which contains both color and a surface normal information. However some limitation exists such as shadows, which are only considered at discontinuities of normal. The proposed reflection/refraction

methods shares similarity with the work of Ritschel et al. [23], as we both conduct refletion/refraction in a non-physical way. Ritschel et al. introduce a sketch-based interface for artist to create reflection effects easily. Later on, they [22] improved the interface that can also edit shadows, caustics, and indirect illumination. However, their results highly depend on the scene, the camera motion, and the performed edit. And all of these have to be tuned carefully by the user.

3. MOCK-3D SHAPE REPRESENTATIONS

In this work, our goal is to obtain qualitative global illumination effects with mock 3D shapes, which are not really 3D but appear 3D. In this work, we introduce a fuzzy and view dependent shape representation that is suitable for global illumination and provides the representational powers of both bas-reliefs and normal maps. Our approach can be better explained with a closer look to bas-reliefs and normal maps.

3.1 Bas-Reliefs as Mock-3D Representations

Bas-reliefs are sculptures that can be viewed from many angles with no perspective distortion as if they are just images. In other words, perspective transformation is embedded in bas-relief sculptures [30]. Figure 3.1 demonstrates bas-relief creation process from any given 3D shape. Let a shape be defined implicitly by a general function $G(x, y, z)$, then, the volumetric reforestation of this shape is

$$S_0 = \{(x, y, z) \in \Re^3 | G(x, y, z) \leq 0\}.$$

Bas-relief process turns these shapes into height fields that embed perspective projection into the shape as shown in Figure 3.1.(b). This new shape, which is significantly simpler than the original one can be represented simply as

$$S_1 = \{(x, y) \in \mathcal{D} \text{ and } z \in \Re | z - F(x, y) \leq 0\}$$

where $\mathcal{D} \in \Re^2$ is just the domain where the height field function F is defined. A bass-relief can be further scaled to obtain a compact sculpture that can be hanged on a wall as shown in Figure 3.1.(c). As a result, bas-reliefs can be represented

implicitly as

$$S = \{(x, y) \in \mathcal{D} \text{ and } z \in \Re | z - sF(x, y) \leq 0\}$$

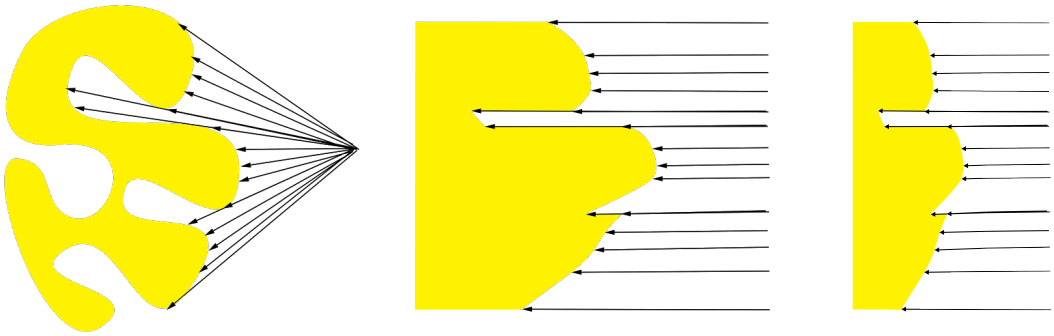
where s is any positive real number smaller than 1.

Remark 1: As shown in the example, even when original function G is continuous, the corresponding bass-relief height field function F may not be continuous.

Remark 2: Surface normals in bass-relief are qualitatively correct only. Assuming that F is continuous, the normal to the surface can be computed from gradient of $z - sF(x, y)$ which is computed as

$$\nabla(z - sF(x, y)) = (-s\frac{\partial F}{\partial x}, -s\frac{\partial F}{\partial y}, 1)$$

where the unit normal directly depends on the value of s. This is not a problem in bass-reliefs probably because scaling operation does not change the positions of extreme points such as maximum, minimum or saddle.



(a) General Shape (b) Bas-Relief (c) Bas-Relief - Scaled

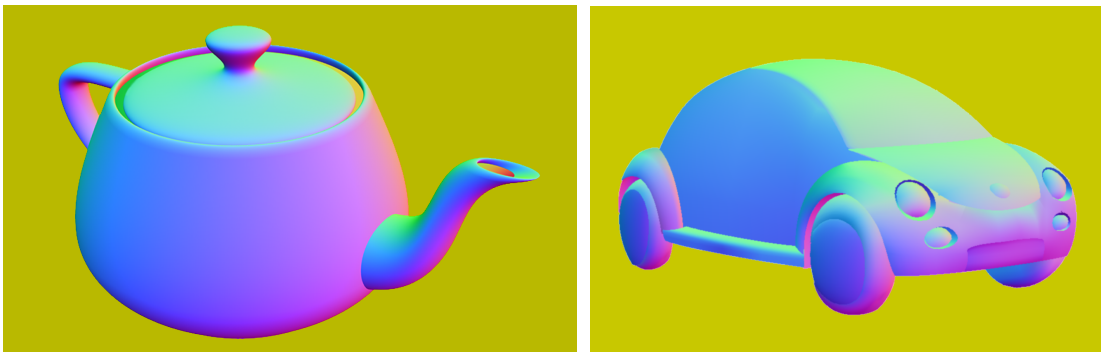
Figure 3.1: An exaggerated 2D presentation Bas-Relief creation from a real object

3.2 Normal Maps as Mock-3D Representations

Cohen et al. introduced normal maps to be a simpler alternative to bump maps [3]. Let $\mathcal{D} \in \Re^2$ denote the domain normal map to be defined and let $(x, y) \in \mathcal{D}$ denote two coordinates of the map. Let $(n_0(x, y), n_1(x, y), n_2(x, y))$ denote the 3D vector field $n_0 : \mathcal{D} \to [-1, 1],$, $n_1 : \mathcal{D} \to [-1, 1]$, and $n_2 : M \to [0, 1]$. One significant advantage of having only three variables for normal vectors is that we can readily convert them into Low Dynamic Range (LDR) images and save them using any common image format which can easily passed to GPU. Let an LDR image on M is denoted by $c(x, y) = (r(x, y), g(x, y), b(x, y))$ where $c : \mathcal{D} \to [0, 1]^3$. The conversion from (n_0, n_1, n_2) to (r, g, b) is given as $(r = 0.5(n_0 + 1), g = 0.5(n_1 + 1), b = n_2)$.

Although, normal maps are mainly used as texture maps to include details to polygonal meshes, they can directly be used as a shape representation as shown by Johnston [15]. He developed a sketch based system, called Lumo, to model normal maps as mock-3D shapes by diffusing 2D normals in a line drawing .Recently, Shado et al. [24] developed CrossShade, another sketch based interface to design complicated shapes as normal maps. These works suggest that normal maps are useful for illustrators to create mock-3D shapes. Figure 3.2 shows normal maps generated by Lumo and Crossshade.

An advantage to thinking of normal maps as 3-color images is that painters can paint mock-3D shapes using a 2D painting software. The biggest advantage of normal map formulation is that the painters do not need to think that they are working on a gradient domain to paint these images. They can imagine an object lit by 2-point lighting illuminated from left with a directional (parallel) red light and from above by a directional green light. Ignoring shadows, they can paint the image based on how much red and green light they want to see in every pixel. For instance, a pixel



Lumo Model Crossshade Model

Figure 3.2: Normal maps generated using an sketch based modeling program

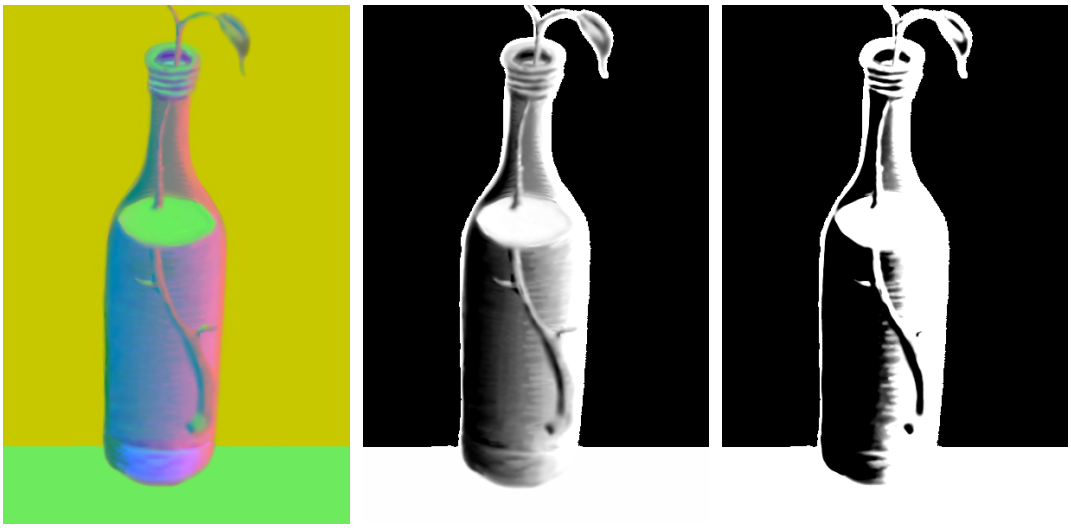
color red=0.75 and green=0.3 means, the artist wants 75% of the light from the left and 30% of the light from the top can illuminate that particular pixel.

Another simplicity for painters, they do not need to provide n_2 values explicitly. Since in normal maps the vectors are encoded as unit vectors and n_2 component is always positive, (n_0, n_1) components of a normal vector is sufficient to compute n_2 component as $n_2 = \sqrt{1 - n_0^2 - n_1^2}$. This property is not only useful for painting, it is also useful for automatic normal map creation. For instance, in Lumo only (n_0, n_1) components of the normal vectors are computed, n_2 component is ignored and computed to maintain unit length [15].

One issue that requires attention with painted 2D (n_0, n_1) vector fields is that $n_0^2 + n_1^2$ can be larger than one. In this case, the painted vector field does not formally correspond to a normal vector field since $\sqrt{1 - n_0^2 - n_1^2}$ can be a complex number. Fortunately, even when the painted 2D vector field is not reliable, there exists several strategies to obtain a normal map. One strategy is to uniformly scale 2D vector field by a parameter $s \in [0, 1]$. For s values smaller than $1/\sqrt{2}$, the unit 3D normal vector always exist $n(u, v) = (sn_0, sn_1, \sqrt{1 - s^2 n_0^2 - s^2 n_1^2})$. This scaling operation is related

to scaling bas-reliefs.

Providing direct control to painters are essential for the creation of unusual images since the artists can deliberately introduce imperfections that lead to expressive and artistic effects. We have created some painted normal maps to demonstrate this idea. For instance, the imperfections introduced by the artist in the wine bottle shape map shown in Figure 3.3 resulted in crosshatching from Fresnel computations when background was completely black and the reflected environment was white. The same imperfections also provided crosshatching in cartoon shading under diffuse illumination as shown in Figure 3.3.



2D vector field Normal Cartoon

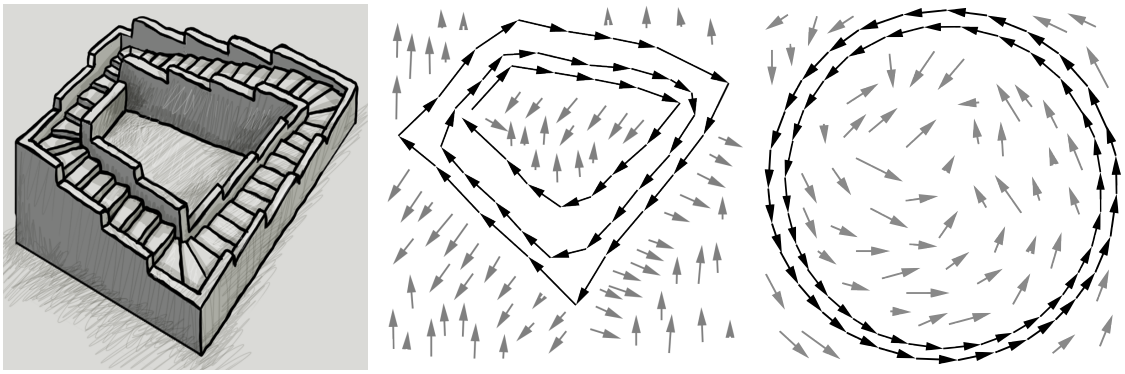
Figure 3.3: The effect of the cartoon shading - In this case, control images are simply black and white. Note that in this case crosshatching effect comes directly from hand-drawn vector field. We expect direct involvement of painters into the process of creating normal maps can result in innovative solutions

Recent experimental results also support that painting and illustration can be

natural way to create normal maps. In particular, Cole et al. [4] demonstrated that people can correctly estimate normal vectors from line drawings and Shado et al. [24] further demonstrated that the normal maps generated by CrossShade can be depicted as if they are 3D shapes. This suggests that normal maps could be easily created and manipulated by artists and play an important role for designing expressive and artistic images.

3.3 Comparison of Bas-Reliefs and Normal Maps

One problem with bas-reliefs for 2D artists is that their construction is still a sculpting process. This may not be suitable for illustrators and painters who are not interested in sculpting shapes. Normal maps can be more suitable for 2D artists since they can be created by sketching, painting or even photographing.



(a) (b) (c)

Figure 3.4: Examples of impossible shapes

A useful way is to view a normal map as a 2D vector field $N = (N_0, N_1)$ where $N_0 = n_0/n_2$ and $N_1 = n_1/n_2$. One additional advantage of this conversion, both N_0 and N_1 can be any real number, that can give flexibility to hand-painting. If

(N_0, N_1) is a conservative field, or, in other words, a gradient field, then there exists a bas-relief function F such that

$$\partial F(x,y)/\partial x = N_0(x,y)$$

$$\partial F(x,y)/\partial y = N_1(x,y).$$

In this case, the function F can easily be computed with line integral. This formulation demonstrates an important problem of normal maps: they cannot represent discontinuities. The gradient can only exists if $F(x,y)$ is a continuous function. Therefore, normal maps can only represent continuous bas-reliefs. To solve this problem, Sýkora et al. developed an interface to introduce discontinuities to obtain bas-reliefs that can provide correct shadows [27]. Their addition makes normal maps a more powerful representation, however since the final geometry is still a bas-relief, which cannot represent impossible shapes.

The real power of normal maps comes from representation of impossible shapes. We first need to formally define impossible shapes.

Let N be a 2D vector field that is obtained from a normal map. Then normal map N is called an impossible shape if it is not a gradient field; i.e. $curl(N) \neq 0$ for some $(x,y) \in \mathcal{D}$. Here curl is defined as $\partial N_0(x,y)/\partial y - \partial N_1(x,y)/\partial x$.

This definition formally classify a normal map as an impossible shape when it does not does not correspond any geometry. Such vector fields can be created by artists by mistake, but more importantly they can actually be related to well-known impossible shapes. For instance, ascending stairs shown in Figure 3.4(a) can be considered a closed-loop of normal vectors that continuously go up. This can be represented by a 2D vector field that includes cycles as shown black vectors in Figure 3.4(b) or (c).

The advantage of the normal maps is that they can still allow to compute shading since they provide the essential shader information: the surface normal in any given point. On the other hand, since these normal maps do not have geometry, they cannot cast shadow.

It is still possible to find a geometric approximation to an impossible shape N into by computing a height field $F(x, y)$ that can minimize the error

$$E = \int\int (\frac{\partial F}{\partial x} - N_0(x, y))^2 + \frac{\partial F}{\partial y} - N_1(x, y)^2) dudv.$$

This is a well-known problem in computer graphics. To obtain such an height field that minimize errors, we have to solve Poisson Equation [5]. One advantage of this approach is that it is robust. However, the robustness is really the problem here. The operation turns an impossible shape into a "possible shape" that is represented by a continuous height field.

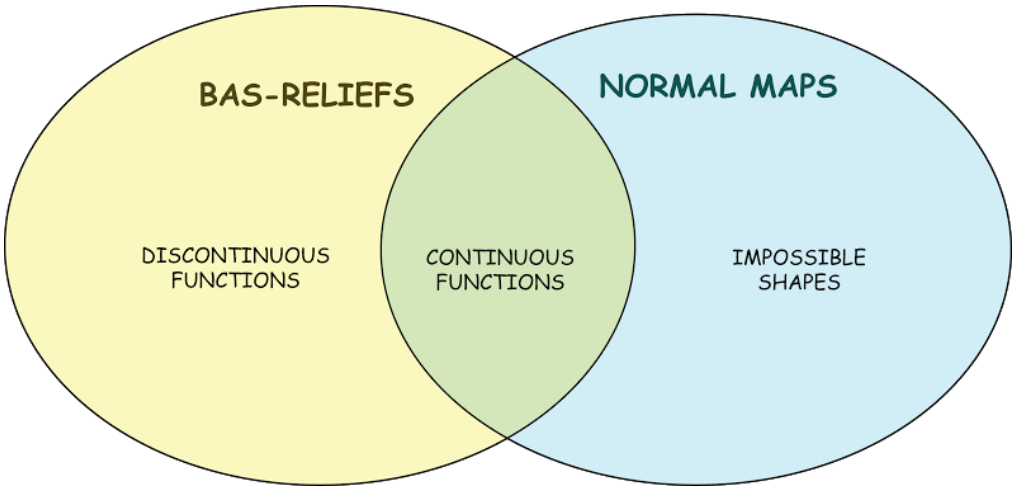


Figure 3.5: The comparative power of representations

Figure 3.5 provides a visual comparison of the representational power of bas-reliefs and normal maps. Note that bas-reliefs and normal maps are related to each other only when bas-reliefs are continuous functions and normal maps corresponds to 2D gradient fields. One additional issue for both representations is that they are one-sided, i.e. they do not have back sides. Therefore, neither of them can really be used in cases where back sides cast shadow. In this paper, we introduce a new representation that can provide the power of bas-reliefs and normal maps along with two-sided shapes.

3.4 Qualitative Similarity in Rendering

We observe that bas-reliefs and normal maps work since they provide qualitatively acceptable rendering results. Although "rendering" results are physical in bas-reliefs and simulated in normal maps, conceptually they provide qualitatively similar type of visual results.

To base our discussion on qualitative rendering we use the definition provided by Forbus et al [7]. Let $Q_x : \mathfrak{D} \rightarrow \mathbb{R}$ and $Q_y : \mathfrak{D} \rightarrow \mathbb{R}$ denote functions $x = Q_x(v)$ and $y = Q_y(v)$ where $v \in \mathfrak{D}$ denote any given domain and $x, y \in \mathbb{R}$ denote real numbers. They, then, say Q_x and Q_y are qualitatively proportional to each other if there exists a monotonically increasing mapping $y = f(x)$: For all $p_1, p_2 \in \mathfrak{D}$ if $x_2 = Q_x(p_2) > x_1 = Q_x(p_1)$, $f(x_2) = y_2 = Q_y(p_2) \geq f(x_1) = y_1 = Q_y(p_1)$, then we can call Q_x and Q_y qualitatively proportional to each other. One advantage of this formulation, there is no need to explicitly derive the monotonically increasing function f. To demonstrate Q_x and Q_y are qualitatively proportional, we can simply demonstrate the following inequality is correct for all p_1 and p_2:

$$\frac{Q_x(p_2) - Q_x(p_1)}{Q_y(p_2) - Q_y(p_1)} \geq 0$$

The symbol $\propto_+$ is used to relate two qualitatively proportional entities to each other as

$$Q_y \propto_+ Q_x.$$

This formalization is especially useful to evaluate qualitative similarities in rendering and post processing. For instance, we can show that a γ corrected image is qualitatively similar to the original image. We can also demonstrate a given tonmapping algorithm guarantees creation of LDR images that are qualitatively similar to the original HDR images. This formalization is also in sync with human visual perception. As shown by many researchers such as Adelson [17] and artists such as Albers [1], the color perception is relative. In other words, it makes sense to check positive correlation between relative differences.

In the case of shaders, we can consider the domain $\mathfrak{D}$ consists of surface positions, surface normals, light directions. The shader parameters we use in computer graphics are like Q_y or Q_x that provide shader parameters for any given set of surface positions, surface normals and light directions. Now, assume that there exists a shader parameter Q_x that correspond "so-called physical reality", we can then call shader Q_y is qualitatively acceptable if and only if $Q_y \propto_+ Q_x$.

A most commonly used shader parameter is $\cos\theta$. Here, we will demonstrate that the scaling the first two components of surface normal creates a new $\cos\theta$ that is qualitatively similar to original in the bas reliefs and normal maps. The shading parameter $\cos\theta$ turns an operation over normal vectors into a real number for any given light direction. If the effect of an operation (regardless of light position) on color of two neighboring points is always qualitative proportional to original color of two neighboring points, then we can say that the operation is qualitatively acceptable.

Without loss of generality, we will evaluate the comparison in 2D. Let $n = (a, b)$

and $v = (x, \sqrt{1-x^2})$ denote denote light direction and normal vector respectively, where $x \in [-1, 1]$ and $a^2 + b^2 = 1$. In this case, $Q_x = n \cdot v = ax + b\sqrt{1-x^2}$. Scaling bas-relief changes the surface normal as $v = (sx, \sqrt{1-s^2x^2})$, so $Q_y = n \cdot v = asx + b\sqrt{1-s^2x^2}$. Note that the Taylor expansion of $\sqrt{1-x^2}$ is,

$$\sqrt{1-x^2} = 1 - \frac{1}{2}x^2 - \frac{1}{8}x^4...$$

Since x smaller that 1, we can ignore higher order terms such as x^4 for the sake of simplicity, then using the fact that

$$\begin{aligned} \frac{Q_y(x_2) - Q_y(x_1)}{Q_x(x_2) - Q_x(x_1)} &= \frac{as(x_2 - x_1) - 0.5bs^2(x_2^2 - x_1^2)}{a(x_2 - x_1) - 0.5b(x_2^2 - x_1^2)} \\ &= s\frac{a - 0.5bs(x_2 + x_1)}{a - 0.5b(x_2 + x_1)} \\ &= s\frac{a - bsy}{a - by} \end{aligned}$$

where $y = 0.5(x_2 + x_1)$ and $-1 \leq y \leq 1$. Note that this ratio is positive if $a > by$ or if $a < bsy$. This suggests that if $a > b$, it is very likely that bas-relief will look qualitatively similar to original shape. On the other hand, when the light is becoming more and more perpendicular to the bas-relief, there may exist some regions on the shape that can give away the bas-relief shape is actually flattened. If s is not very small, the possible light directions that can result in such qualitatively un-similar results in some regions are relatively small. Most likely, therefore, bas-reliefs are acceptable Mock-3D representations of complicated 3D shapes.

This particular approach can only evaluate local shading parameters such $\cos\theta$ term. For the global shading parameters such as shadows, the domain $\mathfrak{D}$ is in higher dimension, which include additional variables such as neighboring point positions

and other shapes in the scene and thickness of the shape. Although the problem is more complicated, it is still possible to make an evaluation. For instance, in flattened bas-relief local shadows will not be visible unless light direction is almost tangential to the bas-relief surface. Bas-reliefs cannot be included in a scene as a replacement of a 3D shape, since they do not have back side. Therefore, we need a new Mock-3D representation that can provide qualitatively acceptable global illumination effects. In the next section, we introduce shape maps, an extended version of normal maps, as an alternative Mock-3D representation.

4. SHAPE MAPS AS MOCK-3D SHAPE MODELS

In this chapter, we will first explain the concept of view dependent 2-sided mock-3D shapes. Then, we will introduce shape maps as a new Mock-3D shapes. Last, we will talk about different ways of creating shape maps.

4.1 View Dependent 2-Sided Mock-3D Shapes

In this work, we present view dependent and 2-sided mock-3D shapes as an alternative to mock-3D shapes to represent all types of discontinuities and all types of impossible geometries as two-sided shapes.

Let $F_0(x, y, \theta)$ and $F_1(x, y, \theta)$ denote two functions from $\mathcal{D} \times [0, 2\pi]$ to $\Re$ with $(x, y) \in \mathcal{D}$ and $\theta \in [0, 2\pi]$. Then, a view dependent and 2-sided mock 3-D shape is defined using the following inequality as

$$S = F_0(x, y, \theta) \geq z \geq F_1(x, y, \theta)$$

where $z \in \Re$.

In this definition, two-sidedness of the shape directly comes from usage of two functions F_0 and F_1. The second and more important difference with bas-reliefs is that the shape is also a function of angle θ. In other words, this definition allows to have a view dependent geometry since shape depends of the angle θ. View dependency is a desired feature in artistic applications [21]. It also provides a fuzzy definition for the shapes that can be particularly be useful for representing impossible shapes. The only problem with this general structure is that it is hard for 2D artists to create these two essentially 3D functions. In this work, we presents shape maps, which are a relatively simple extension of normal maps. We demonstrate that

shape maps can be used to construct these two functions.

4.2 Shape Maps as Mock-3D Representations

Shape maps consists of three maps: normal maps, displacement maps and thickness maps. We defined normal maps and discussed how to create normal maps earlier in Section 3.2. Displacement and thickness maps are also defined over the same domain of the normal maps, $p = (x, y) \in \mathcal{D} \subset [0, 1]^2$. Let $Z(p)$ and $T(p)$ denote displacement and thickness maps respectively and $Z : \mathcal{D} \to [-1, 1]$, and $T : M \to [0, 1]$. Note that both maps can be represented as images.

Using these maps, the functions F_0 and F_1 is computed as a summation of the two line integrals of 2D gradient fields that are obtained from normal maps and displacement maps as

$$F_0(p_1, \theta) = s_0 G_0(p_1, \theta) + s_1 G_1(p_1, \theta)$$

$$F_1(p_1, \theta) = F_0(p_1, \theta) - s_2 T(p)$$

where

$$G_0(p_1, \theta) = \int_{p_0}^{p_1} N_0(p(t)) \cos\theta dt + N_1(p(t)) \sin\theta dt$$

$$G_1(p_1, \theta) = \int_{p_0}^{p_1} \frac{1}{n} \lfloor n \frac{\delta Z(p(t))}{\delta t} + 0.5 \rfloor dt$$

with $p_0 = (x_0, y_0)$ is the starting point of the integral, which is computed as the intersection of the ray starting from $p_1 = (x_1, y_1)$ in the direction of $(-\cos\theta, -\sin\theta t)$ with the boundary of the domain $\mathcal{D}$ and $p(t) = p_0 + (\cos\theta, \sin\theta)t$ and $\lfloor x \rfloor$ is the floor function that return largest integer smaller than x, n is an integer, quantization term and $s_0, s_1, s_2 \in [0, 1]$ are scale parameters.

If the 2D vector field (N_0, N_1) is conservative, then $G_0(p_1, \theta)$ is independent of

θ. In other words, this integral provides all continuous function bas-reliefs when 2D vector field is conservative. If 2D vector field is not conservative, then the integral is dependent on θ, but it is still uniquely defined. The resulting function is continuous in the direction of θ and may not necessarily be continuous in other directions. Therefore, it turns a normal map defining impossible shape into a fuzzy geometry.

We use displacement map $Z(x_1, y_1)$ to make F_0 and F_1 discontinuous functions in the direction of θ. The $\frac{1}{n}\lfloor\frac{n\partial Z(p(t))}{\partial t} + 0.5\rfloor$ term quantizes the derivative to the nearest i/n where i is an integer. The resulting piecewise constant functions approximates the original derivative functions with a precision provided by $1/n$. For small values of n, the new derivative functions may not necessarily be a gradient field anymore and provide impossible discontinuities.

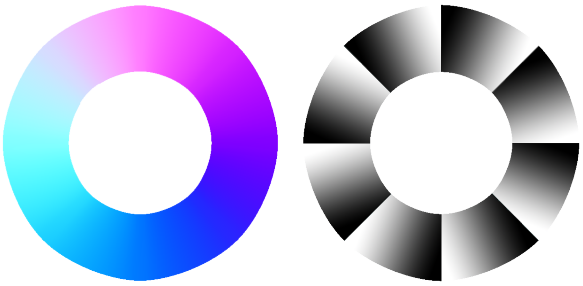
Let $\frac{1}{n}\lfloor\frac{n\partial Z(p(t))}{\partial t} + 0.5\rfloor$ be a 2D vector field that is obtained from a displacement map. Then, it is again an impossible shape if it is not a conservative field; i.e. there exist a closed path $p(t) \in \mathcal{D}$ such that

$$\oint \frac{1}{n}\lfloor\frac{n\partial Z(p(t))}{\partial t} + 0.5\rfloor dt \neq 0.$$

This is particularly useful to for creating shadows impossible shapes, for instance using Z function shown in Figure 4.1, we can turn impossible ramps into impossible stairs.

These three maps provides three different representational powers:

- 2D gradient field obtained from normal map provides both continuous functions and impossible shapes.
- Thickness map provides back side of the shape.
- Displacement map provides discontinuities and local layering information.



(a) Normal Map (b) Displacement Map

Figure 4.1: Ascending stairs viewed from above - Note that normal map does not carry any information since we can only see the top of the stairs, whose normals are simply $(0, 0, 1)$. Quantization with $n = 1$ removes all details in displacement map and in a circular path, the integral does not become zero

The shapes maps can be considered "extended billboards". They can easily be used to create scenes that allow for global illumination effects. And we can also use shape maps in a multi-layer way to introduce discontinuities. Details about this will be introduced in Section 5.3.

4.3 Creation of Shape Maps

In this section, we discuss the creation of shape maps. Since shape maps consists of only three maps: normal, displacement and thickness, they can be encoded as images similar to normal maps. As it can be seen in shape map examples, the shape map images are more colorful than normal maps since we use blue color for thickness information. This configuration is useful since when the thickness information is not available, z component of normal maps can provide an acceptable thickness map. As discussed in Section 3.2, the main advantage of using images as maps is that artists can create these maps directly using a painting software such as Gimp or Photoshop for details).

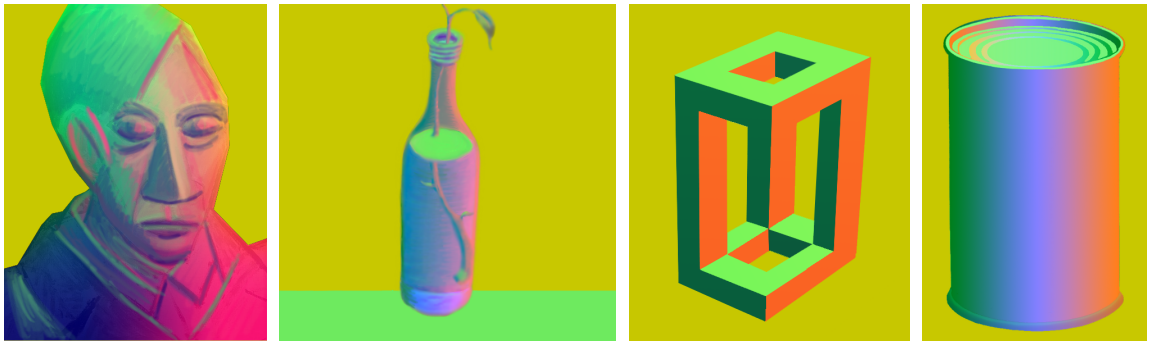


Figure 4.2: Shape maps generated by a painter using a paint program

4.3.1 Shape Map Image Painting

As discussed earlier, to paint a shape map, artists can imagine an object that is illuminated with parallel red light from their left side and with parallel green light from top. By ignoring shadow, they paint the image based on how much red and green light they want to see in every pixel. For instance, a pixel color red=0.95 and green=0.75 means, the artist want 95% of the light from the left and 75% of the light from the top to illuminate that particular pixel. Note that unlike normal maps, summation of their squares do not have to be smaller than 1 as in this example. This is not a problem, since like bas-reliefs and normal vectors, we can always scale values of n_0 and n_1 to sn_0 and sn_1. This operation, in effect, changes $n_2 = \sqrt{1 - s^2n_0^2 - s^2n_1^2}$. In our implementation, we use the two vectors from shape to rebuild normal vector as $(sn_0, sn_1, \sqrt{1 - s^2n_0^2 - s^2n_1^2})$, where we call $s \in (0, 1]$ a Shape Map quality parameter. This value indicates how reliable current (n_0, n_1) is to real normals. And we can easily see that $s \leq 1/\sqrt{2}$ can guarantee $1 - s^2n_0^2 - s^2n_1^2 < 0$. So we would like to grant user the control to set s value, if the normal of the painted shape map is very accurate, e.g. it comes from the normal of a real 3D object, user can set the s value high, otherwise, user can choose a s value that can provide best

visual effects.

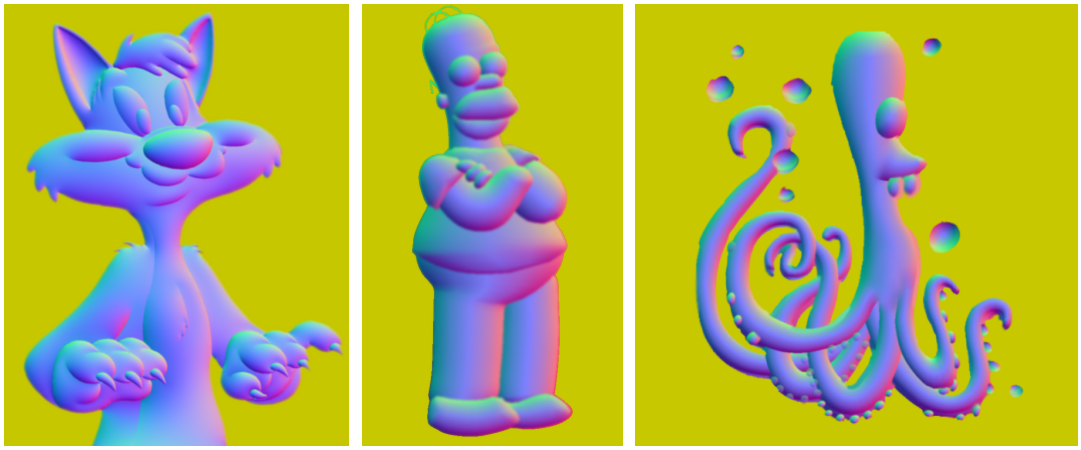
Red and green lights vectors $(1, 0)$ and $(0, 1)$ are linearly independent from each other. Therefore, any 2D light can be given a linear combination of the two as $(x_L, y_L) = x_L(1, 0) + y_L(0, 1)$. Therefore, to compute illumination coming from an arbitrary parallel light, all we need to do is to compute the contribution from two linearly independent components.

Remark 1: Painting thickness values d are easier. As discussed earlier, d values have to be non-zero for the object and zero for everywhere else. Moreover, d values has to be small close to object boundaries(if considering perspective transformation) and thin regions. This is sufficient to obtain visually correct looking refractions. For the rest of the object, the d values can simply be any reasonable positive real number smaller than 1. The Figure 4.2 shows shape maps painted by artists by imagining a red light from the left side and a green light from the top.

The bottle image shows how thickness values should be painted. Since this bottle is half-filled with a liquid, in the places where the bottle is not filled with the liquid, the d value has to be very small to indicate a thin glass, although z component of the unit vector is not small.

4.3.2 Illustrating Shape Map

Another option, particularly for illustrators, is to model 2D vector fields directly with sketch based interface. To create the 2D vector fields, we have implemented a simple modeling system based on boundary gradient interpolation, a concept suggested by Johnston [15]. Since modeling is not the focus of this paper, we do not include details here. Our rendering approach does not depend on our models. Normal maps created by Lumo [15] or CrossShade [24] can be used as shape maps as shown in Figure 4.3(b). Although neither of them provides a separate thickness in-



Lumo Model Two models from Our Software

Figure 4.3: Shape maps generated using an sketch based modeling program

formation, normal maps' blue channel can be used as thickness if there is no other option. To provide thickness in an illustration environment, a sketch based modeling software is created in our group. Several of our examples are created using that software. Since modeling is not the focus of this thesis we do not give detail here.

Figure 4.4: Shape maps generated from photographs

4.3.3 Turning 3D Shapes into Shape Map Images

Although we prefer artists to create shape maps, it is also possible to convert real objects into shape maps. The procedure to obtain a 2D vector field is a straightforward rendering process. The x and y components of the 3D-normal vector of the visible point is simply converted to red and green colors of image. It is even possible to use z component of the unit normal vector as the d value. This choice will give small d values in object boundaries. However, the object may be thin even if the z value is not small. We do not provide any example for this case since it is straightforward 3D graphics application.

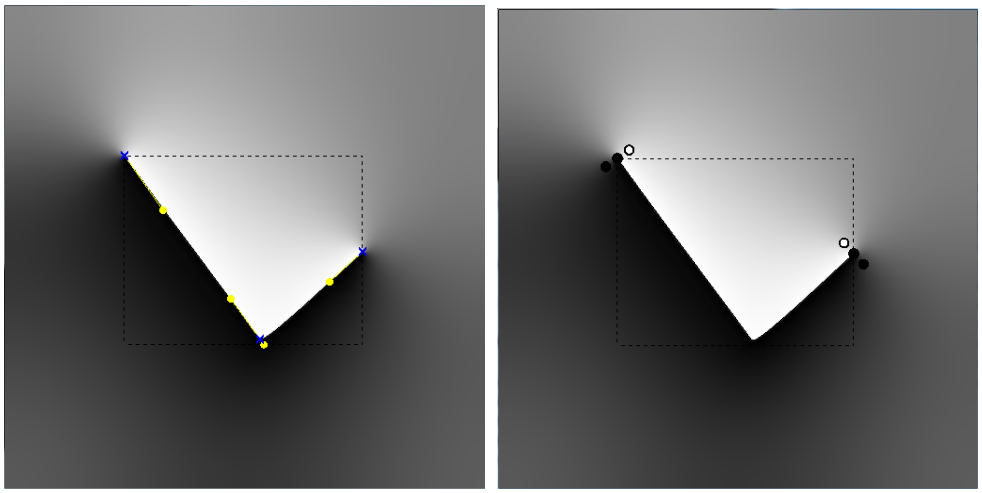
Figure 4.5: More shape maps generated from photographs

4.3.4 Shape Maps from Photographs

Another method to obtain shape maps is by photographing real objects using red and green lights, which can be a simple alternative to environment matting [33]. The Figure 4.4 and Figure 4.5, show such examples. In these example, we have made only minimal changes in original image: (1) we removed and replaced backgrounds with yellow color; and (2) we added a non-zero blue value for object regions. Note that yellow background corresponds to zero thickness and therefore does not refract light. Although constant thickness is not correct, the resulting refractions appears reasonably convincing. The artists, of course, can further manipulate these photographs to obtain desired effects.

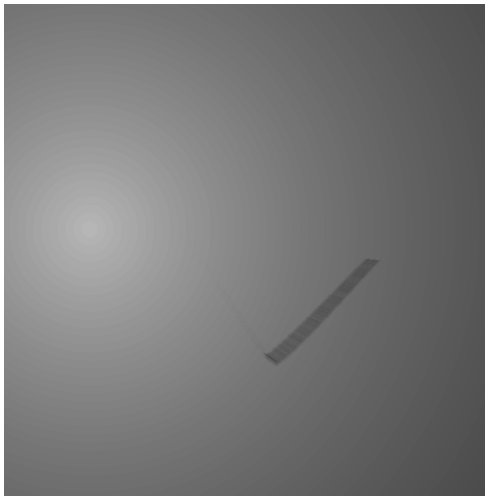
4.3.5 Creation of Displacement Map

As we discussed, displacement maps are optional. They are used only to introduce discontinuities when necessary. The creation of displacement map is in general the same in all cases, and user can simply use diffusion curve to create a displacement map. Each discontinuity lines can be drawn as one diffusion curve[20], as shown in Figure 4.6(a). The colors at two sides of the diffusion curve represent the higher and lower height that would create a discontinuity, therefore, user can set a high value at the higher side, and low value at the lower side, as shown in Figure 4.6 (b). Figure 4.6 (c) and (d) shows the effects of displacement map. The normal map here has (0,0,1) normals everywhere and the thickness map here has a constant value everywhere. So if the light is on the higher side, shadow will be created along the direction of pixel position to light position, otherwise, no shadow will be cast.

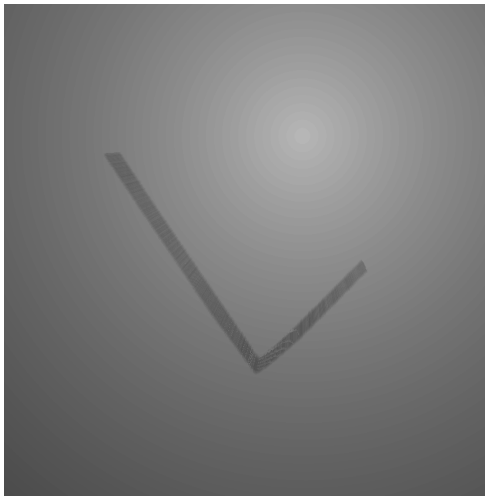


(a)

(b)



(c)



(d)

Figure 4.6: Displacement map created using diffusion curve - (a) and (b) show how to create displacement from diffusion curve. (c) and (d) show the results, with the yellow spot as lighting position

5. MOCK-3D SCENES

In this chapter, we will introduce two different types of Mock-3D scenes, the simple layered Mock-3D scene and Billboard Mock-3D scene. Then, we will talk about the basic of how to build the shader of the scene.

5.1 Mock-3D Scene Types

Mock 3D scenes consist of shapes maps that are projected on planes, control images which are associated with each shape map, one background image and one environmental image. We consider only two types of scenes: (1) Simple layered and (2) Billboard based.

1. **Simple layered Mock-3D scenes:** These scenes consist of shape maps that are projected on planes perpendicular to $(0, 0, 1)$. Simple layered scenes are similar to layers in 2D painting software such as photoshop or gimp. These scenes are particularly useful for 2D artists since they are very similar to existing 2D scene descriptions that are familiar to most 2D artists. More importantly, these scenes significantly extend the capabilities provided by 2D painting software. Using these scenes 2D artists can obtain re-illumination with local shadows, compositing with reflection and refraction.

2. **Billboard based Mock-3D scenes:** These are shape maps that are projected on random planes. If one wants to obtain more complicated effects such as global shadows or reflections on a water surface, there is a need for this type of the scenes, which allow projections on the planes that can be rotated and translated in space. Although the visual experience of the user are related to simple layered scenes, we can still obtain local layering and planes that can

intersect with each other. As we will discuss later, this is important to obtain global shadows and more interesting reflections.

5.2 Simple-layered Mock-3D Scenes

Simple layered Mock-3D scene consists of one background and one environment image and a set of shape maps projected on planes that are perpendicular to $(0, 0, 1)$. The z positions of planes defines layer orders. Eye is also given by its z position. This type of the simple layered scenes can be further simplified into a single layered shape map. Anything behind the eye can be turned into one single environment image. Anything behind the first shape map after eye can be turned into a single background image. We then, view the problem as a single shape map that reflects the environment image and refracts background image. Reflections and refractions are combined with a user defined Fresnel function.

5.3 Billboard-based Mock-3D Scenes

The simple layered Mock-3D scenes cannot provide more general Mock-3D shapes such as primarily horizontal ones. These horizontal shapes such as ground plane, table top or water surface, are important to obtain realistic looking rendering results: (1) Other objects can cast shadows on these horizontal shapes and (2) other objects are reflected and refracted from these horizontal shapes.

Billboard-based Mock 3D scenes consist of shape maps projected on planar rectangles. We assume the whole scene is inside a unit cube. The users can move the center of planar rectangle in 3D and change the normal vectors of it. However, parallel projection of this rectangle never changes. This is not an issue since the rectangle is fully inside the cube. However, once it reaches the boundary, we cannot further rotate or move rectangle in 3D. For example, users cannot get $(0, 1, 0)$ as a normal vector unless make the height of the projected rectangle 0. In other words, the shape

of projected rectangle constraints possible positions and normals, however it is still possible to obtain almost horizontal shapes. One advantage of this approach is users can feel as if they are working completely on 2D, we can significantly extend global illumination effects that can be obtained with shape maps.

5.4 Shader for Mock-3D Scene

We introduce a simple description of shader parameters to obtain predictable visual results. Our approach is based on control images that are provided by artists. The final image is simply computed as a weighted average of these control images using a Bezier formulation as

$$I(u,v) = \sum_{i=0}^{N} \sum_{j=0}^{M} DI_{i,j}(u,v) B_{i,j}(c_j(u,v)) \tag{5.1}$$

where $B_{i,j}(c_j(u,v))$ denotes tensor product Bezier basis functions, $DI_{i,j}(u,v)$ denotes Bezier control images, $0 \leq c_j(u,v) \leq 1$ is shading parameter j where $j = 0, ..., M$ computed from shape map, details of this computation will be in chapter 6. The only condition for shading parameters they have to be between 0 and 1. They can represent diffuse reflection and shadow parameters for each light. In addition, they can be silhouette and ambient occlusion parameters. In this formulation, it is also possible to use quantized shading parameters to obtain controlled cartoon shading.s

One advantage of this approach is that the color of the lights are embedded in the control images and directly controlled by the painter. Because of convex hull property of Bezier curves, the equation guarantees that the result is also an image and it stays in the convex hull of control images. The advantage of Bezier basis functions over others such as B-spline basis functions is that Bezier formulation guarantees that the function interpolates $DI_{0,j}(u,v)$ and $DI_{N-1,j}(u,v)$ control images. In other

words, if our shading computation can guarantee to obtain 0 and 1 for a set of shader parameters, the painter-defined colors can always be obtained in the final image. As a conclusion, the obvious advantage of this formulation is that it can help artists to plan exactly what kind of results they expect to see.

In practice, it is hard to create such a large number of images consistently when both M and N are large. In practice, we noticed that only one light is sufficient for basic control of visual results. Therefore, without loss of generality, all our examples in this work uses only two control images and one shading parameter that corresponds to one light. The effects produced by additional lights can simply be included in the production of two control images. In this case, the general Bezier curve equation simplifies into

$$I(u,v) = DI_0(u,v)(1 - c(u,v)) + DI_1(u,v)c(u,v) \tag{5.2}$$

where $DI_0(u,v)$ and $DI_1(u,v)$ denote two images that are provided by the artist. The first image defines the color when the given point (u,v) is not illuminated. The second image $DI_1(u,v)$ defines the color when the point is fully illuminated. In all our examples DI_0 and DI_1 are always painted by an artist. It is also noted that that this formulation closely resemble to Gooch shading formulation [9]. Figure 1.4 demonstrate how an artist can turn a hand-painted image into a 3D looking image with shadows.

The advantage of Bezier formulation can be demonstrated by comparing this equation with classical rendering formulation that is given in a ray formula as

$$I(u,v) = DI_0(u,v) + VI(u,v)c(u,v) \tag{5.3}$$

Note that we did not change $DI_0(u, v)$, since mathematically speaking it corresponds to the ambient term of classical shading equation. However, the term that corresponds to diffuse term in classical equation is a vector and will be computed as $VI(u, v) = DI_1(u, v) - DI_0(u, v)$ to obtain the same result. Note that $VI(u, v)$ may not necessarily be an image, since for some (u, v) values $DI_1 - DI_0$ can be negative. In practice, this can happen very frequently for artist defined control images (See control images in Figure 5.1). In other words, if we do not use a formulation that does not guarantee convex hull property, such as ray equation, we significantly restrict the creativity of the artist.

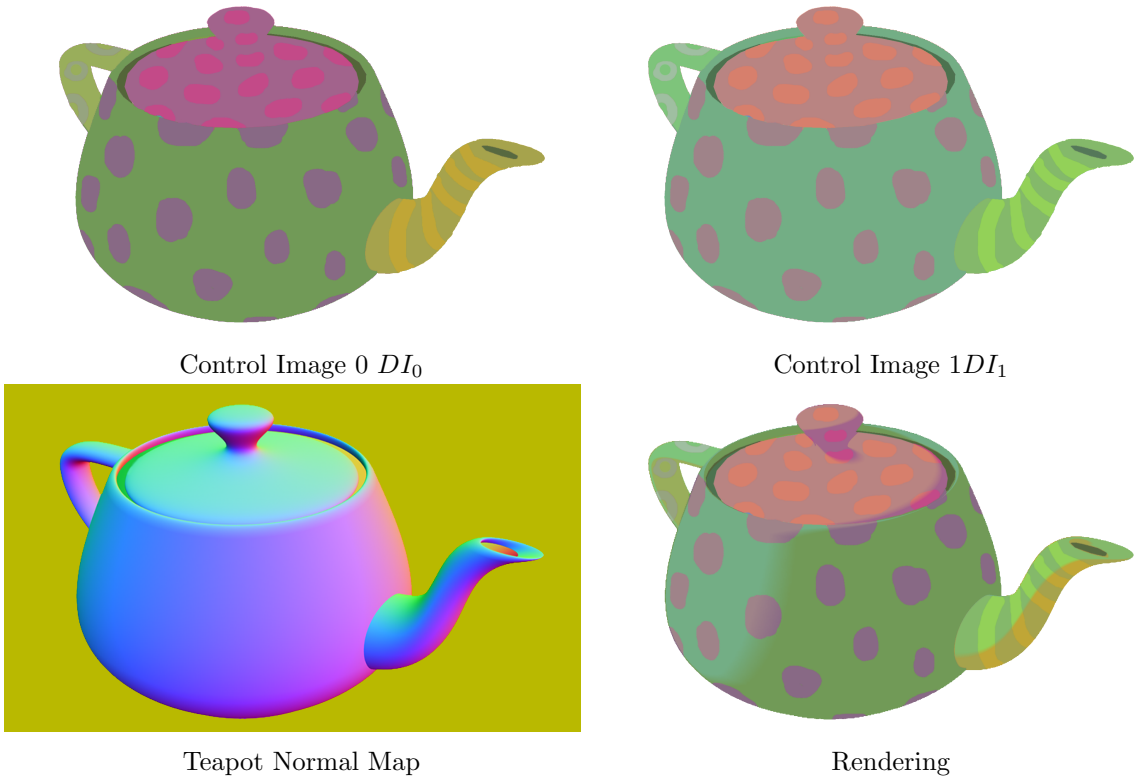
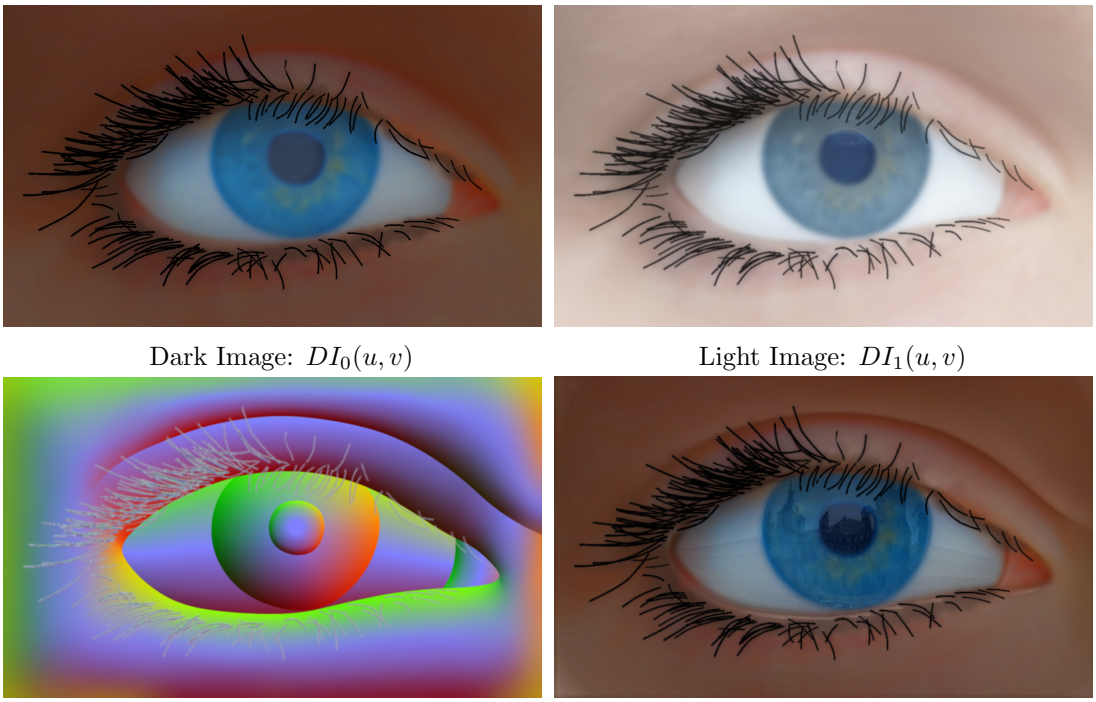


Figure 5.1: Teapot example - Model from Lumo, used by permission of Fleeting Image Animation, Inc.



Dark Image: $DI_0(u, v)$ | Light Image: $DI_1(u, v)$

Shape Map: $SM(u, v)$ | Rendering

Figure 5.2: Reflections on the eyeball - This is obtained with α_I term by using slight transparency on eyeball region. Since the overall α is 1, the rest of image is diffuse. The shape map, and initial version of light image are obtained by a sketch based interface. Dark and light images are later painted by an artist using the initial version of light image as a guide

A shape map with two images DI_0 and DI_1, and an associated shader parameter corresponds to one rendered image, namely I. This particular image can refract and reflect images behind and in front of it. As discussed before, in simple layered scenes we assume that there is only one image behind and in front of the rendered shape map image for the sake of simplicity. Let $BI(u, v)$ and $EI(u, v)$ denote these images, called "background image" and "environment image" respectively and let $\alpha_I(u, v)$ denote opacity of I. Note that shape map transparency defines the shape of $callD$ and has nothing to do with reflection and refraction. On the other hand both

DI_0 and DI_1 can have transparent regions. The term α_I of I results in combined transparencies of the two images DI_0 and DI_1 during rendering. This term is used to make only certain parts of the image transparent or reflective (see Figure 5.2), or obtain the effect of subsurface scattering (see Figure 5.3). We also have a user-defined global parameter, α_G, such as those in GIMP or Photoshop, that can make the entire layer more transparent. Then $\alpha_C(u,v) = \alpha_G\alpha_I(u,v)$ denotes combined transparency of the layer $I(u,v)$. Using combined transparency, α_C, we compute the composited image as follow:

$$CI(u,v) = \alpha_C I + (1-\alpha_C)(fEI(R) + (1-f)BI(T)).$$

where $R(x,y) = (u,v)$ represents reflection mapping; $T(x,y,\eta) = (u,v)$ represents refraction mapping; and the term $f(x,y,\eta)$ represents a Fresnel term. For perfect mirrors, we simply use a Fresnel term $f(x,y,\eta) = 1$. For each of these functions, it is possible to use physically correct formulas, but those formulas do not provide the kind of artistic control we want to provide. Therefore, we introduced linearized formulas that can provide artistically inspired versions of refraction, reflection and Fresnel, which can be specifically used for compositing images. How we compute these three terms will be explained later.

In conclusion, to create dynamic 2D artworks, artists have to create at most two control images in addition to shape maps to control all effects from shadow to reflection and refraction. In most figures we included artist created control images to provide an idea about the process.

5.4.1 Cartoon vs. Smooth Shading

It is widely known that we can smoothly turn diffuse shading into cartoon shading by re-mapping c values in Equation 5.2. For the sake of brevity, we provide the re-

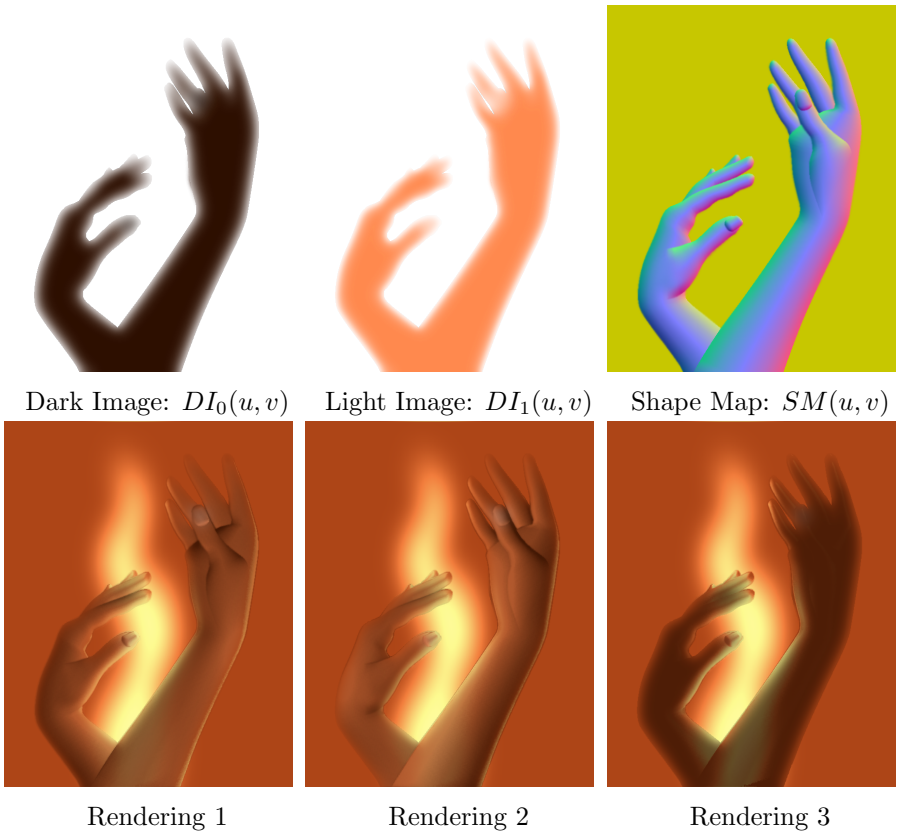


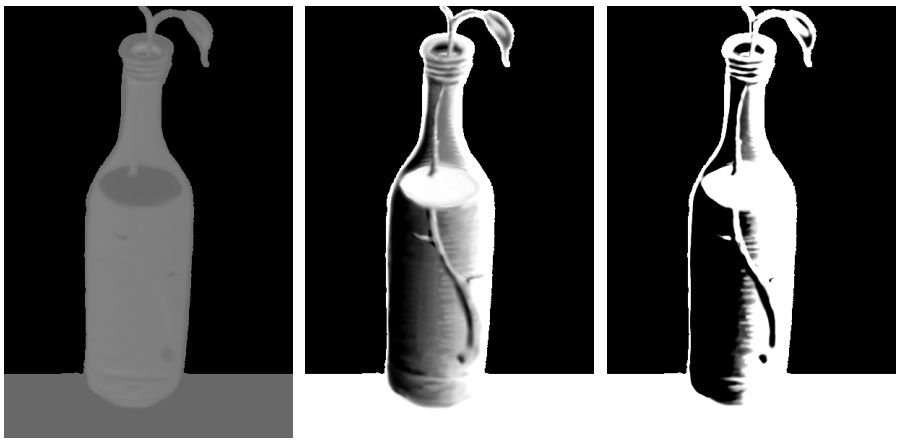
Figure 5.3: Subsurface scattering effect - This is obtained by using α_I term by using slight transparency around the silhouette regions of hands and arms. Shape map, dark and light images are created with a sketch based interface

mapping formula that we use to control the smoothness of the shading as:

$$c \longleftarrow TR(\frac{c - c_0 - 0.5}{\delta_c}) \tag{5.4}$$

In this equation, $TR()$ is a truncation operation, which simply clamps the values if they are smaller than zero or larger than one. The variables c_0 and δ_c are real numbers that are normally in between 0 and 1. These parameters are used to obtain final result from the shading, shadow and ambient occlusion computations.

The term $\delta_c = 0$ corresponds to cartoon shading and $\delta_c = 1$ corresponds to normal



$\delta_c = 2$ $\delta_c = 1$ $\delta_c \approx 0$

Figure 5.4: The effect of the parameter δ - In this case, $DI_0(u, v)$ is a black image and $DI_1(u, v)$ is a white image and background is black. Note that it is possible to allow δ larger than 1, but this choice makes the image flatter and it is not really useful

diffuse shading. The values of δ_c that are larger than 1 flatten the image. Figure 5.4 shows an example of the effect of δ_c. The c_0 parameter controls shifts in the position of 0.5 and is used to control the ratio of dark and bright regions.

6. RENDERING MOCK 3D SCENES

In this chapter, we first talk about how shading parameters are combined. Then, we will explain the algorithm to compute ambient occlusion. Later, we will present shadow computation algorithm, and how the thickness and displacement map are used. And finally we will briefly talk about how the shadow is computed in multi-layer case.

6.1 Shading Parameters

As we discussed in Section 5.4, the core of rendering is to compute the shading parameters. Let $c_D(u,v), c_A(u,v), c_S(u,v) \in [0,1]$ denote parameters that are computed from diffuse shading, ambient occlusion and shadow computation. We combine these three parameters to obtain an overall shading parameter $c(u,v) \in [0,1]$ as in Equation 5.3, with an operator guarantees that $c(u,v)$ is always between $max(c_D(u,v), c_A(u,v), c_S(u,v))$ and $min(c_D(u,v), c_A(u,v), c_S(u,v))$. Examples of such operators are multiplication $c(u,v) = c_D c_A c_S$ and mean $c(u,v) = w_D c_D + w_A c_A + w_S c_S$ where $w_D + w_A + w_S = 1$ and $w_D, w_A, w_S \geq 0$.

Diffuse illumination comes directly from the lights that are represented as point lights. Using light positions, we obtain shading parameters such as diffuse reflection and shadow. For simple layered surfaces, we only consider self-shadows, which we also call local shadows. These are shadows that are casted by mock 3D shapes onto themselves. In billboard based scenes, mock 3D shapes can also cast shadow to others.

In the following sections, we will talk about the computation of ambient and shadow parameters in details (As for diffuse, we simply use classical dot product, so we will not talk in detail here). And since simple layer shape map is a special case

of billboard case, we will talk about the computation as if they are all in Billboard case.

The general notations we will use are as follows: 3D Lighting position is denoted as $\vec{L} = (x_L, y_L, z_L)$. And we use $p(u, v)$ to denote a pixel in the shape maps with image position (u, v), and $\vec{p}(u, v) = (x(u, v), y(u, v), d(u, v))$ denote the 3D position of this pixel in the unit cube of billboard case, as we discussed in Section 5.3, the normal information associated with this pixel is denoted as $\vec{n}(u, v) = (sn_0(u, v), sn_1(u, v), \sqrt{1 - s^2 * n_0^2 - s^2 * n_1^2})$. And for the sake of simplicity, we will omit (u, v) in most of the case.

6.2 Ambient Occlusion

Ambient occlusion parameter can simply be considered as a function of mean curvature [10]. In this work, we adopt a simple qualitative estimation of ambient occlusion from normal map. For a pixel $p(u, v)$ on the normal map, we compute our ambient occlusion using an idea like image processing.

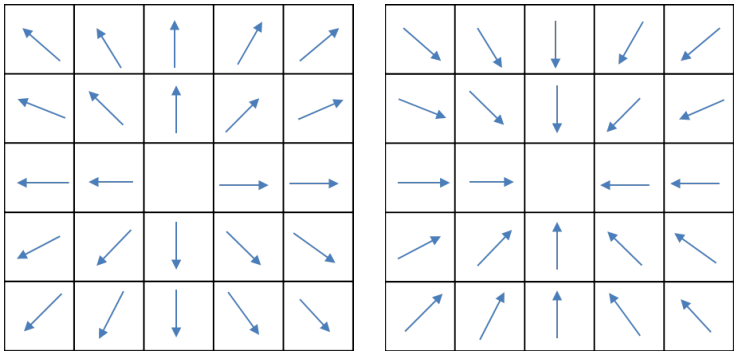
We define a filter of window size $2M + 1 \times 2M + 1$, and each entry of this filter is a 2D vector, which is,

$$a(i, j) = \begin{cases} 0 \text{ for } i = j = 0 \\ \frac{(i,j)}{|i,j|} \text{ for others} \end{cases}$$

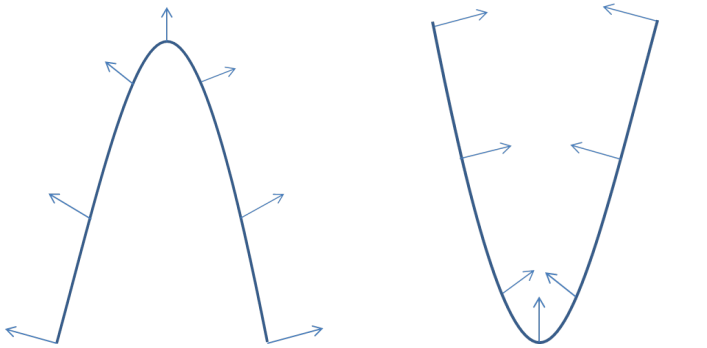
where i and j are the indices within the window coordinates, which are $i, j = \{-M, -M + 1, ... - 1, 0, 1, ...M - 1, M\}$, and $i \neq j = 0$. Then our ambient occlusion parameter is obtained through a convolution of this filter with the 2D vector field of the (n_0, n_1) components from the normals of our shape map. So for the

ambient parameter $c_A(u,v)$ of pixel $p(u,v)$,

$$
\begin{aligned}
c_A(u,v) &= \frac{2}{(2M+1)^2} \sum_{i=-M}^{M} (n_0(u+i, v+i), n_1(u+i, v+i)) \cdot a(i,j) \\
&= \frac{2}{(2M+1)^2} \sum_{j=-M}^{M} \sum_{i=-M}^{M} \frac{n_0(u+i, v+j) * i + n_1(u+i, v+j) * j}{|(i,j)|}
\end{aligned}
$$



(a) Maximum mean curvature (b)Minimum mean curvature



(c) Underlying shape of (a) (d) Underlying shape of (b)

Figure 6.1: Min-max mean curvature - (a) Shows the 2D vector fields of the first two components that can reach maximum mean curvature, while (b) shows the case when it reaches minimum. (c) and (d) shows the underlying shape that can generate the normal that has the vector fields like (a) and (b) separately

It can be seen from the the above equation that the filter is a vector field that all

the vectors are pointing out from the center. And if the 2D normal vectors of (n_0, n_1) has exactly the same direction, then we can reach the maximum ambient strength, as shown in Figure 6.1(a). If the 2D normal vector (n_0, n_1) has totally opposite directions, we got the minimum ambient strength as shown in Figure 6.1(b). Intuitively, they correspond to the underlying shape like Figure 6.1(c) and (d). Therefore, this is a reasonable computation for ambient occlusion. In the following, we are going to show mathematically how our computation in Equation 6.1 is a qualitative estimation to mean curvature.

The mean curvature can be estimated by the average of all the curvatures from the lines that passes through point $p(u, v)$. However, our 2D vector field of normal is discrete, in order to estimate the curvature discretely, we can build a sample window of $2M + 1 \times 2M + 1$ pixel size around pixel p, and take samples of line direction as v_{ij}, that starts from pixel p_{ij} and points to the symmetric pixel $p_{\{-i\}\{-j\}}$, as shown in Figure 6.2. In this case, we will have totally $\frac{(2M+1)^2}{2}$ lines, and the mean curvature H_p at pixel p can be computed as:

$$H_p = \frac{2}{(2M+1)^2} \sum_{j=-M, j\neq i}^{M} \sum_{i=0}^{M} \kappa_{ij}$$

The curvature along the line that passes through p can be estimated similar as in discrete differential geometry [11], we discretize the line v_{ij} as starting pixel P_0, center pixel P and ending pixel P_1, as shown in Figure 6.3(a), so that the discrete curvature can be estimated as

$$\kappa_{ij} = \frac{\varphi_{ij}}{e_{ij}}$$

, where φ is the change of tangent angle between φ_0 and φ_1, and e is the length of the curve segment, which are shown in Figure 6.3(b). The value of s can be easily

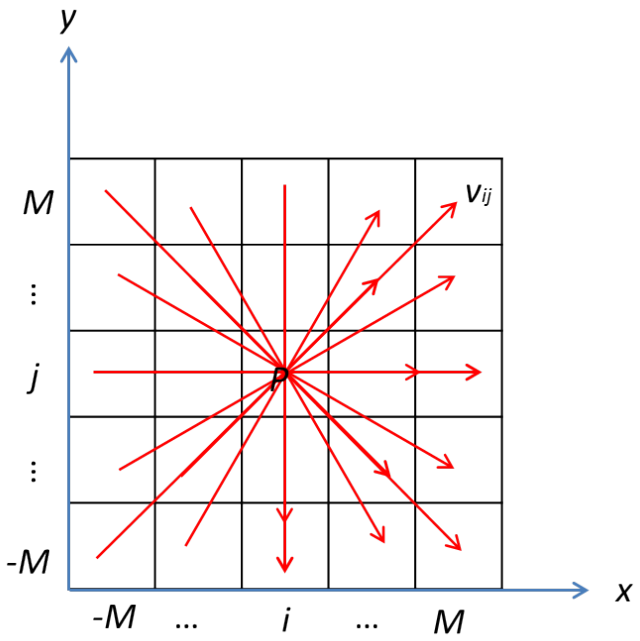


Figure 6.2: Average curvature as mean curvature - Red lines are the sample direction we used to estimate curvature, and with in a window of $2M+1 \times 2M+1$, we totally have $\frac{2}{(2M+1)^2}$ different lines

computed as the length between P_0 and P_1, and the value of φ can be simplified to be a 2D filter convolution as in Equation 6.1.

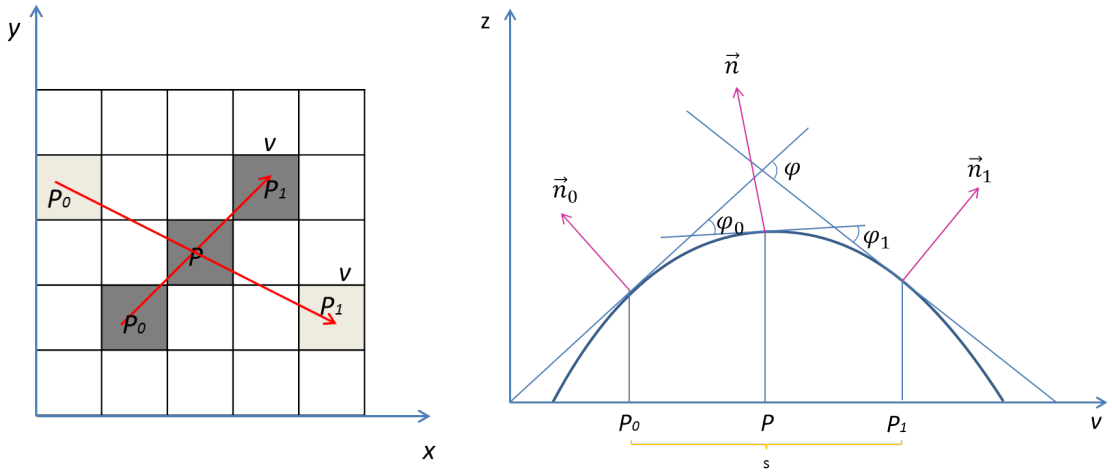
Assuming we are on the plane defined by P_0, P and P_1, with normal N_φ, as shown in Figure 6.4, the value of φ can be computed as

$$\varphi = \pi - \theta_0 - \theta_1$$

, And

$$cos(\theta_0) = \tilde{n}_0 \cdot (-v_{ij})$$

$$cos(\theta_1) = \tilde{n}_1 \cdot v_{ij}$$



(a) Lines used to estimate curvature

(b) Discretised line

Figure 6.3: Curvature estimation - (a) shows the sample lines that we used to estimate and P_0 and P_1 are the sample point we choose along the line in order to estimate curvature, and (b) shows the underlying curvature along the line

, where $\tilde{n}_0$ and $\tilde{n}_1$ are the projection of $\vec{n}_0$ and $\vec{n}_1$ onto plane L and v_{ij} is the 2D direction of the line. Since normals always point towards $(0, 0, 1)$ direction, θ_0 and θ_1 are both within $[0, \Pi]$, and $\cos(\theta)$ is monotonically increasing when $\theta \in [0, \pi]$. We have $\cos(\theta_0) + \cos(\theta_1) \propto \theta_0 + \theta_1$, where $\propto$ denotes positive proportional. Therefore,

$$\varphi = \pi - (\theta_0 + \theta_1) \propto -(\tilde{n}_0 \cdot (-v_{ij}) + \tilde{n}_1 \cdot v_{ij}) = \tilde{n}_0 \cdot v_{ij} + \tilde{n}_1 \cdot (-v_{ij})$$

And since, v_{ij} is perpendicular to N_φ, $\tilde{n}_0 \cdot (-v_{ij}) = \frac{(n_0 - N_\varphi \cdot n_0 * N_\varphi)}{|n_0 - N_\varphi \cdot n_0 * N_\varphi|} \cdot (-v_{ij}) \propto n_0 \cdot (-v_{ij})$, similarly, $\tilde{n}_1 \cdot v_{ij} \propto n_1 \cdot v_{ij}$. So that, Equation 6.2 can be re-written as

$$\varphi \propto (n_0 - n_1) \cdot v_{ij}$$

Since line v_{ij} starts from p_{ij} and points to $p_{\{-i\},\{-j\}}$, v_{ij} is simply (i, j), substituting

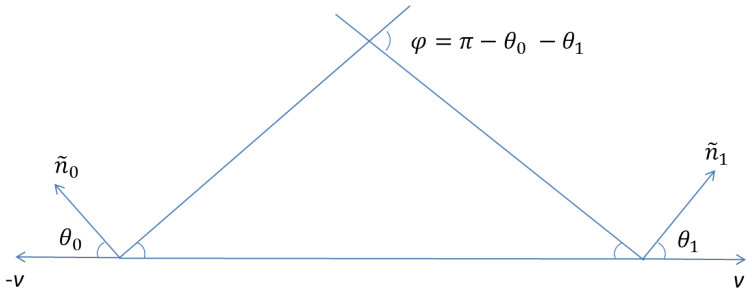


Figure 6.4: Angle from arc-cosine - This figures shows the φ from plane defined by p_0, p_1, so that φ can be computed as $= \pi - \theta_0 - \theta_1$, where θ_0 and θ_1 can be computed as $\tilde{n}_0 \cdot (-v_{ij})$ and $\tilde{n}_0 \cdot v_{ij}$

the component of $\vec{n}_0 = (x(u+i, v+j), y(u+i, v+j))$ and $\vec{n}_1 = (x(u-i, v-j), y(u-i, v-j))$ we have

$$n_0 \cdot v_{ij} + n_1 \cdot (-v_{ij}) = x(u+i, v+j) * i + y(u+i, v+j) * j + x(u-i, v-j) * (-i) + y(u-i, v-j) * (-j)$$

Therefore,

$$\begin{aligned}
H_p &\propto \frac{2}{(2M+1)^2} \sum_{j=0, j\neq i}^{M} \sum_{i=1}^{M/2} \kappa_{ij} \\
&= \frac{2}{(2M+1)^2} \sum_{j=0, j\neq i}^{M} \sum_{i=1}^{M} \frac{\alpha_{ij}}{s_{ij}} \\
&= \frac{2}{(2M+1)^2} \sum_{j=0, j\neq i}^{M} \sum_{i=1}^{M} \frac{n_0(u+i, v+j) * i + n_1(u+i, v+j) * j}{|(i,j)|} \\
&\quad + \frac{2}{(2M+1)^2} \sum_{j=0, j\neq i}^{M} \sum_{i=1}^{M} \frac{n_0(u-i, v-j) * (-i) + n_1(u-i, v-j) * (-j)}{|(i,j)|} \\
&= \frac{2}{(2M+1)^2} \sum_{j=0, j\neq i}^{M} \sum_{i=-M}^{M} \frac{n_0(u+i, v+j) * i + n_1(u+i, v+j) * j}{|(i,j)|} \\
&= c_A
\end{aligned} \tag{6.1}$$

Therefore, we can say our computation of c_A is qualitative proportional to the mean curvature. A particular results of ambient occlusion with different value of M is shown in Figure 6.5

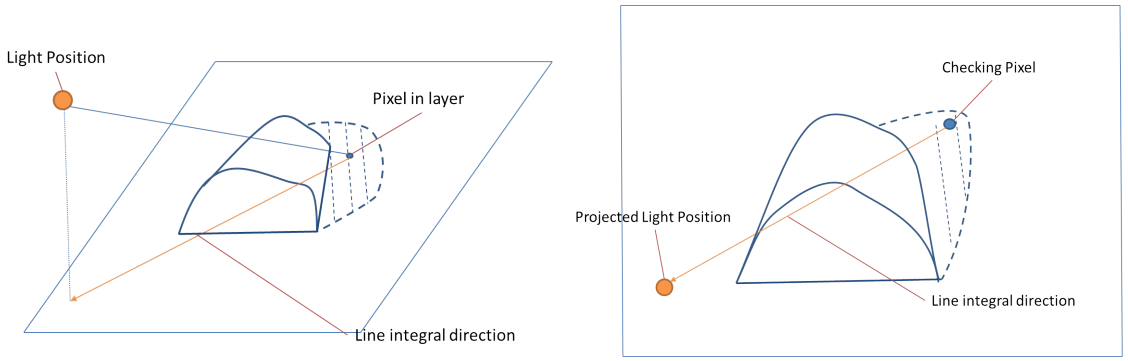
Figure 6.5: Ambient occlusion example - This figure has showed an example of ambient occlusion. And by choosing different size of sample windows, we get different results

And it can also be seen that if we consider the whole 2D vector fields as in continuous domain, the size of M actually corresponds to the number of samples around P, and this does not need to be the size in terms of pixel.

6.3 Shadow

In physics, shadow is created as the light ray is blocked by an obstacle. Similarly, in order to estimate the shadow strength of the current pixel $p(u, v)$ in the image, the basic idea is to check how much the ray between current pixel and the light L has been blocked by the underlying object. We can quantize this by checking how many times the ray has been blocked at a set of sample points along the ray. On each

sample point, what we need to do is to check if the height of the underlying object is higher than the lighting ray at that point. Therefore, the shadow computation in general has two steps. First, we need to estimate the height of the underlying object at the sample points on the line along the direction from $p(u, v)$ to lighting position L. Second, we need to check if the ray at the sample points has been blocked(inside the object) or not.



(a) 3D View of Shadow Cast

(b) Line integral in 2D

Figure 6.6: Shadow cast - (a) shows a shape and its shadow, and how the lighting position is projected onto the image plain. (b) shows what we expected to see from the view angle, and the 2D line integral we actually work on

The core of shadow computation is to estimate the height of the underlying object. In order to achieve this, we adopt a line integral approach. We will first project the lighting position $\vec{L}$ onto the the plane same as the pixel $p(u, v)$ that we are checking, and do the line integral along the projected line with direction $\vec{v}_L$, as shown in Figure 6.6. In simple layer case, we only need to project the lighting position onto the (0,0,1) plane. However, in the billboard case, we have to project the lighting position onto the plane same as $p(u, v)$, and since each pixel might receive

shadow cast from other layers, we also need to project $p(u, v)$ onto all other possible layers in the scene. Details of these will be discussed in the following sections.

6.3.1 Height Estimation

In order to check if a pixel is casted shadow from a certain layer with a plane normal N, we will first project this pixel position $p(u, v) = (x(u, v), y(u, v), d(u, v))$ and light position $L = (x_L, y_L, z_L)$ onto this plane, and we get projected position as P_p and L_p separately. We then compute line integral along the line that starts from P_p to L_p on the plain, we denote this direction as a vector V. With a user-defined sample step size δ_s, we take a bunch of sample points along the line integral direction, their position on the plane is denoted as $\tilde{P}_i$, so $\tilde{P}_0 = P_p$ and $\tilde{P}_{i+1} = \tilde{P}_i + \delta_s * V$. And we also denote the position on the estimated shape that corresponds to $\tilde{P}_i$ as P_i, and we can also see that $\tilde{P}_0 = P_0$. Therefore, the height for each sample points at i is $H_i = (P_i - \tilde{P}_i) \cdot N$. Therefore, if δ_s is small enough, we can estimate P_{i+1} by a linear approximation from P_i, as shown in Figure 6.7.

Assume the plane defined by position P_i and normal n_i is

$$n_i(P - P_i) = 0$$

, As a point on this plane, P_{i+1} satisfies

$$n_i \cdot (P_{i+1} - Pi) = 0 \tag{6.2}$$

And along the integral direction, denote the height increment between P_{i+1} and $P_i + \delta_s v$ as a scalar h multiplied by layer normal N, we have

$$Nh_i + (P_i + \delta_s v) = P_{i+1} \tag{6.3}$$

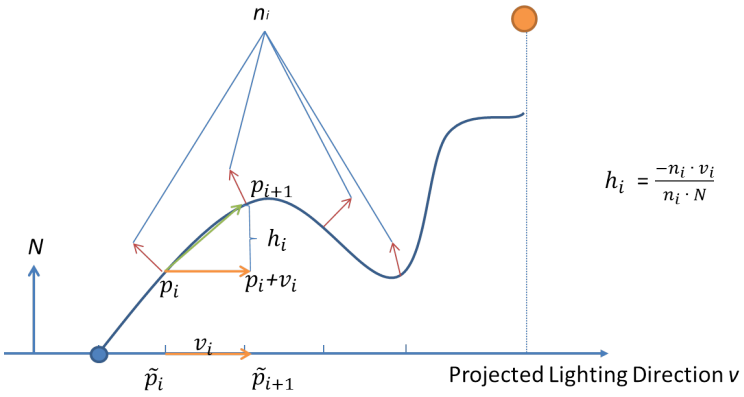


Figure 6.7: Line integral - This figure shows how we compute line integral for height H_i, where h_i is the height incremental between two successive sample points, so the height H_i can be computed as an accumulation of h_i, which is $H_i = \sum_{j=0}^{i} h_j$

substitute Equation 6.3 into Equation 6.2, we have

$$n_i \cdot (Nh_{i+1} + (P_i + \delta_s v) - P_i) = 0$$

so that

$$h_i = \frac{-n_i \cdot \delta_s v_i}{n_i \cdot N}$$

this h_i is the small height increment from position P_i to P_{i+1}, therefore,

$$P_{i+1} = P_i + h_i \cdot N$$

So that the height H_i at sample point i can be computed as,

$$\begin{aligned} H_i &= (P_i - \tilde{P}_i) \cdot N \\ &= (P_0 + \sum_{j=0}^{i} h_j \cdot N - \tilde{P}_0 - i\delta_s v) \cdot N \end{aligned}$$

(6.4)

Since v is perpendicular to N, and $P_0 = \tilde{P}_0$. So, the total height H_i can be computed as the summation from h_0 to h_i.

$$H_i = \sum_{j=0}^{i} h_j$$

6.3.2 Soft Shadow with Shadow Strength

We define shadow strength as a real number $Sha(u, v) \in (0, \text{inf})$, which indicates how much shadow a pixel is receiving with current lighting position. The higher this value is, the darker the pixel should be. And our shading parameter w_S has the inverse relationship with $Sha(u, v)$, which we will talk later. We compute the strength of the shadow as the total length of the parts of light ray when it is inside the the object, intuitively, this computation indicates that the more the ray is inside the object, the darker the affected pixel will be. As shown in Figure 6.8, r_1 has longer lighting ray inside the object than r_0, therefore, when the lighting position is at r_1, pixel will appear to be be receiving more shadow than the lighting position at r_1, and if the lighting position is at r_2, the pixel will appear to have no shadow.

Sha can be computed within the iterative process of line integral as we discussed in Section 6.3.1. Sha is 0 when we start the integral from P_0, and for each sample point P_i. By comparing the value of H_i and the corresponding height of the light ray $\tilde{H}_i$ along the ray, we can simply decide if the lingering ray is inside the object or not. If inside, $Sha \leftarrow Sha + Sha_{acc}/(N \cdot v_h)$, where N is the layer normal and v_h is the 3D direction of light ray from P to L, and Sha_{acc} is a user defined value to control how the shadow strength is accumulated along the line integral.

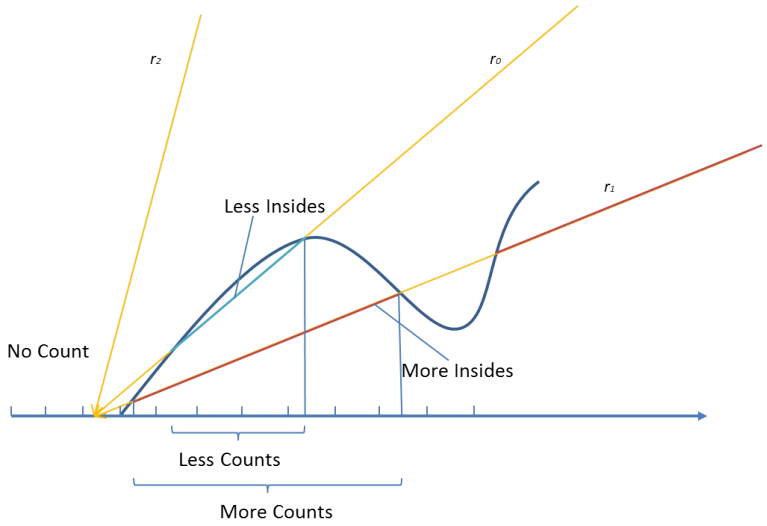
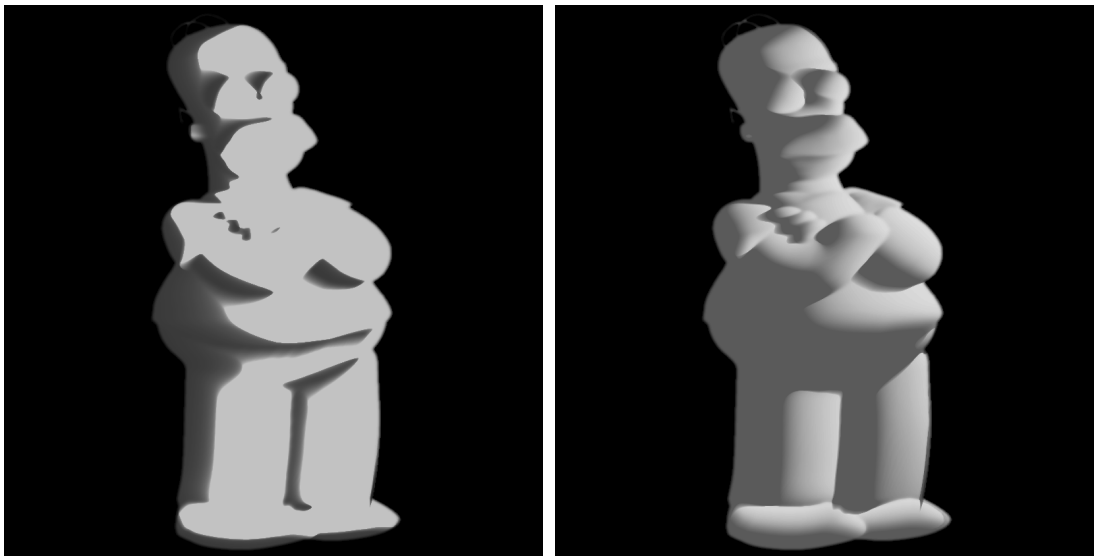


Figure 6.8: Counts of the intersections - Lighting ray r_1 has more counts than r_0, then pixel will be having more shadow(darker) when the light is at the direction of r_1 than at r_0. Lighting ray r_2 has no counts, so when the light is at r_2 direction, the pixel will not have any shadow.

6.3.3 Thin-film and Thickness Map

Shadow strength has provided very good control for the parts that are in shadow, however, by simply checking whether the current height H_i is larger than the ray height or not, the non-shadow parts will appear to be uniform color. Since for these pixels, the lighting ray will never be inside the object, therefore, there would not be any accumulation for *Sha*. So, even though shadow parts are correct as shown in Figure 6.9(a), the non-shadow parts look bad. However, we would like to get the non-shadow part similar as diffuse results which is shown as Figure 6.9(b).

In order to solve this problem, we introduce a concept of thin-film, this thin-film is simply an offset of the original surface, it can be viewed as we add a very small offset of the original surface. Basically, this small offset makes it possible for parts of the lighting ray to be inside the object. And we can show that the resulting



(a) Without Thin-film (b) With Thin-film

Figure 6.9: Thin-film effect - (a) The shadow parts appear to be smooth, but non-shadow parts have uniform color. (b) A much better looking non-shadow parts like diffuse shading

shader parameter w_S from adding this thin-film is a qualitative estimation to the dot product of the surface normal and the lighting direction(which is the classic diffuse shading).

We will start from the planar surface, as shown in Figure 6.10, green line segments are the length that the lighting ray is inside the object from our computation of Sha, denote the thin-film value as tf. It is obvious that the cosine of angle between normal of pixel and lighting ray is $\frac{tf}{Sha}$. As a sphere surface shown in Figure 6.11, r_0 has the maximum value of $\frac{tf}{Sha}$, as in this case, the line segment inside has exactly the same value as tf. The maximum value of $\frac{tf}{Sha}$ equals to 1, which is the same as the diffuse shading. r_2 has the minimum value of $\frac{tf}{Sha}$, since this is the case when the length of line segment inside reaches maximum, while diffuse shading also reaches minimum at this position. Even though diffuse shading has a minimum value of 0,

while $\frac{tf}{Sha}$ can not reach 0, it is obvious to see from Figure 6.11 that our $\frac{tf}{Sha}$ has the same monotonicity as diffuse shading, therefore, we can say $\frac{tf}{Sha}$ is qualitatively proportional to diffuse shading. As w_S has the inverse relationship with Sha, we define our $w_S = \frac{tf}{Sha}$, so that, w_S is also qualitative proportional to diffuse shading.

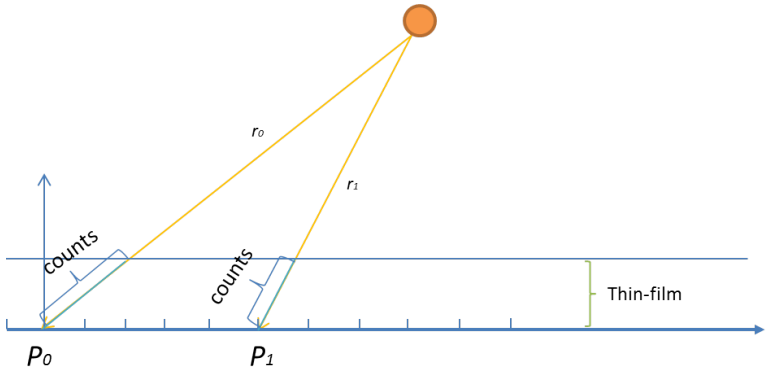


Figure 6.10: Thin-film as cosine on planar surface - Green line segmentation are the length that the lighting ray is inside the object, diffuse shading of each point P can simply be computed as the length of Thin-film over the length of green line segment

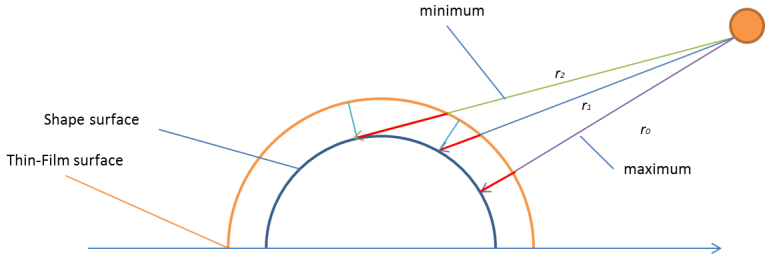


Figure 6.11: Thin-film as cosine on spherical surface - Green line segmentation are the length that the lighting ray is inside the object

And for each shape map, we also have a thickness map, indicating the thickness of the object. The object will be considered as if it is in 3D, and we are trying to find if the light ray goes through the object. In addition to the surface s, which is computed from height, we consider the object between surface created by thin-film, and the the surface created by thickness, as the surface s_1 and s_0, as shown in Figure 6.12. The light height $\tilde{H}_i$ at the sample points P_i can be easily computed as

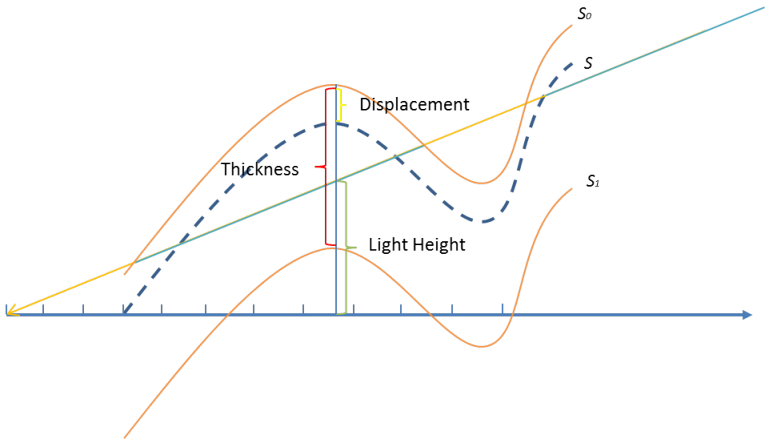


Figure 6.12: Intersection with the lighting ray

a ratio of projected height of current checking pixel $p(u, v)$ and the projected height of lighting position L, based on the length of the current line integral. By comparing the light height $\tilde{H}_i$ and the two surfaces, we can decide if the lighting ray is inside the shape at the current sample point by checking:

$$(H_i + tf - \tilde{H}_i) * (H_i - tf - T - \tilde{H}_i) < 0$$

. If the above statement is true, we can increase the shadow strength value by

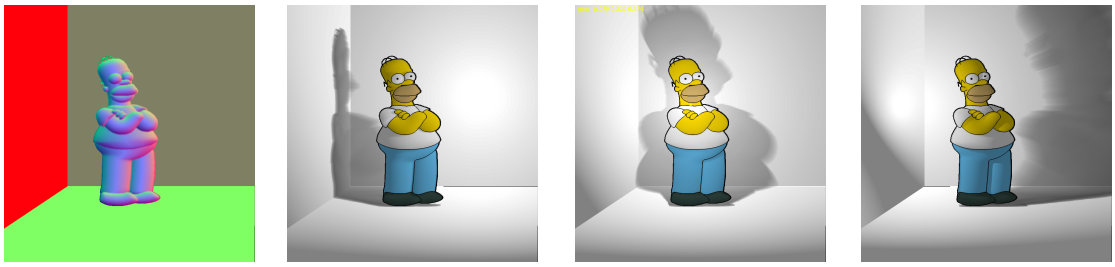
$Sha \leftarrow Sha + Sha_{acc} * (1 - N \cdot v_h)$. In this particular example, we have the back side the same shape as front side. However, in practice, user can deicide how the back side should be related to front side.

6.3.4 Sharp Height Change from Displacement Map

The displacement map is used on top of shadow strength value, as we discussed in Section 4.2, the derivative of displacement map is used as a threshold to decide if there is a sharp change in the height. Therefore, the information carried by displacement map can be interpreted by checking the difference between displacement values of two consecutive sample points along the line integral path: $H_i \leftarrow H_i + \delta_h \lfloor (Z(p_i) - Z(p_{i+1})/\delta_z \rfloor$, where δ_h and δ_z are two user-defined values, δ_h is the amount of the height change that is brought by discontinuity which corresponds to s_1 in Equation 4.2 in Section 4.2, and δ_h is the user-defined quantization term from Equation 4.2 in Section 4.2.

6.3.5 Shadow Cast from Other Layers

When we are using the billboard Mock-3D scene, we have to consider shadows casted from other layers, as shown in Figure 6.13. In order to check if Shape 2 is creating shadow on a pixel $p(u, v)$ in Shape 1, we have to project this pixel from Shape 1 to Shape 2, as well as the lighting position on to Shape 2, and see if Shape 2 is creating Shadow on the projected position of $p(u, v)$. In this case, every pixel is possible to receive shadows from the shapes in every layers, therefore, for each pixel, we will project it onto all other possible layers to see if it can receive shadow from the shape in that layer, as shown in Figure 6.14. And within each layer, the computation of shadow cast is the same as we discussed before. And for each visible pixel from scene, we will still keep one single Sha value, and the check of whether Sha should be increased will be for all the samples in all layers.

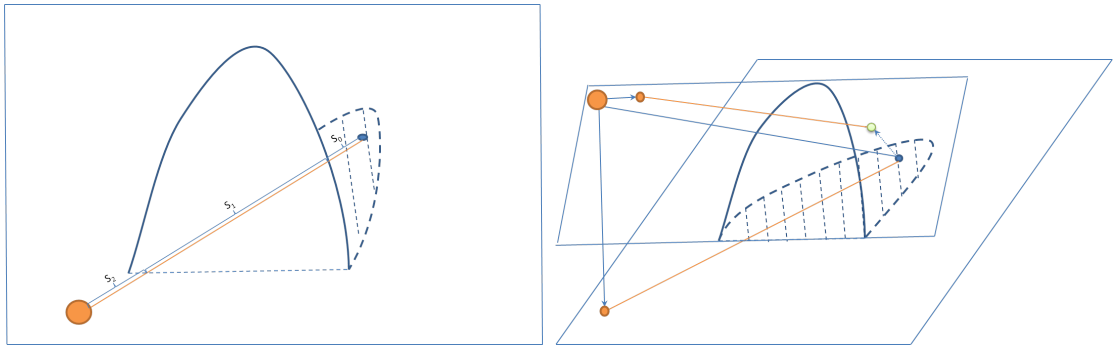


(a) Shape Map (b) Light on the right (c) Light in the center (d) Light tn the left

Figure 6.13: Shadow cast in billboard mock-3D scene - This figure shows how the shadow is casted from other layer. (a) is the shape map of the scene, (b) (c) and (d) show how the shadow cast changes with the lighting position

6.4 Reflection and Glossy Reflection

For reflection mapping, our goal is to provide a simple and intuitive-to-use method for 2D artists. Our method is closely related to sphere mapping, which is one of the most widely used reflection methods [15]. In sphere mapping a far away spherical environment is stored as an image that depicts what a gazing ball (i.e. mirrored sphere) would reflect if it were placed into the environment, using an orthographic projection. Although spherical mapping is one of the simplest methods for obtaining reflections, it can still be complicated for some 2D artists who want to work only with rectangular images. Moreover, in our case, an additional problem comes from the fact that $n_0^2 + n_1^2$ can be larger than 1. Therefore, we need a similar method that can use rectangular environments. Fortunately, there exists a gazing ball shape that can reflect the whole environment into a square image. This particular gazing ball shape can be given by an implicit surface $max(n_0^2, n_1^2) + z^2 = 1$. This shape, when placed in an environment, reflects the whole environment in a square using orthographic projection along the z direction. We can, then, create reflection simply



(a) Projected (u, v) onto all other layers (b) Integration along lighting direction

Figure 6.14: Integral for shadow cast from other layers - (a) shows how the current checking pixel and lighting position are projected onto all the layers in the scene. (b) shows all the line integral that will be used to check if there is a shadow cast

by using the following operation:

$$(u, v) = R((u, v) - (n_0, n_1)) \tag{6.5}$$

This reflection is shown in Figure 6.15.

The only caveat in this approach is that every point on the boundary of a square image reflects the same point in the environment sphere. Therefore, every point on the boundary of the square image has to be the same. In practice, any seamlessly tile-able wallpaper image can be used as an environment map. In our experience, any image works as an environment map, probably due to humans' high tolerance for discontinuities in mirror images.

For realistic reflections, we move the center of the environment image in tandem with the light position. This creates visually acceptable specular reflections. Therefore, we do not think an additional specular highlight is necessary. Glossy reflection is simply obtained using smoothed versions of environment maps provided

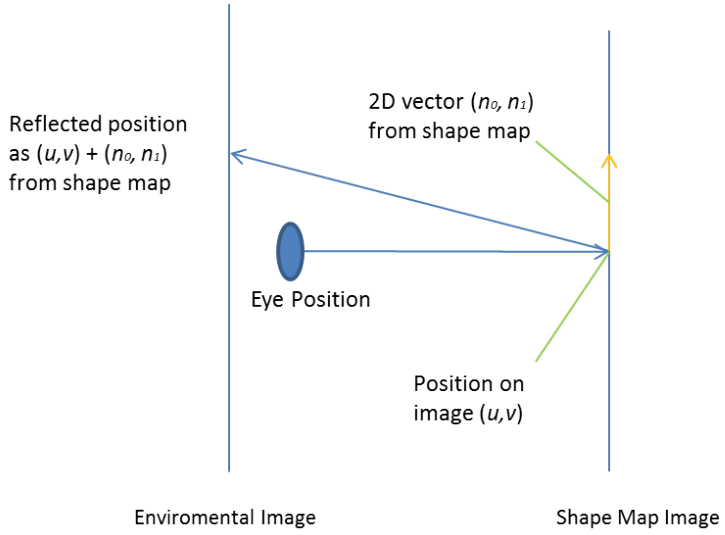


Figure 6.15: Illustration of reflection

by mipmap. Glossy reflections combined with other compositing operations such as multiplication can be used to obtain other effects such as environment illumination.

6.4.1 Refraction and Translucency

For refraction we developed an artificial refraction mapping that provides simple and intuitive artistic control, for

$$(u, v) = T((u, v) - a * t * (n_0, n_1)) \tag{6.6}$$

where $a \in [-1, 1]$ is a user-defined global parameter that corresponds to index of refraction as the value of a is computed log_2 of $\eta = \eta_2/\eta_1$. Although shape maps are not supposed to have well-defined unit normal vectors, it is still possible to evaluate physical sense of this equation qualitatively. For instance, when $a = 0$, there is no displacement of the background image, which is exactly what we expect when $\eta = 1$. In the regions where $t = 0$ regardless of a, there is again no displacement, which is

also what we expect to see in transparent regions that are extremely thin.

The values of a that are larger than 0 correspond to $\eta > 1$, which is representative of travel from air to water or glass. In this case, the incoming eye-ray is $(0, 0, -1)$ since we assume orthographic projection. The refracted ray will be bent towards the vector $(-n_0, -n_1, -1)$. Therefore, we can assume that this vector can be given as a weighted average of the two vectors $(0, 0, -1)$ and $(-n_0, -n_1, -1)$. Using a as a weight, we obtain a vector $(-an_0, -an_1, -1)$. If the object simply consists of two plates with thickness t, this vector will hit the back side of the object at $(u - tan_0, v - tan_1, -t)$. Once it hits the back side, we assume it is refracted back to $(0, 0, -1)$ and continues until it hits the background image. Therefore, we can simply approximate the refraction as a 2D displacement $(-tan_0, -tan_1)$. This process is shown in Figure 6.16. The biggest advantage of this equation is that refrac-

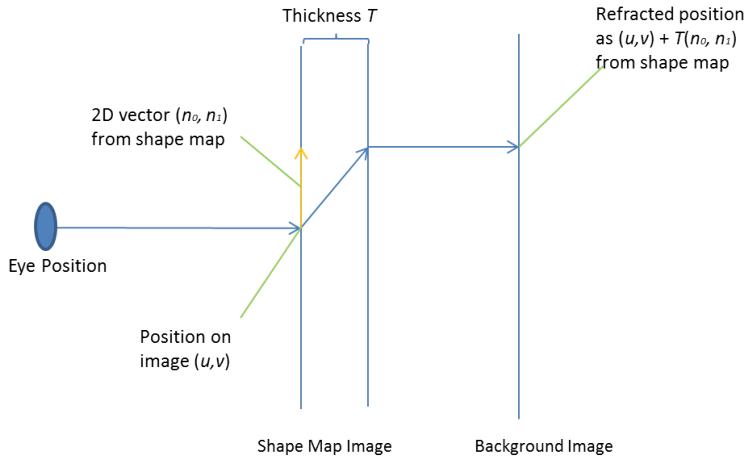


Figure 6.16: Illustration of refraction

tion changes linearly with global parameter a. This provides predictable control for artists. Following the same logic, the equation provides a vector between $(0, 0, -1)$

and $(n_0, n_1, -1)$ for negative a values. Although the length is not exactly correct, the direction of displacements is correct (this also provides a predictable control). Translucency is simply obtained using smoothed versions of the background images. Moreover, we smooth the background images based on the value of $|(tan_0, tan_1)|$ to improve visual quality and realism.

This refraction computation can also provide easy to understand and control deformations. It is also possible to obtain simpler operations such as translation, scaling and rotation with refraction. For instance, a 2D vector field $(n_0, n_1) = (v, -u)$ rotates a background image 45^0 and uniformly scales the image by $\sqrt{2}$. From the same shape map, using a value one can obtain a full range of scaled rotations. Shapes maps that rotates are also good examples of impossible objects since they do not correspond any 3D shape.

6.5 Fresnel Effect

For artists the Fresnel effect is important since it changes how reflection and refraction is composited based on the incident angle and the index of refraction. From an art direction point of view, the number of parameters for defining Fresnel is undesirable. Therefore, the first issue is to reduce the number of parameters. Let $t = \theta(n_0, n_1)$ denote a function that converts (n_0, n_1) into a single variable between $[0, 1]$ that correspond to incident angle. Based on our earlier discussion, such a function can be obtained in several ways. Since there already exists a user-defined global parameter a that provides the index of refraction, we can rewrite the variables of the Fresnel equation as $f(x, y, \eta) = f(t, a)$. By observing that the most important issue in Fresnel is to control the regions of strong reflection with the index

of refraction, we further simplified the equation into the following form:

$$f(t, a) = af(t, -1) + af(t, 1) \tag{6.7}$$

Based on this equation, artists need only define two Fresnel curves that are given for $a = -1$ and $a = 1$. We then can simply interpolate these curves using a. We also observed that curves $f(t, -1)$ and $f(t, 1)$ can simply be piecewise linear curves. Figure 6.17 shows a hand-drawn effect obtained with an exaggerated Fresnel function. In a dynamic version, the user can move the positions of white regions with a slider that controls a value.

(a) (b) (c)

Figure 6.17: Fresnel effect controlled by pseudo index of refraction - This figure shows Fresnel effects using a black background and white environment map. By changing a value, from (a) to (b) to (c) the object appears to be more refractive than reflective

6.6 Reflection in Billboard

In Billboard case, we still have reflection from environmental image and refraction from background image. These are computed using the same linearized formula as in simple layer case. The only difference is the a from Equation 6.5 will be different for each different shape, and the depth of each pixel will also be taken into consideration. Therefore, Equation 6.5 becomes:

$$(u, v) = T((u, v) - (1 - d(u, v)) * (n_0, n_1))$$

where $d(u, v)$ is the depth of current pixel at (u, v), as we introduced in Section 5.3, Billboard scene is inside a unit cube, and environmental image is placed at $d = 1$. Similarly, the refraction from Background image in Billboard scene can be written as:

$$(u, v) = R((u, v) - a(u, v) * d(u, v) * t * (n_0, n_1))$$

where $a \in [-1, 1]$ corresponds to index of refraction and will be different for each different shape.

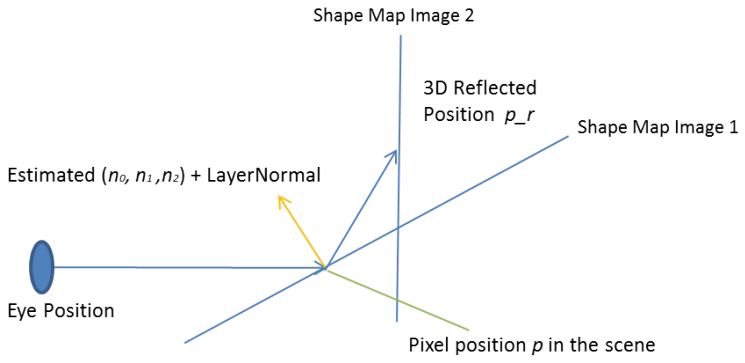


Figure 6.18: Billboard reflection

In addition to environmental image and background image, Billboard case will also consider the reflection/refraction from other shape maps, however, our computation is based on classical 3D rendering technique, which is shown in Figure 6.18, we will not talk this in detail. The reflected ray will be checked with all other layers to see if there is any intersection, if there is an intersection, the pixel color from the interaction point will be taken to replace the color from the original checking point. And the new image with replaced color will be the output of compositing process. And in order to have reflection of reflection/refraction, we used similar iterative approach as in Path Tracing [7]. The first input color for compositing process is the output from shading process. And the output of the compositing process will be saved as the new input for the next compositing iteration, and the whole process will be repeated several times to get the final results.

7. IMPLEMENTATION AND RESULTS

In this chapter, we will first talk about the advantage and disadvantage of GPU implementation. Then we will talk about some details in rendering that are not covered in Chapter 6. Finally, we will show more results from our program.

7.1 GPU Implementation

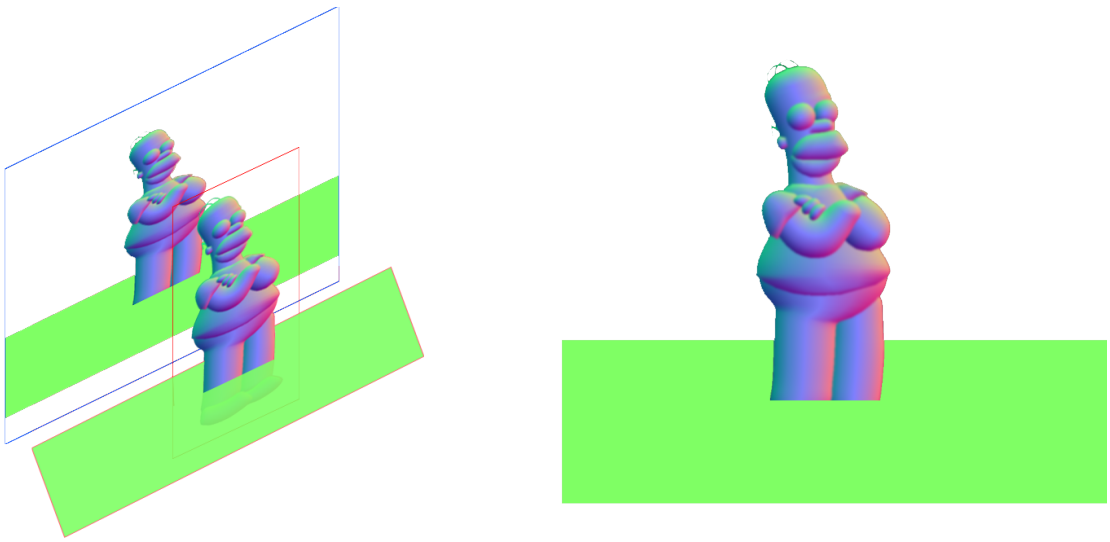
We used GPU to implement all the rendering and compositing system, and CPU is only used to arrange shapes in Mock-3D scene. The advantage of GPU is the strong parallel computing power, and since our shape map representations are all kept in image format, they can be easily passed to GPU and processed as textures. Each single pixel can be processed separately, which is especially powerful for compositing part. However, one limitation with GPU implementation is, for each rendering pass, GPU can only register 32 different textures, therefore, we tried to convert multi-layer shape maps in Billboard case as one shape map with layer label information for each pixel, rather than several separate shape maps. Another more serious limitation with GPU implementation is, when each pixel is processed separately, this particular process does not have access to the processed results from other pixels, so in the line integral computation, all the pixels along the line can not use the results from previous pixel. This leads to a waste of computation.

7.1.1 Simple-layered Scenes

Simple-layered mock-3D scene will have totally six different images, one normal map image, one thickness image, one DI_0 image, one DI_1 image, one background image, one environmental image, and optional a displacement image. All of these image will be treated as textures in GPU.

7.1.2 Billboard-based Scenes

From implementation side, we can still consider the billboard mock-3D scene as one multi-layer shape map. This multi-layer shape map still appears as one single image, which is obtained by the orthogonal projection of all the shape maps in the mock 3D scene, according to their layer normals and depth. In other words, the multi-layer shape map contains all visible pixels from the shape maps of the mock-3D scene according to current viewing angle. Figure 7.1 shows how the multi-layer shape maps is obtained from mock 3D scene.



(a) Shape Maps and Layers in 3D

(b) Multi-layer Shape Map

Figure 7.1: Billboard-based mock-3D scene

In addition to the normal, thickness and displacement information from single layer shape map, a multi-layer shape map will have a label for each visible pixel $Lab(u, v)$ indicating which layer this visible pixel belongs to. And for each layer

A, we have a layer size (W_A, H_A), center position in 3D $\mathbf{D_A}$, normal vector $n_A = (n_{0A}, n_{1A}, n_{2A})$, where A is the layer label and $n_{0A} \in [-1, 1], n_{1A} \in [-1, 1], n_{2A} \in [0, 1]$. From this information we can compute a depth value for every pixel inside one layer. Besides $x(u, v)$ and $y(u, v)$, we will have a fuzzy depth information $d(u, v)$ for each pixel, which can be computed from n_A and D_A, as follow:

$$d(u, v) = ((W_A - u) \cdot n_{0A} + (H_A - v) \cdot n_{1A})/n_{2A}$$

where $A = Lab(u, v)$. Please note, this depth is computed from layer orientations, which means it has nothing to do with the actual shape of the underlying 3D model. As inside a unit cube the range of $d \in [0, 1]$, and we placed shape maps in such a way that, $d = 1$ means closet and $d = 0$ means farthest.

From controlling side, the user has to set all the control images for each shape rectangles as in the simple layered case. Besides, the user will be able to change the size, position and normal of the shape rectangle. And as we mentioned above, the shape rectangle will never go outside of the unit cube. Therefore, if the user try to changes n_A to a new value that makes one of the corner of shape rectangle having a depth larger than 1 or smaller than 0, then the change n_A will be reverted.

7.2 Rendering Mock 3D Scenes

7.2.1 Diffuse

We can simply compute the dot product of the first two components (n_0, n_1) of normal and the lighting direction. which is

$$c_D = \frac{n_0 * (x_L - x) + n_1 * (y_L - y) + n_2 * (z_L - d)}{|(x_L - x, y_L - y, z_L - d)|}$$

In single layer shape map case, this can be further simplified as,

$$c_D = \frac{z_L * (n_0 * (x_L - x) + n_1 * (y_L - y))}{|(x_L - x, y_L - y)|}$$

A simple change to this function can make a parallel light looking, such as

$$c_D = \frac{n_0 * x_L + n_1 * y_L + n_2 * (-d)}{|(x_L, y_L, z_L)|}$$

7.2.2 Ambient Occlusion

As we discussed, the size of the the sample window does not need to be at pixel size, and since we are using texture to store our information, the sample could be at inter-pixel level. User can give a δ_u and δ_v value indicating the sample step along x and y direction, and the convolution is applied on the neighborhood point $(u + i\delta_u, v + j\delta_v)$ of (u, v), Therefore, our ambient occlusion is implemented as

$$c_a(u, v) = 0.5 \sum_{i=-N}^{N} \sum_{j=-N}^{N} \frac{w_{i,j} A_{i,j}(u, v)}{\sum_{i=-N}^{N} \sum_{j=-N}^{N} w_{i,j}} + 0.5 \tag{7.1}$$

where N is a user-defined number of samples, and

$$A_{i,j}(u, v) = \delta(n_0(u + i\delta_u, v + j\delta_v), n_1(u + i\delta_u, v + j\delta_v)) \bullet (i\delta_u, j\delta_v)$$

,

7.2.3 Shadow Cast from Own Layer

After projecting the lighting position onto the same plane as pixel, we do the line integral, actually it is pretty simple as an iterative accumulation:

$$H_i = H_i + \frac{n_i \cdot \delta_s v_i}{n_i \cdot N_j}$$

In the single layer case, the above function can be further simplified as

$$H_i = H_i + \frac{sn_0 * x_L + sn_1 * y_L}{\sqrt{(}1 - s^2 n_0^2 - s^2 n_1^2)}$$

The lighting ray height at current sample point can be computed as:

$$H_r \leftarrow H_p + \frac{(H_L - H_p) * |P_i - P_p|}{|P_p - L_p|}$$

And as we mentioned in Section 6.3.3, we can set back side differently, usually we would recommend symmetric but a little simpler back side to have a better visual effects, such as in Figure 7.2 Then we can check if $(H_i + S_{disp} - \tilde{H}_i) * (-0.1 * H_i - S_{disp} - T - \tilde{H}_i)$ is smaller than 0, and if this is true, the shadow strength will be added with $Sha_{acc}/(N \cdot v_h)$, the choice of Sha_{acc} will be discussed in the next section.

7.2.4 Shadow Cast from Other Layer

In billboard case, only the information of the visible points is recorded in shape map, and we will use the label information of each point to identify the layer of that pixel. So that, along the integral direction from P_p to L_p, each sample point P_i may have differen labels as the starting position P_p, and may have different labels from current layer. We identify these as four different cases. In each case, integral behavior and accumulated shadow strength will act differently.

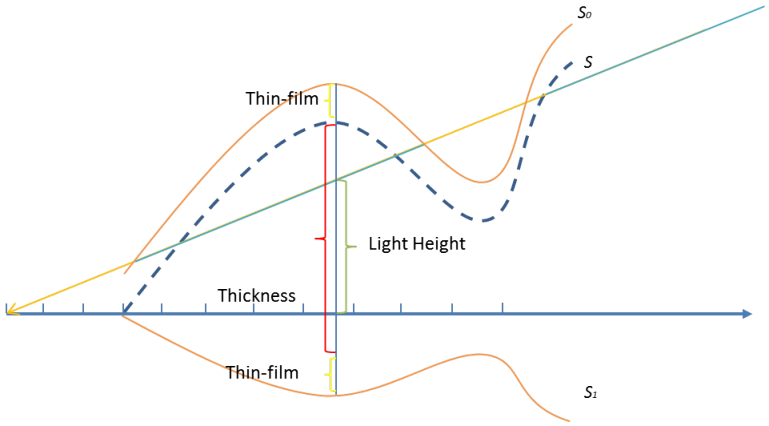


Figure 7.2: Symmetric simpler back-side

Case 1: Label of the starting position p, the current checking layer and the sample position P_i are all the same. This case indicates that the shadow computation(accumulated height from line integral) is using the sample pixels from the same layer as checking layer, and the shadow is also cast onto the pixel from the same layer. So, the line integral will still follow the Equation 6.3.1, and if it is casting shadow, the shadow strength will be added with a user defined value for the case when shadow is cast from the shape in same layer. This is the only case in the simple layer case.

Case 2: Label of the current checking layer is the same as the label from sample position P_i, but they are different from the label of starting position p. This case indicates we are on a different layer to check if there is any shadow cast, and the shadow computation is still on the sample pixels from the same layer as the checking layer. So, the line integral is the same as in case 1, and if it is casting shadow, the shadow strength can be added with a user defined value different from case 1.

Case 3: Label of the starting position p is the same as the label from checking layer, but they are different from the label of sample position P_i. This case indicates

that the the shadow computation is not reliable, but the starting position is on the same layer as the current checking checking layer, we should keep the consistence. This case happens when partial of the line integral is blocked by the pixels from other layer, as shown in Figure 7.3. In this case, the accumulated height H_i should be kept the same as previous iteration. And the shadow strength should be as the same as in case 1.

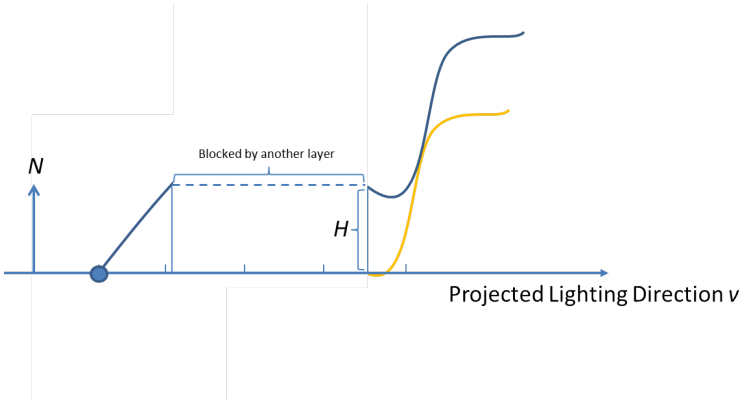
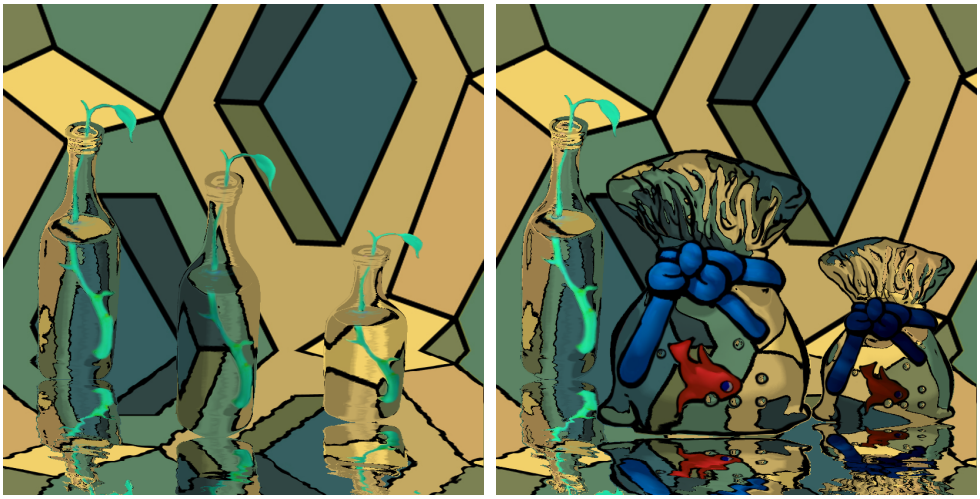


Figure 7.3: Occlusion from other layer

Case 4: Label of the starting position p, the current checking layer and the sample position P_i are all totally different from each other. This case indicates the integral is not reliable, and this result should not have any effects on other layer. Therefore, the shadow strength can simply be set to zero in this case. For every pixel $p(u, v)$, we will have a label $b(u, v)$, and its associated depth $D(u, v)$, so the pseudo 3D position of this pixel can be represented as $(x(u, v), y(u, v), D(u, v))$, for totally N layer, the pseudo code is as follow:

7.3 Reflection and Refraction

The reflection and refraction of background and environmental image is straight forward, in GPU, we simple need to added a displacement of position in texture. But the reflection of reflection/refraction is hard to implement since GPU does not have the information from other pixels when it is processing current pixel. Therefore, we have to compute all the pixel together after each reflection/refraction, and save the image as a frame buffer, which can be used for the input of compositing process again. We separate the rendering and compositing process into two GPU codes. We save the



(a) (b)

Figure 7.4: Reflection of refraction - This figure shows an example of reflection of refraction, (a) and (b) are with differen object, and please also note each object in the same image also have different refraction index

output of rendering process as the shading results(the output of the shading process is the combination of DI_0 and DI_1 using $c(u, v)$). We then apply the compositing process, and save the result in frame buffer as a new shading result image. Then we

will reapply the same compositing process for a few times, in our implementing it is 4 times. And pretty good results can be obtained, such as in Figure 7.4. However, inter-layer refraction is not possible with our current implementation of Billboard, since the shape in front will block the shape in the back. Partial of the shape information is missing, so that refraction can not be built.

7.4 More Results

In this section, we will show more results from our system. Figure 7.5 and Figure 7.6 shows artistic reinterpretation, all the shape maps are drawn by artist. Figure 7.7 shows the use of shape map from other modeling software, such as Crossshade[24]. Figure 7.8 is a pure non-photo realistic results from our system. Figure 7.9, Figure 7.10 and Figure 7.11 shows the reflection/refraction effects. Figure 7.12 and Figure 7.13 shows the reflection effect from Billboard shape map;

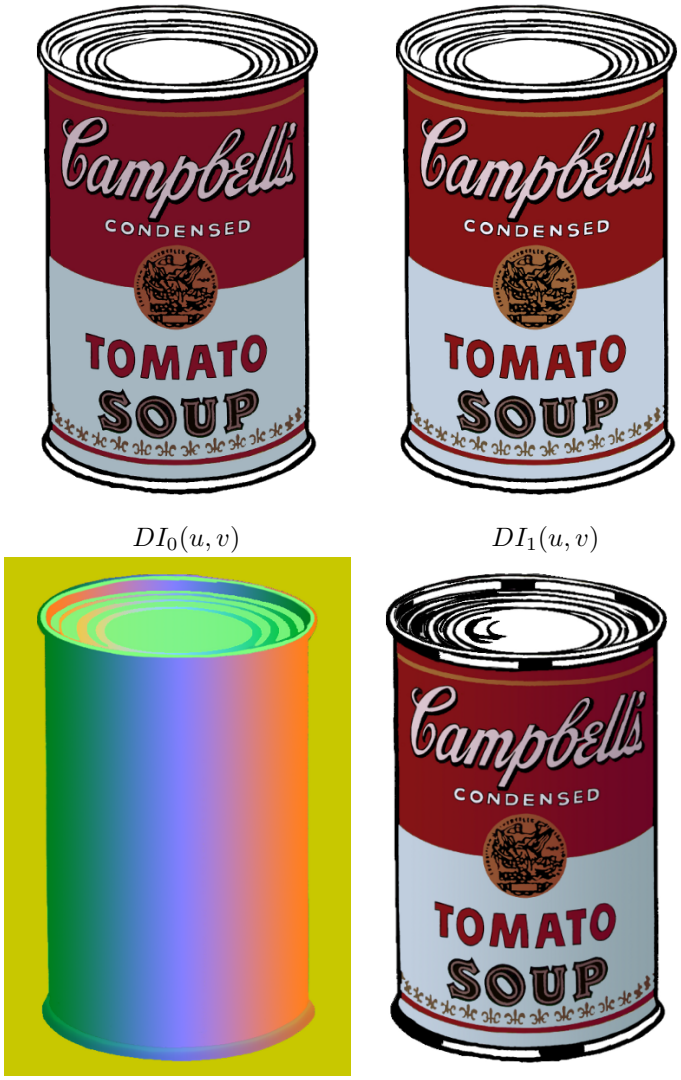
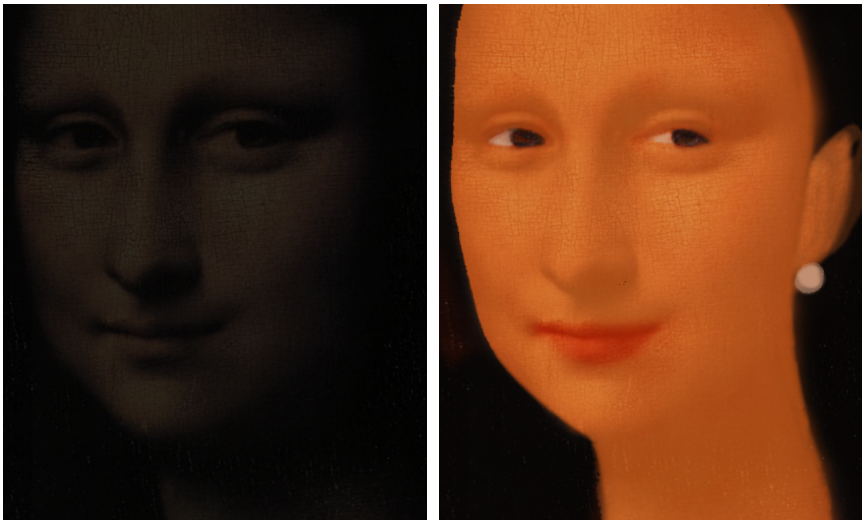
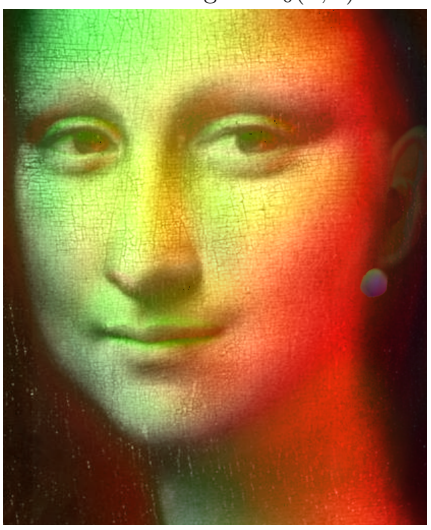


Figure 7.5: Warhol's campbell - This figure shows Re-interpretation of Warhol's Campbell Soup painting with art directed reflections. In his original painting, Warhol painted mirror reflected black areas on top of the can and a very subtle and almost invisible diffuse reflection on body of the can. Using a black and white striped image as an environment map and by painting slightly varying control images, an artist was able to move both subtle diffuse reflection and mirror reflected black in tandem, which can help to better appreciate the idea behind this painting. Creation of control images and 2D vector field did not take more than one hour

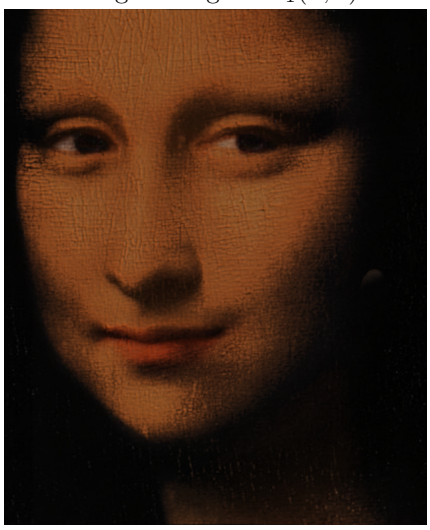


Dark Image: $DI_0(u, v)$

Light Image: $DI_1(u, v)$

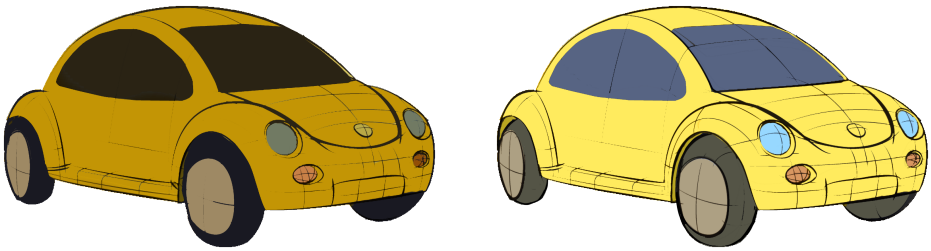


Shape Map : $SM(u, v)$



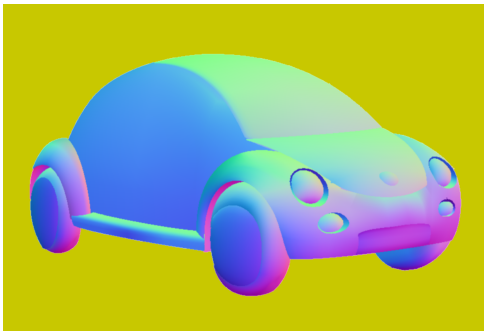
Rendering 1

Figure 7.6: Example of Mona-Lisa - This figure shows an example of painting re-interpretations: close-ups of a diffusely relit re-interpretation of Mona Lisa. Shape map, dark and light images are all painted by an artist inspired by Picasso's original painting. The artist added red lipstick, an ear and a pearl earring to the original image as an homage to Johannes Vermeer's Girl with a Pearl Earring. This particular shape map is a prime example of imperfect, incorrect, and inconsistent shape maps. Despite that, we can obtain acceptable results

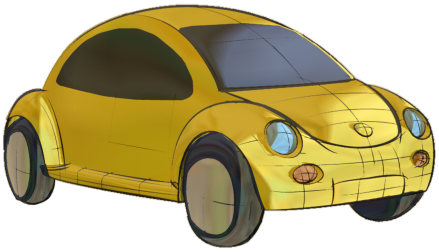


Dark Image: $DI_0(u, v)$

Light Image: $DI_1(u, v)$



Cross-Shade model



Rendering

Figure 7.7: Shadow and subtle reflection on a Cross-Shade model - This figure shows shadow and subtle reflection on a CrossShade model. Dark and light images are obtained from the rendered images in the paper. We only added yellow background to CrossShade model (i.e. normal map) to differentiate outside regions. The CrossShade model is used in permission

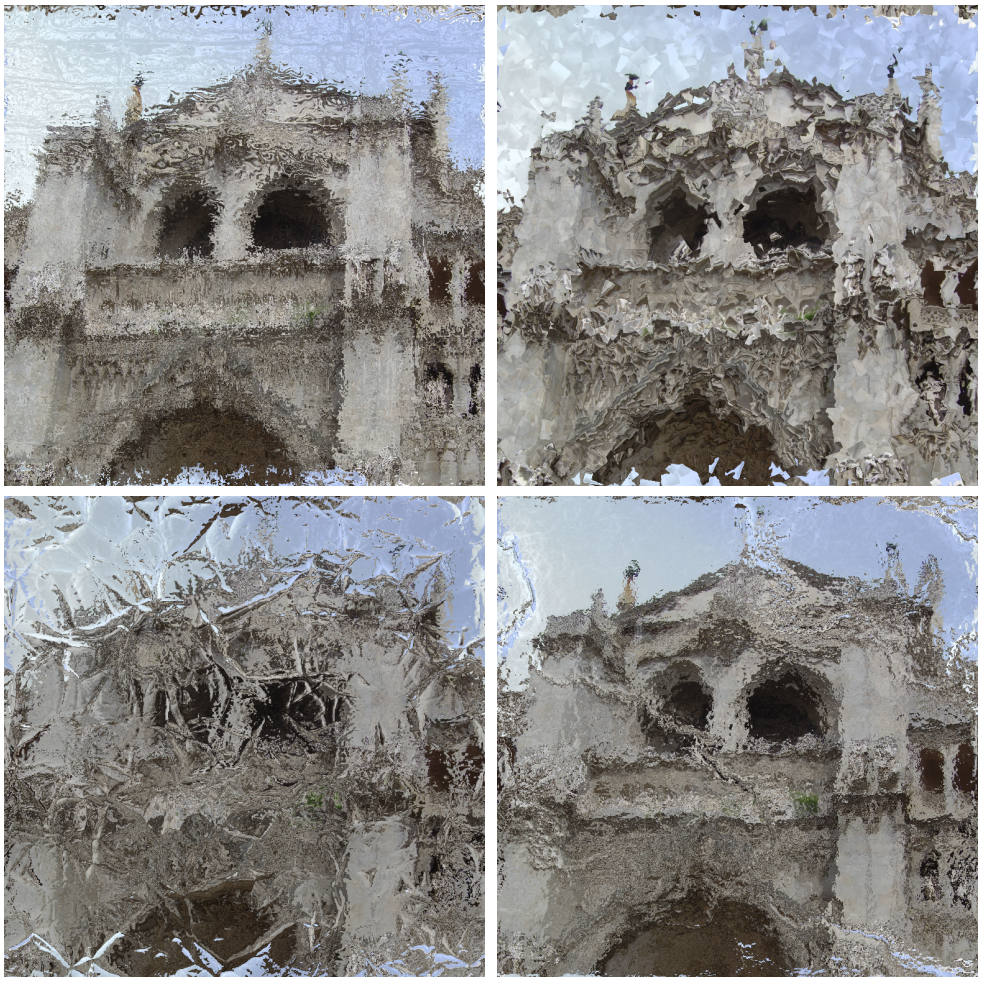
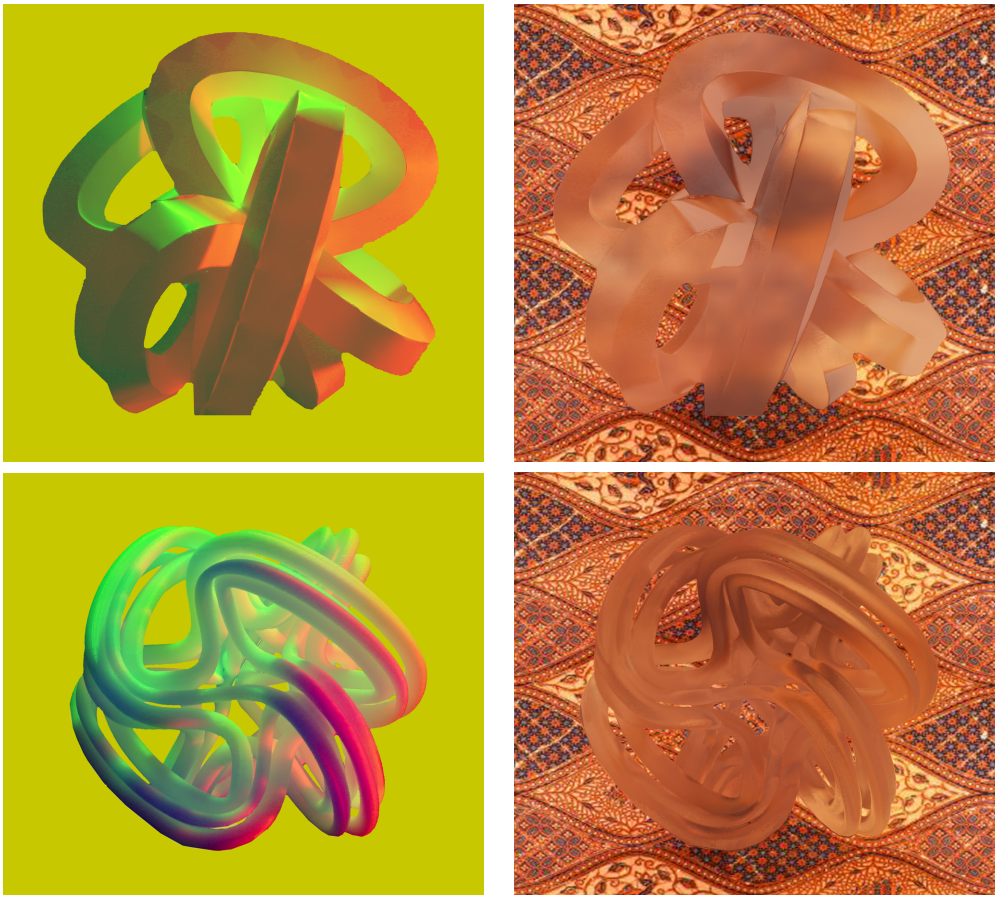


Figure 7.8: Artistic filtering effects obtained by a variety of shape maps

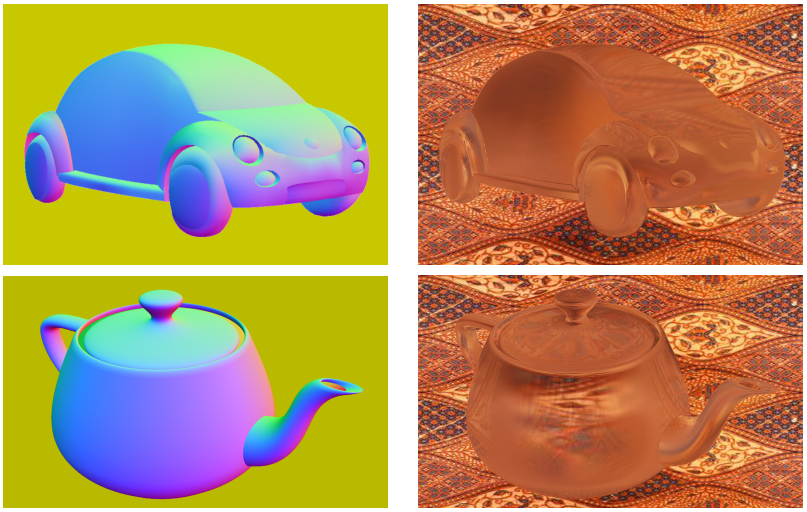


Shape Map Photos Composite

Figure 7.9: Examples of shape map photographs of diffuse objects - This figure shows examples of shape map photographs of diffuse objects. In these cases, we do not have any foreground object, i.e. $\alpha = 0$. The composite images are simply the result of refraction and refraction of the same image that is used both environment and background image. The original of genus-6 object at the top is made from paper. The high genus object in the middle is made from ABS plastic

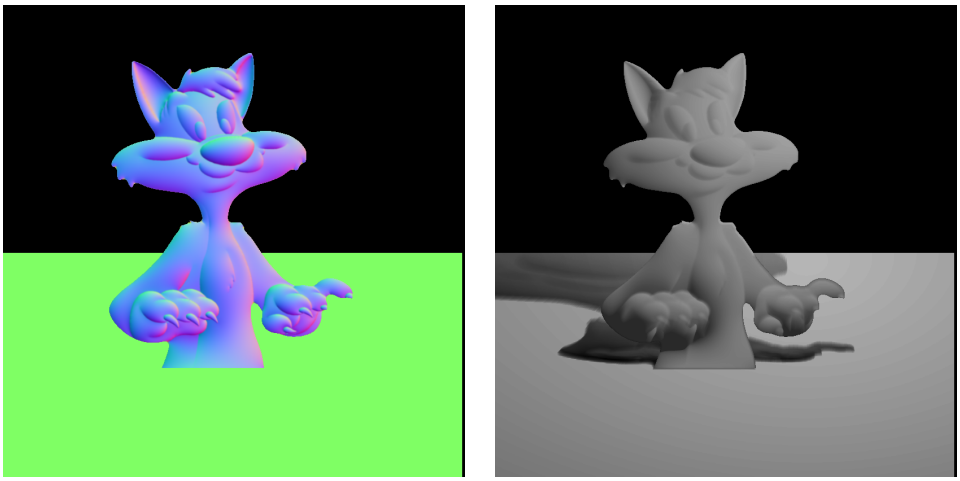
Shape Map Photo Composite

Figure 7.10: An example of shape map photograph of a translucent object - This figure shows an example of shape map photograph of a translucent object. In this case, we do not have any foreground object, i.e. $\alpha = 0$. The composite images are simply the result of refraction and refraction of the same image that is used both environment and background image



Shape Maps Composite

Figure 7.11: Reflection and refraction with Lumo and Cross-Shade models - In these cases, we do not have any foreground object, i.e. $\alpha = 0$. The composite images are simply the result of refraction and refraction of the same image that is used both environment and background image. Lumo and CrossShade normal maps are used in permission



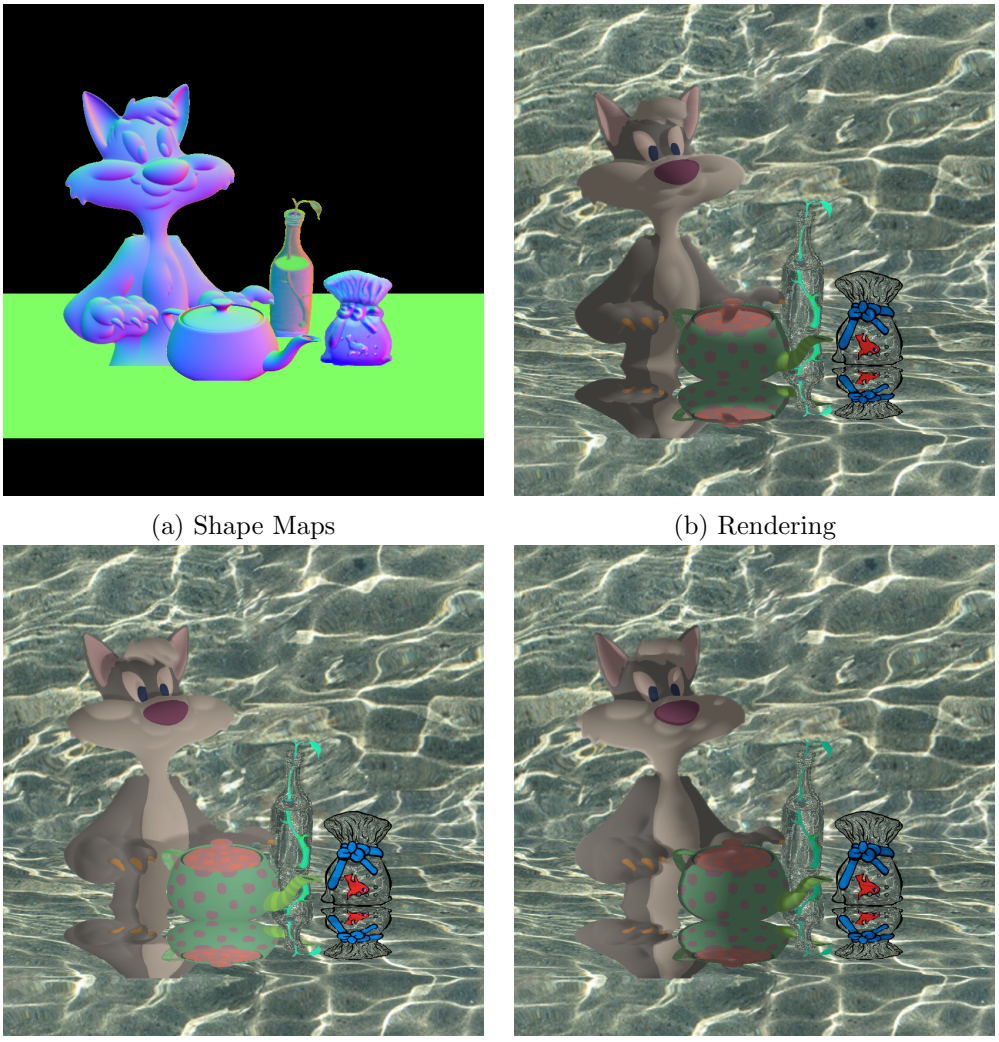
(a) Shape Maps

(b) Rendering

(c) Rendering

(d) Rendering

Figure 7.12: Rendering with Lumo models - In these cases, we showed the results using Lumo's Cat model. (a) is the original scene, and (b),(c) and (d) are renderred with lighting at different positions



(a) Shape Maps (b) Rendering

(c) Rendering (d) Rendering

Figure 7.13: Rendering with many models - In these cases, we showed different aspects of our system, we have refraction, reflection, and shadow cast from same layer and other layers. (a) is the original scene, and (b),(c) and (d) are renderred with lighting at different positions

8. CONCLUSION

In this work, we developed a system that uses shape maps to generate many different kinds of visual effects. Shape maps are particularly useful for 2D artists whose primary focus is painting and illustration. These artists have a good understanding of how reflection and refraction works, but they may not want to follow the conventional 3D rendering process for obtaining 2D works. They also want to have creative control over the results. Shape maps allows them to create paintings that can be rendered and composited exactly as they want.

REFERENCES

[1] Josef Albers. *Interaction of color: unabridged text and selected plates.* Yale University Press, 302 Temple Street, New Haven, CT, USA, 1975.

[2] H. Bezerra, B. Feijo, and L. Velho. An image-based shading pipeline for 2d animation. In *Proceedings of the XVIII Brazilian Symposium on Computer Graphics and Image Processing*, SIBGRAPI 2005, pages 307–314, 2005.

[3] Jonathan Cohen, Marc Olano, and Dinesh Manocha. Appearance-preserving simplification. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 115–122, 1998.

[4] Forrester Cole, Kevin Sanik, Doug DeCarlo, Adam Finkelstein, Thomas Funkhouser, Szymon Rusinkiewicz, and Manish Singh. How well do line drawings depict shape? *ACM Transactions on Graphics (TOG)*, 28(28), 2009.

[5] Raanan Fattal, Dani Lischinski, and Michael Werman. Gradient domain high dynamic range compression. *ACM Transactions on Graphics (TOG)*, 21(3):249–256, 2002.

[6] M. Finch, J. Snyder, and H. Hoppe. Freeform vector graphics with controlled thin-plate splines. *ACM Transactions on Graphics (TOG)*, 30:166:1–166:10, 2011.

[7] Kenneth D Forbus. Qualitative process theory. *Artificial Intelligence*, 24(1):85–168, 1984.

[8] Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. A non-photorealistic lighting model for automatic technical illustration. *ACM Transactions on Graphics (TOG)*, pages 447–452, 1998.

[9] Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. A non-photorealistic lighting model for automatic technical illustration. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, pages 447–452. ACM, 1998.

[10] Wesley Griffin, Yu Wang, David Berrios, and Marc Olano. Gpu curvature estimation on deformable meshes. In *Symposium on Interactive 3D Graphics and Games*, pages 159–166. ACM, 2011.

[11] Eitan Grinspun, P Schröder, and Mathieu Desbrun. Discrete differential geometry: an applied introduction. *ACM SIGGRAPH Course*, 7, 2006.

[12] John Hart. (2013) private conversation: According to a market research firm 3D Graphics is only 8% of the whole graphics market. 2D graphics such as vector, image and video is 90% of the graphics market.

[13] Berthold KP Horn. Hill shading and the reflectance map. *Proceedings of the IEEE*, 69(1):14–47, 1981.

[14] Ken ichi Anjyo, Shuhei Wemler, and William Baxter. Tweakable light and shade for cartoon animation. In *Symposium on Non-photorealistic Animation and Rendering*, NPAR '06, pages 133–139, 2006.

[15] Scott F. Johnston. Lumo: illumination for cel animation. In *Symposium on Non-photorealistic Animation and Rendering*, NPAR '02, pages 45–52, 2002.

[16] Gordon Kindlmann, Ross Whitaker, Tolga Tasdizen, and Torsten Moller. Curvature-based transfer functions for direct volume rendering: Methods and applications. In *Proceedings of IEEE Conference on Visualization*, pages 513–520. IEEE, 2003.

[17] Isamu Motoyoshi, Shin'ya Nishida, Lavanya Sharan, and Edward H Adelson. Image statistics and the perception of surface qualities. *Nature*, 447(7141):206–209, 2007.

[18] Diego Nehab, Szymon Rusinkiewicz, James Davis, and Ravi Ramamoorthi. Efficiently combining positions and normals for precise 3d geometry. *ACM Transactions on Graphics (TOG)*, 24(3):536–543, 2005.

[19] Makoto Okabe, Gang Zeng, Yasuyuki Matsushita, Takeo Igarashi, Long Quan, and Heung-Yeung Shum. Single-view relighting with normal map painting. *Proceedings of Pacific Graphics*, pages 27–34, 2006.

[20] Alexandrina Orzan, Adrien Bousseau, Holger Winnemoller, Pascal Barla, Joelle Thollot, and David Salesin. Diffusion curves: A vector representation for smooth-shaded images. *ACM Transactions on Graphics (TOG)*, 27(3):92:1–92:8, 2008.

[21] Paul Rademacher. View-dependent geometry. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 439–446. ACM Press/Addison-Wesley Publishing Co., 1999.

[22] T. Ritschel, T. Thormlen, C. Dachsbacher, J. Kautz, and H. Seidel. Interactive on-surface signal deformation. *ACM Transactions on Graphics (TOG)*, 29(4):36:1–36:8, 2010.

[23] Tobias Ritschel, Makoto Okabe, Thorsten Thormlen, Hans peter Seidel, and Mpi Informatik. Interactive reflection editing. *ACM Transactions on Graphics (TOG)*, 28(5):129:1–129:7, 2009.

[24] Cloud Shao, Adrien Bousseau, Alla Sheffer, and Karan Singh. Crossshade: shading concept sketches using cross-section curves. *ACM Transactions on Graphics*

(TOG), 31(4):45:1–45:11, 2012.

[25] J. Sun, L. Liang, F. Wen, and H. Shum. Image vectorization using optimized gradient meshes. *ACM Transactions on Graphics (TOG)*, 26(11):11:1–11:7, 2007.

[26] Daniel Sýkora, John Dingliana, and Steven Collins. Lazy- brush: Flexible painting tool for hand-drawn cartoons. *Computer Graphics Forum*, 28(2):599–608, 2009.

[27] Daniel Sýkora, Ladislav Kavan, Martin Čadík, Ondřej Jamriška, Alec Jacobson, Brian Whited, Maryann Simmons, and Olga Sorkine-Hornung. Ink-and-ray: Bas-relief meshes for adding global illumination effects to hand-drawn characters. *ACM Transactions on Graphics (TOG)*, 33(2):16:1–16:15, 2014.

[28] Corey Toler-Franklin, Adam Finkelstein, and Szymon Rusinkiewicz. Illustration of complex real-world objects using images with normals. In *Symposium on Non-photorealistic Animation and Rendering*, NPAR '07, pages 111–119, 2007.

[29] Romain Vergne, Pascal Barla, Roland W. Fleming, and Xavier Granier. Surface flows for image-based shading design. *ACM Transactions on Graphics (TOG)*, 31(94):94:1–94:9, 2012.

[30] Tim Weyrich, Jia Deng, Connelly Barnes, Szymon Rusinkiewicz, and Adam Finkelstein. Digital bas-relief from 3d scenes. In *ACM SIGGRAPH 2007 papers*, volume 26 of *SIGGRAPH '07*, pages 32:1–32:7, 2007.

[31] Holger Winnemöller, Alexandrina Orzan, Laurence Boissieux, and Joëlle Thollot. Texture design and draping in 2d images. In *Computer Graphics Forum*, volume 28, pages 1091–1099. Wiley Online Library, 2009.

[32] T. Wu, C. Tang, M. Brown, and H. Shum. Shapepalettes: Interactive normal transfer via sketching. *ACM Transactions on Graphics (TOG)*, 26(3):44:1–44:5,

2007.

[33] Douglas E Zongker, Dawn M Werner, Brian Curless, and David H Salesin. Environment matting and compositing. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 205–214, 1999.