AN INTEGRATED MODEL FOR DIRECT ILLUMINATION, SUBSURFACE SCATTERING, SPECULAR HIGHLIGHTS AND SHADOW COMPUTATION

A Thesis

by

FERMI NIVEDITHA PRAMANANDA PERUMAL

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

MASTER OF SCIENCE

August 2017

Major Subject: Visualization

ABSTRACT

In this thesis, we present an integrated model to compute direct illumination, subsurface scattering, specular highlights and soft shadow effects in a single equation. Using this model, it is possible to obtain various effects in a qualitatively consistent way even with a single light. The model is tested for rendering bas-reliefs, i.e. height fields under both single light and environment illumination. The model is implemented as a web-based program using WebGL. Using our system, users can see rendered results in real time even while modifying shader parameters, because of GPU rendering and computational simplicity of our integrated model.

ACKNOWLEDGMENTS

I would like to thank my chair Professor Ergun Akleman for giving me continuous support and motivation to finish my thesis. I am grateful for the trust he invested in me and for all the helpful feedback he provided throughout the course of my work. I would like to thank my committee members Professor Richard Davison and Professor Rodney Hill for their encouragement and guidance that helped me look at my work from different perspectives. Also, I would like to thank the faculties and staff members of the Visualization department who helped me through each and every step along the way.

I would like to extend my thanks to my friends of the Viz lab who helped me and inspired me to work harder. I would like to acknowledge and extend my heartfelt gratitude to my family: my father Pramananda Perumal, my mother Jesintha Mary and my brother Jugal for being my pillars of support throughout my thesis. Finally, I would like to thank my boyfriend, Sharan for being there for me during this whole process. I could not have asked for a better partner to get me through the challenging times and completing my thesis work. This work wouldn't have been possible without all their love and confidence in me. Thanks for making this happen.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supervised and supported by a thesis committee consisting of Professor Ergun Akleman and Professor Richard Davison of the Department of Visualization and Professor Rodney Hill of the Department of Architecture. All work for this thesis was completed independently by the student.

Funding Sources

There are no outside funding contributions to acknowledge related to the research and compilation of this document.

NOMENCLATURE

2D	Two Dimensional
3D	Three Dimensional
CG	Computer Graphics
NPR	Non-Photorealistic Rendering
RGBN	Red, Green, Blue and Normal
SFS	Shape From Shading
BSDF	Bidirectional Scatter Distribution Function
BRDF	Bidirectional Reflectance Distribution Function
BTDF	Bidirectional Transmittance Distribution Function
BSSRDF	Bidirectional Surface Scattering Reflectance Distribution Function

TABLE OF CONTENTS

LIST OF FIGURES

1. INTRODUCTION AND MOTIVATION

1.1 Introduction

This thesis presents an integrated model for qualitatively consistent rendering of height fields that represent bas-reliefs. Various effects such as $\cos\Theta$, reflections, subsurface scattering and soft shadows are computed using an integrated technique that provides qualitatively similar results. The system is implemented in a web environment that supports real-time computations. This model also provides artistic control over these effects. Using interactive lighting and artistic controls, users can tailor the rendering result based on their artistic requirements.

1.2 Motivation

During the last several decades many models for rendering have been developed to emulate illumination effects such as direct illumination, shadows and subsurface scattering. However, these models are not really consistent with each other since each model relies on a different type of simplification. Because of this inconsistency, it is hard to combine these effects for obtaining qualitatively consistent results. In practice, this problem is covered by using more and more lights. This is not really recommended since it is better to use a few lights as interface controls to obtain desired results. There is, therefore, a need for a simple method that can produce qualitatively consistent results for a wide variety of effects. This works demonstrates the power of a new model that produces results qualitatively similar to $\cos\Theta$ computations and provides better control over them even with one light.

2. LITERATURE REVIEW

2.1 Normal Maps

Normal maps are images that are generally used to define smaller details on top of low polygon. As per Johnston [1], normal maps can also be used to identify the shape of the object through each pixel in the 2D image for a cel-based animation scene. This could help us create shadows in the image as if it were in a 3D environment. The goal of the technique is not accuracy according to Johnston, but convincing illumination over surfaces that change over time in a live action scene. This method would avoid the time-consuming task of creating 3D geometry for all objects in the animation scene. Building on Johnston's work, Bezerra, Feijo and Velho [2] propose an image-based technique that would handle the environment reflections on a 2D surface in a scene. Bezerra et al. suggests that their approach is more intuitive than Johnston's approach because they use the 2D image as is without vector conversion, and that the vectorization of the image as suggested by Johnston would be difficult for artists to use. While Johnston's work identify the normal maps using the edge information and gradients, Shao, Bousseau, Sheffer and Singh [3] create normal maps by using cross sections of the 2D image. The cross section network drawn by the artist in a concept art can be used to detect the normals with the help of perception calculations. Even though this tool is very effective in converting concept sketches by artists into 3D designs, this only works if the concept sketch is non-orthogonal. If the concept sketch is drawn orthogonally, Shao et al. algorithm might give wrongly deformed projections. Another technique suggested by Okabe, Zeng, Matsushita, Igarashi, Quan and Shum [4] make use of normal maps in photography. Okabe et al. allows the artist to paint normal maps over a photograph that will be given as an input to their algorithm to manipulate lighting in each pixel. Although these approaches help in creating a 3D looking environment, they compute the 3D shading components separately in an inconsistent way and combine them to create a consistent look. Also, subsurface scattering and soft shadows aren't addressed in these solutions.

2.2 Normal Maps with Diffuse and Shape

Toler-Franklin, Finkelstein and Rusinkiewicz [5] make use of a datatype that falls in between 2D RGB (Red, Blue and Green) image and a full 3D mesh, called RGBN (Red, Blue, Green and Normal). Each pixel in RGBN has information on both the color and surface normal that are leveraged to create non-photorealistic illustrations. This technique allows multiple objects in a 2D scene, because color and normal information is stored per pixel basis. Stylized effects like toon shading and line drawing is possible in this technique, however, if the image is dark or highly specular, it is not possible to obtain proper normal information. Moreover cast shadows are not possible because the depth information is not present. Wang [6] elaborates a technique where he combines normal maps with thickness maps and displacement maps. The combination is called shape maps that can be used to create "fuzzy geometry". Fuzzy geometry allows the creation of impossible shapes in the 2D scene. There are different methods in which the input image could be provided to this framework. Artists can directly create shape maps using 2D tools like Photoshop or upload a photograph. An intuitive approach is used for creating shape maps. The artist has to imagine a parallel directional red light from the left and a parallel green light from the top. Based on the 3D object, the artist paints how much of red and green light falls on each pixel. The framework doesn't require all three maps to create a fake 3D scene, so it works even if some of the maps are not provided. Unlike Toler-Franklin et al.'s approach, the scene could also be layered as billboards or just a single layer could be used. Xiong [7] enhanced the shape mapping technique by creating a web-based application that provides artistic controls for lights, reflection, refractions and ambient occlusion. The tool also allows artists to render impossible shapes in real-time, which cannot be modeled in 3D. Although these approaches uses cos Θ, the above two techniques do not handle consistent shadow calculation and subsurface scattering in the scene.

2.3 Bas Reliefs

Bas relief is another approach used to create illuminated 3D looking surfaces. Bas reliefs can be found in ancient wall sculptures and coins. Wu, Martin, Rosin, Sun, Lai, Liu and Wallraven [8] propose a method where the 3D shape of a 2D image can be extracted by using photometric stereo. The image is lit under different lighting conditions and then rendered using NPR (non-photorealistic rendering). Sỳkora, Kavan, Čadík, Jamriška, Jacobson, Whited, Simmons and Sorkine-Hornung [9] creates a framework called ink and ray, that automatically generates approximate 3D geometry from 2D images, just enough to create global illumination effects and consistent lighting in the scene. This geometry will give the impression of 3D in the scene. The ink and ray approach makes approximations of the image depth when making the proxy 3D mesh. This might be a limitation because such unreal approximations could sometimes be smaller or larger than the actual desired depth, because depth information is extracted from only one image given as input from one angle. While the approaches suggested by Wu et al and Sỳkora et al. address self-shadowing and reflections, these techniques are only applicable for human faces and character portraits.

2.4 Shadow Computation

Wu, Martin, Rosin, Sun, Langbein, Lai, Marshall and Liu [10] present a Shape from Shading (SFS) method for creating bas-reliefs for human portraits, which makes bas-reliefs from the shading of the image, given a known set of light direction and material behavior. While the enhanced method helps suppress smaller details and emphasizes the important features of a face, lot of specular information is lost in the process. While Wu et al. provides a bas relief solution for human faces, [9] present the use of bas relief in creating 3D looking images with just one input image, eliminating the necessity of creating multiple images as suggested by Wu et al. Governi, L., Carfagni, M., Furferi, R., Puggelli, L., and Volpe, Y. [11] suggests a SFS algorithm that constructs digital bas-reliefs from a single image. The method recreates the bas-reliefs with user provided boundary conditions as initialization and then running a SFS-based reconstruction algorithm. The main limitation that this method runs into is an over-smoothing effect.

2.5 Other Methods

A 2D scene could also be imagined to be an elastic cloth, where the artists could fix points on the cloth and elongate/contract the cloth in other regions, to define the surface signals, as suggested by Ritschel, Thormählen, Dachsbacher, Kautz and Seidel [12]. Ritschel et al. provides an interactive framework where artists can achieve artistic results even though they are physically accurate. 3D shading components like soft shadows and indirect illumination that are not addressed by Toler-Franklin et al. are solved in the approach by Ritschel et al.

3. COMPUTATION METHODOLOGY

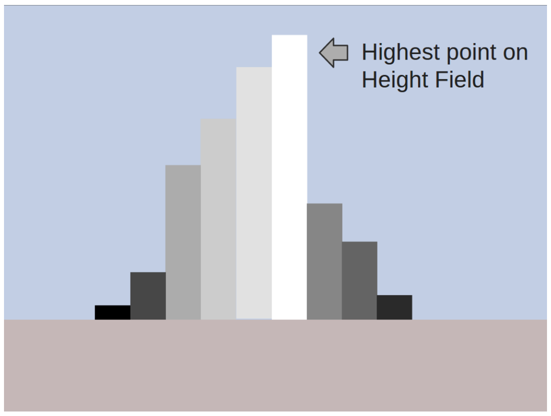
Shading is fundamental in the process of conveying a three-dimensional structure for an object in a two dimensional image [13]. Over the years, many methods have been suggested for rendering diffuse, shadows and scattering with traditional and distributed ray tracing techniques. They use 3D scene information like normals, object boundaries and material properties to render realistic and non-realistic images. While a lot of techniques use calculations that are physically accurate, we present an integrated system that produces results that are much simpler to render. This model computes diffuse, soft shadows and subsurface scattering with height fields that look qualitatively consistent with $\cos\theta$ shading.

3.1 Height Fields

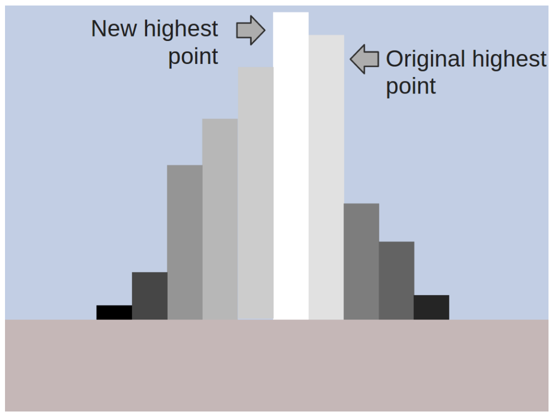
Height fields are images that store height information of the scene (See Figure 3.2). A height map holds values at every point in the range [0, 1], 1 being the highest point and 0 being the lowest point in the scene. Unlike normal maps, height maps are relative i.e. adding a higher point in the scene would change the values of all other points in the map. Height fields can be generated from normal maps by computing the relative height at each point. Apart from this method, 3D manipulation tools can be used to calculate height maps in a scene.

3.1.1 Visualization of Height Fields

To better understand height fields, we can visualize a height field in 1-Dimension as shown in Figure 3.1 a. The color smoothly transitions from bright to dark from the tallest to the shortest point. The tallest point is indicated using the arrow. Now in Figure 3.1 b, we can see that a new height point is added that is taller than the existing points. Now, that



(a) Height field.



(b) New height point added to Height field.

Figure 3.1: Visualization of Height Fields in 1-Dimension.

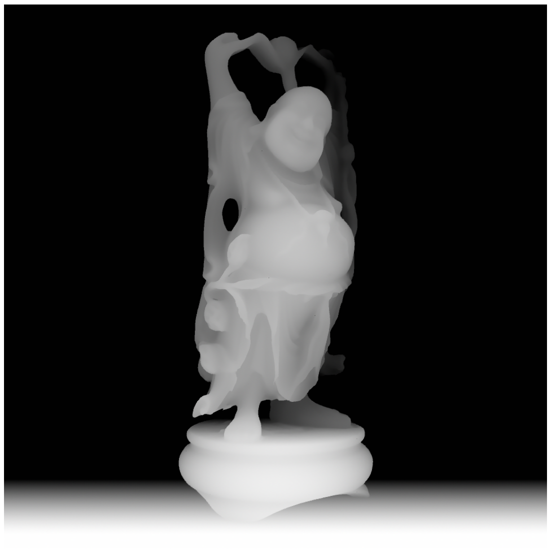
becomes the brightest point and the shorter points scale themselves between the white and black color range smoothly.

3.1.2 Height Fields from 3D scenes

Using 3D manipulation tools like Maya, height fields can be generated that are more accurate than the height fields created using normal maps method explained above. This method results in more accurate height fields because the actual 3D model is used. The models of Buddha, Lucy and Dragon have been downloaded from Stanford Computer Graphics Laboratory [14]. The height fields shown in Figure 3.2) have been generated using Maya. We created a simple depth shader adapting an example from Autodesk Maya Help [15] that stores the height or z-depth information at each point and normalize it later based on the highest and lowest z-depth values. More examples of height fields generated using this method is shown in Figure 3.3)

3.1.3 Height Fields from Normal Maps

Height fields can be generated from Normal Maps. The relative height at a point can be measured by taking its derivative. For calculating the relative height at point $\mathbf{p}_N$, we


(a) Buddha.

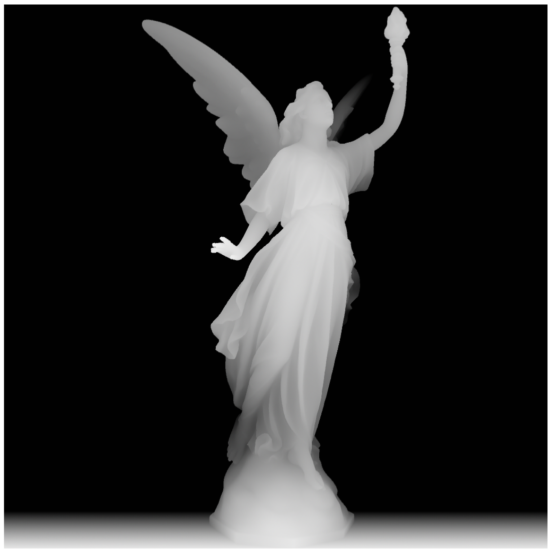
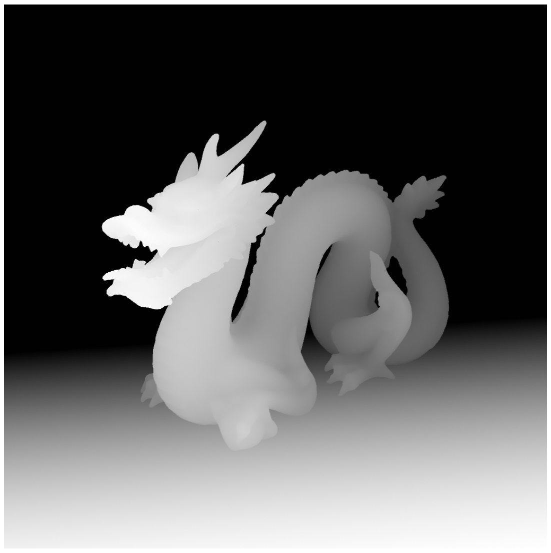

(b) Lucy.

Figure 3.2: Examples of Generated Height Maps using 3D tool.

travel from point $\mathbf{p}_0$ towards $\mathbf{p}_N$ and calculate the change in height that happens along the way. Since the path to $\mathbf{p}_N$ differs based on the starting point, a height field calculated from $\mathbf{p}_0$ will be different from a height field calculated from $\mathbf{p}'_0$. This method generates successful results when using simpler objects but fails when using complex scenes. Since height field generation is not the research problem this thesis is focusing on, this could be an opportunity for research in future work.

3.2 Scattering

Bartell, Dereniak, and Wolfe [16] introduce BSDF (Bidirectional Scattering Distribution Function) as a generalized function that defines how light is scattered by a surface. This function can be split into a reflected component, BRDF (Bidirectional Reflectance Distribution Function) and a transmitted component, BTDF (Bidirectional Transmittance Distribution Function). In this thesis, we present an integrated function that takes a similar approach for calculating diffuse, scattering and reflectance. Figure 3.4 shows how light is


(a) Dragon.

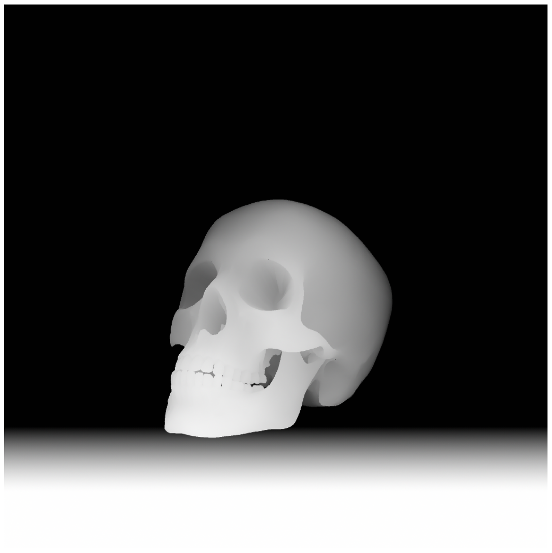

(b) Skull.

Figure 3.3: More examples of Generated Height Maps.

scattered and transmitted on a surface.

3.2.1 Diffuse Scattering

As shown in 3.5 b, diffuse scattering is when light gets reflected in all directions from the surface. Diffuse can be measured by calculating the amount of illumination or diffuse scattering $\mathbf{d}$.

3.2.1.1 Model for Computing Illumination

According to our model, the amount of illumination at a point is inversely proportional to the amount of light being blocked by other points in the scene. So, the diffuse or illumination factor $\mathbf{d}$ can be calculated by measuring the length of the light ray traveling above the height field. A higher point will have less hindrance so it will have a higher diffuse factor, which means that the peak will have full illumination. As shown in Figure 3.6, point A is a peak and point B is a dip. The points from region C to B will have smooth gradient of increasing illumination.

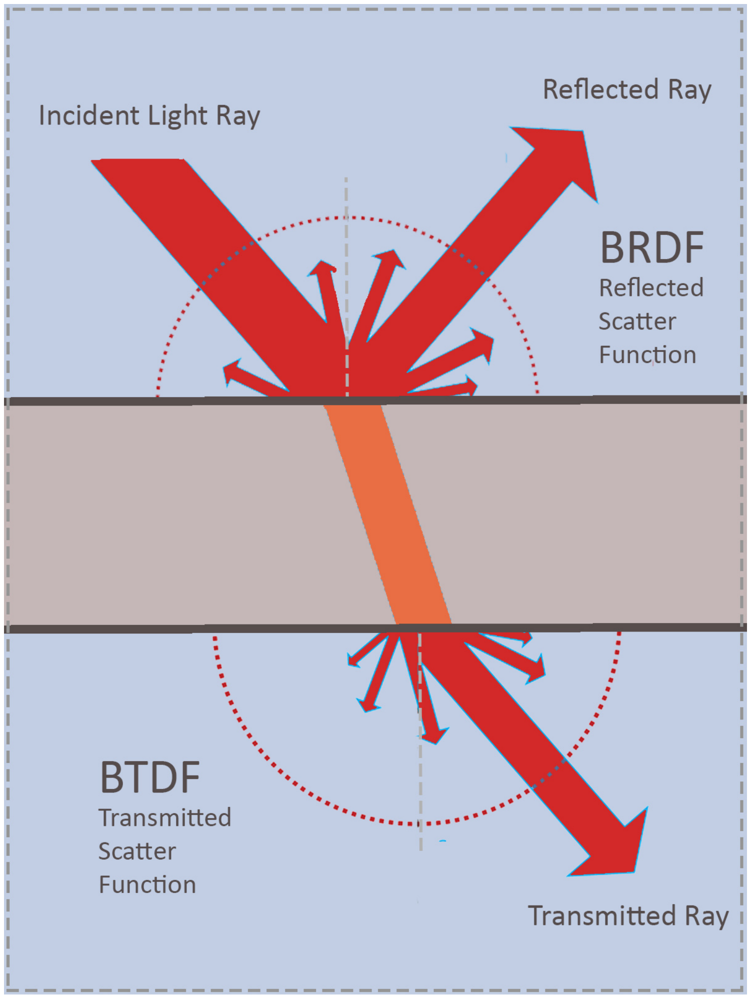
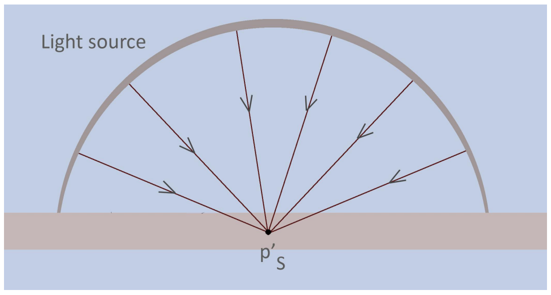
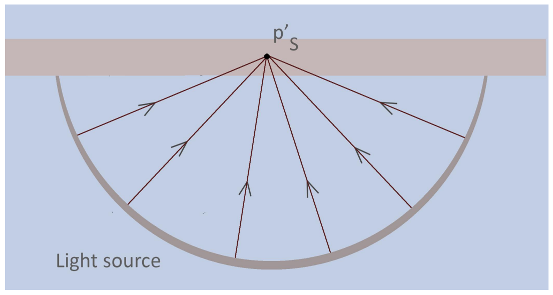


Figure 3.4: BSDF Function.

Let $\mathbf{p}_L$ denote the position of a point light and, let $\mathbf{p}_S$ and $\vec{n}_S$ denote the position and the normal vector of a given shading point on a surface. We define a new shading position under the surface as $\mathbf{p}'_S = \mathbf{p}_S - t\vec{n}_S$. Let $L(\mathbf{p}_L, \mathbf{p}'_S)$ denote the line segment that starts from $\mathbf{p}'_S$ and ends at $\mathbf{p}_L$ and let r denote the length of $L(\mathbf{p}_L, \mathbf{p}'_S)$ goes through inside of the shapes. Then, the contribution of the illumination is computed simply as $d = r/t$ (See Figure 3.7).



(a) $\cos\theta$.



(b) Diffuse Scatter.

Figure 3.5: $\cos\theta$ and diffuse scatter calculated using one integrated equation.

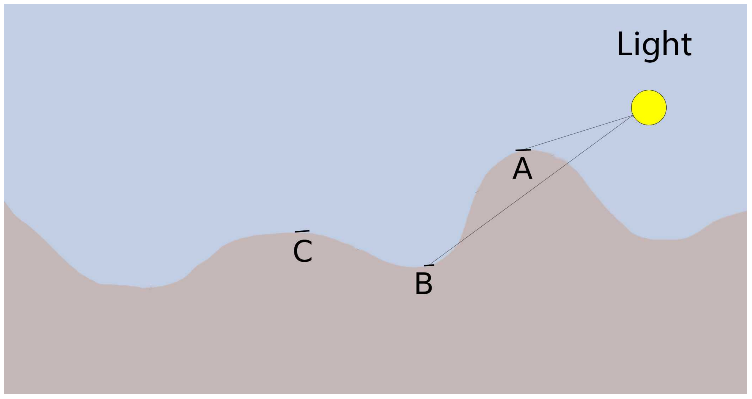
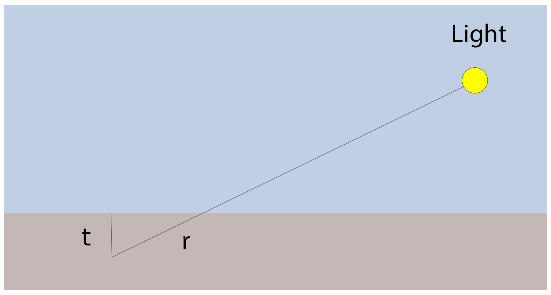


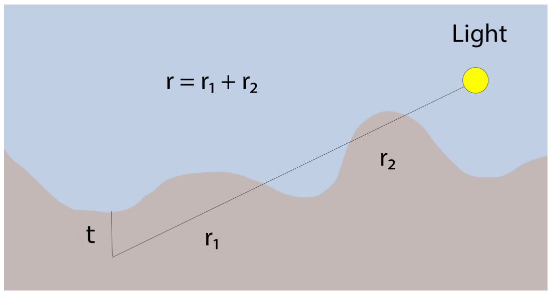
Figure 3.6: Illumination at different regions of Height field.

3.2.1.2 Qualitative Similarity with $\cos\theta$

This model works because it provides results qualitatively similar to existing shading models. Even though the computations in this model are different from traditional techniques, they provide qualitatively consistent visual results. We base this on qualitative similarity concept introduced by Forbus et al [17]. 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 $y = f(x)$. One advan-



(a) $\cos\Theta$.



(b) Subsurface Shadow.

Figure 3.7: Integrated Computation of the direct illumination and shadow.

tage of this formulation, there is no need to explicitly derive the monotonically increasing function f. This has been explained in detail in Wang's thesis [6].

Let the shading surface be a plane, then it is easy to demonstrate that $d = r/t$ gives us $\cos\theta$ regardless of dt (See Figure 3.7a). Note that the equation is valid only for $(\mathbf{p}_L - \mathbf{p'}_S) \cdot \vec{n}_S > t$ where $\cdot$ denotes dot product since the light cannot be inside of the object. Moreover, even if the surface is not planar, when $\lim t \to 0$, it still provides $\cos\theta$.

3.2.1.3 Qualitative Similarity with shadows

Shadow is the one that produces inconsistencies. In computer graphics we have two different sources of shadow: (1) Self-shadows of convex shapes that are identified as $\cos\theta < 0$ regions; and (2) other shadows that are identified by occlusion of the light by other objects and other parts of the same but non-convex object. Note that our new model actually combine both cases in one equation as shown in Figure 3.7b where r_1 is qualitatively similar the first type shadows and r_2 is qualitatively similar to second type of shadows and $r = r_1 + r_2$ logically combines both.

3.2.1.4 Illumination Computation Algorithm

The algorithm defined here will provide the illumination at a shading point $\mathbf{p'}_S$ using the height field information at that point and all points along the length of the line segment

$L(\mathbf{p}_L, \mathbf{p'}_S)$.

Algorithm 1 Algorithm to calculate the illumination at a point

```
Data: Light position $\mathbf{p}_L$, Shading position $\mathbf{p'}_S$
Result: Illumination or Diffuse factor $\mathbf{d}$
$\mathbf{t} \longleftarrow 1$
foreach point $\mathbf{p}$ on line segment $L(\mathbf{p}_L, \mathbf{p'}_S)$ such that $\mathbf{p'}_S \leq \mathbf{p} \leq \mathbf{p}_L$ do
    if $Height(\mathbf{p'}_S) < Height(\mathbf{p})$ then
        $\mathbf{t} \longleftarrow \mathbf{t} + 1$
    end
end
$\mathbf{d} \longleftarrow \frac{1}{\mathbf{t}}$
return $\mathbf{d}$
```

The color $\mathbf{c}_S$ at position $\mathbf{p'}_S$ will be then determined as,

$$\mathbf{c}_S = \mathbf{d}\ \mathbf{c}_D\ \mathbf{c}_L$$

where $\mathbf{c}_D$ is the diffuse color and $\mathbf{c}_L$ is the color of light.

3.2.2 Diffuse using Environment Lighting

Instead of using just a single lighting source, an environment sphere can also be used to create bas-reliefs (See Figure 3.8). The algorithm above can be used to imitate an environment lighting model. Either an array of colors could be manually provided or an image could be mapped in a sphere to imitate a realistic lighting setup.

As shown in Figure 3.9, a 2D texture map position $\mathbf{t}_{map}(\mathbf{u}, \mathbf{v})$ can be mapped on to a

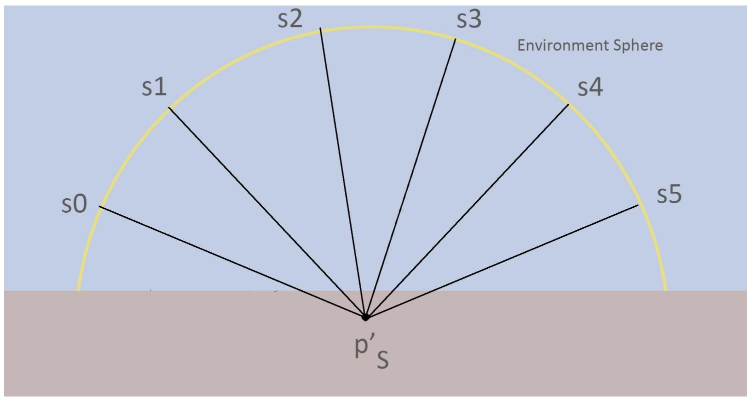


Figure 3.8: Illumination with an Environment Lighting setup.

sphere position $\mathbf{p'}_L(\mathbf{x}, \mathbf{y}, \mathbf{z})$ using its parametric coordinates,

$$\mathbf{x} = \mathbf{r} \cos\theta \sin\phi$$

$$\mathbf{y} = \mathbf{r} \sin\theta \sin\phi$$

$$\mathbf{z} = \mathbf{r} \cos\phi$$

where $\mathbf{r}$ is the radius of the sphere, θ is the angle from the Z axis between $0 \leq \theta \leq \pi$ and ϕ is the angle from the X axis between $0 \leq \phi \leq 2\pi$. The values of θ and ϕ are calculated as,

$$\theta = \pi\mathbf{u} + \theta_{\mathbf{offset}}$$

$$\phi = 2\pi\mathbf{v} + \phi_{\mathbf{offset}}$$

where $\theta_{\mathbf{offset}}$ and $\phi_{\mathbf{offset}}$ are offset values that determine the rotation of the sphere.

Each of the new light position $\mathbf{p'}_L$ calculated using the spherical coordinates will now be considered as a separate single light source and illumination at that point is calculated using Algorithm 1.

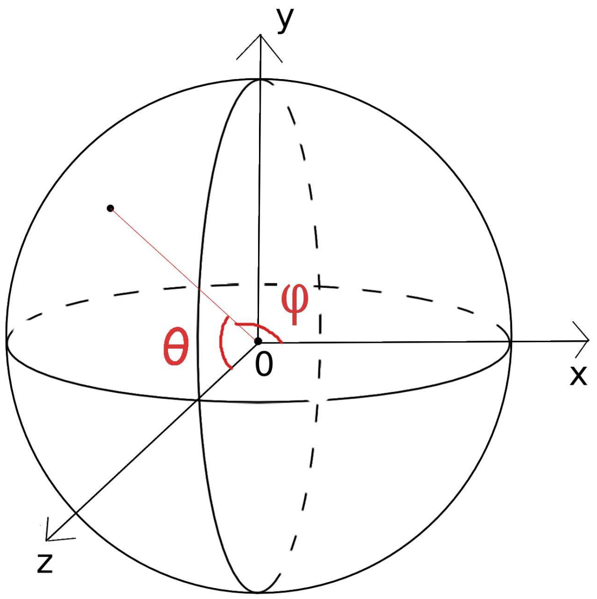


Figure 3.9: Texture mapping onto a sphere.

$$\mathbf{c}_S = \sum\nolimits_{(u_i,v_i)\in t_{map}} \mathbf{d}\,\mathbf{c}_D\,\mathbf{c}'_L(u_i,v_i)$$

where $\mathbf{c}'_L(u_i, v_i)$ is the light color, $\mathbf{c}_D$ is the diffuse color and $\mathbf{d}$ is the illumination at point $\mathbf{p}'_L$.

The color $\mathbf{c}'_S$ at shading position $\mathbf{p}'_S$ will be then determined using,

$$\mathbf{c}'_S = \frac{\mathbf{c}_S}{s}$$

where s is the number of light samples from the environment map.

3.3 Base Color and Light Color

In the above sections, we introduced the algorithm for illumination using a single and environment light source and how the final color is then determined using the illumination factor. In addition to the light colors, base material properties such as color can also be set that will be blended with the light based on the illumination factor. Without a base color, the less illuminated regions will be assumed to be black while the brighter regions will be affected by the light color. Let the light color be $\mathbf{c}_L$, diffuse color be $\mathbf{c}_D$ and base color be $\mathbf{c}_B$. Then, the color is calculated as,

$$\mathbf{c}_S = \mathbf{c}_B + \mathbf{d}\,\mathbf{c}_D\,\mathbf{c}_L$$

3.3.1 Subsurface Scattering

Subsurface scattering is the light transport within a translucent object where the light penetrates the object and interacts with the material. To render translucent surfaces, Jensen's BSSRDF (Bidirectional Subsurface Scattering Distribution Function) model [18] uses photon maps to simulate subsurface scattering. The light is distributed uniformly in all directions using photons in a distributed ray tracer. Even though the results are physically accurate, the render time becomes extremely slow.

The algorithm explained here uses BSDF (Bidirectional Scattering Distribution Function) for calculating subsurface scattering. As long as the results are qualitatively consistent, physically accurate calculations can be replaced with simpler methods that simulate subsurface scatter. As shown in Figure 3.7 b, the shading position $\mathbf{p'}_S$ is computed in a way that it is under the surface of the actual height field. This gives an effect as though the light is traveling inside the object. Upon further research, this method can produce even more accurate results with more ways to control it.

3.3.1.1 Forward Scattering

As shown in Figure 3.10, light travels all the way through the skin and exits on the other side. Forward scattering is important in simulating subsurface scattering and this effect is implemented in our model. This effect is visible when light is behind the object. The scattering factor would be the cosine of the angle between the light ray and eye direction. So, as light moves away, the effect of the scattered light becomes lesser and lesser and there is a smooth falloff. This is demonstrated in Figure 3.11 a and 3.11 b. This illustrations shows that when light moves in front of the object, then $\cos\phi$ becomes negative and hence no forward scattering is visible.

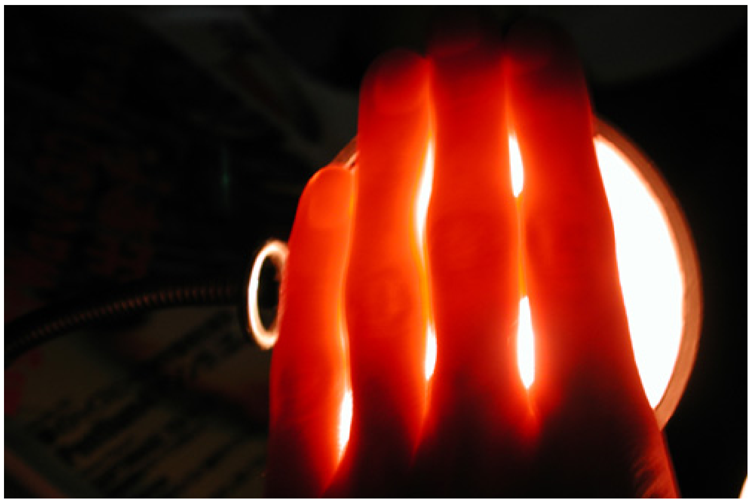


Figure 3.10: Subsurface scattering effect in human skin.

Let a light ray from shading position $\mathbf{p}'_S$ to light $\mathbf{p}_L$ be $\vec{l}_S$. It is calculated as,

$$\vec{l}_S = \mathbf{p}_L - \mathbf{p}'_S$$

$$\hat{l}_S = \frac{\vec{l}_S}{|\vec{l}_S|}$$

Let the eye direction from eye position $\mathbf{p}_E$ to shading position $\mathbf{p'}_S$ as shown in 3.11 be $\vec{v}_S$,

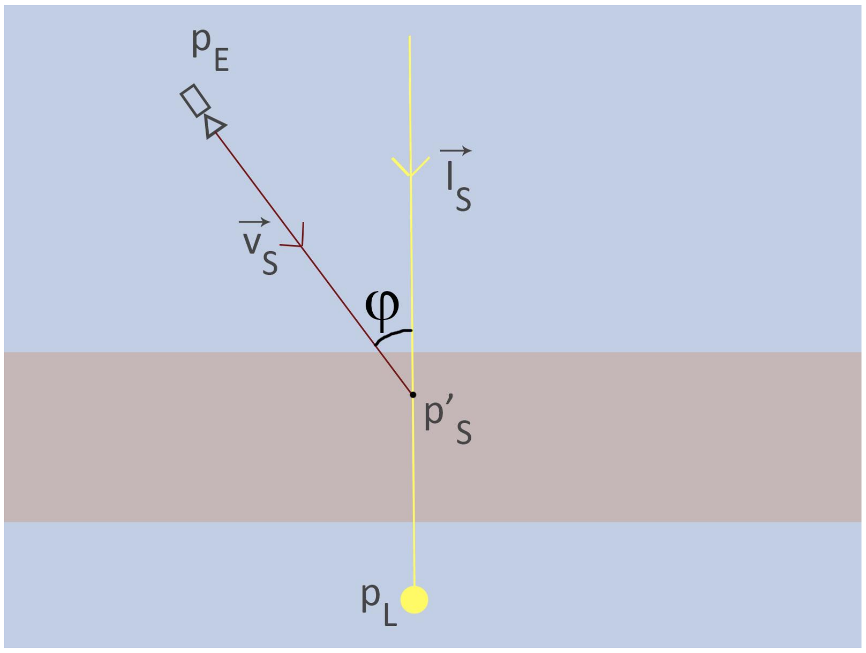
$$\vec{v}_S = \mathbf{p'}_S - \mathbf{p}_E$$

$$\hat{v}_S = \frac{\vec{v}_S}{|\vec{v}_S|}$$

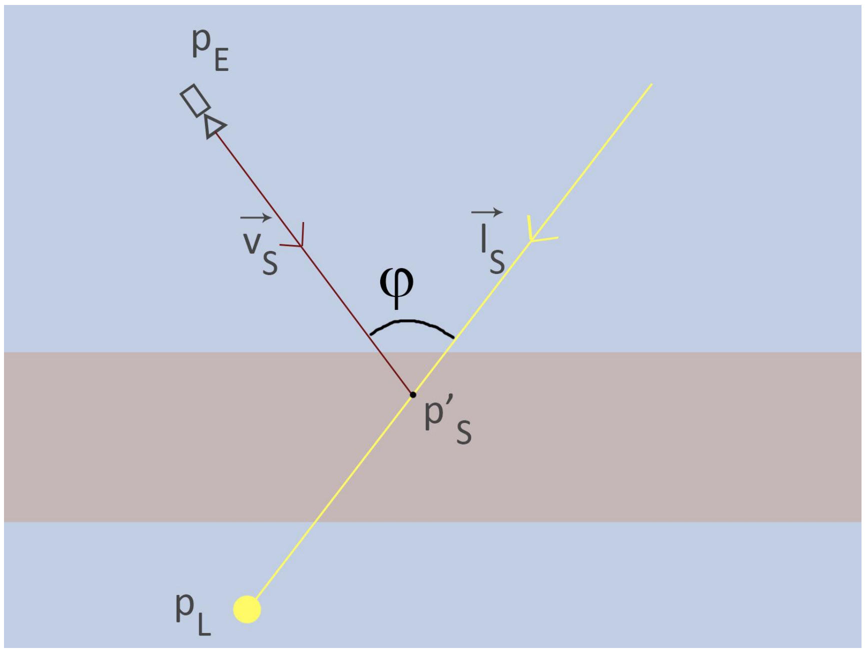
So, the scattering factor $\mathbf{f}$ is calculated as,

$$\mathbf{f} = F(\hat{v}_S \cdot \hat{l}_S)$$

where $\hat{v}_S$ is the eye direction, $\hat{l}_S$ is the light ray and function F() can be used to control the cosine curve using one or more controls.



(a) Calculating scatter factor.



(b) Scatter factor decreases as ϕ increases.

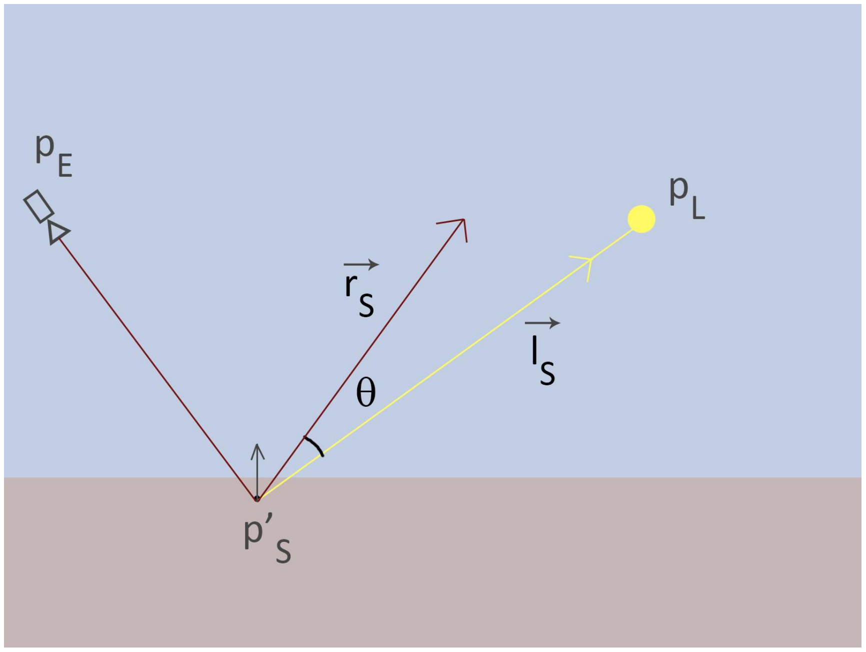
Figure 3.11: Calculating forward scattering.

3.3.1.2 Qualitative Similarity with Subsurface Scattering

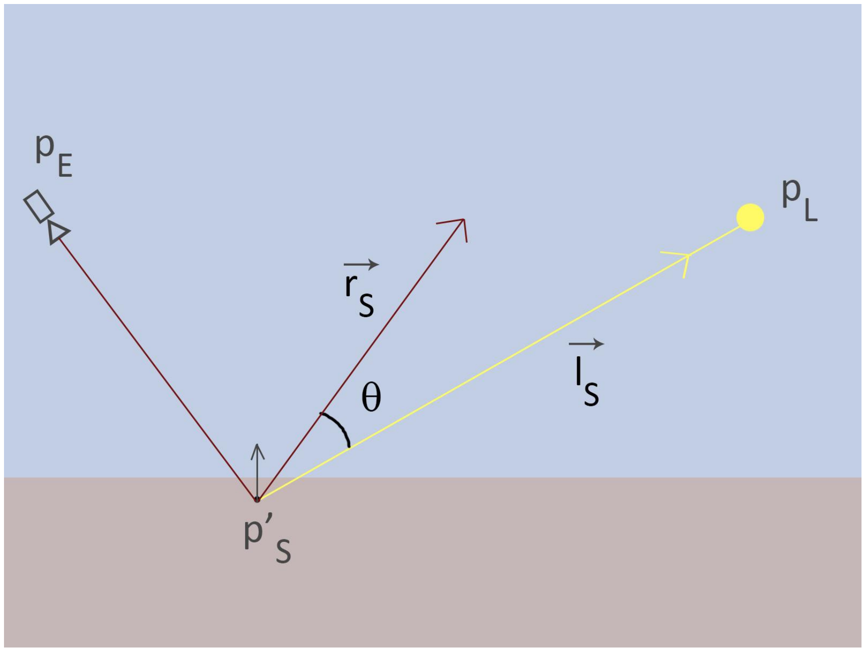
Exponential attenuation due to scattering along a path can be given as e^{-ax} where a is a positive real based on subsurface scattering property of the object material and x is the total length of the path inside of the object. The total scattering effect can be computed as an integral of exponential attenuation of all possible paths between the two points $L(\mathbf{p}_L$ and $\mathbf{p}_S)$. Without loss of generality, we can assume that the total scattering is qualitatively similar e^{-ar} where r is the length of $L(\mathbf{p}_L, \mathbf{p'}_S)$ goes through inside of the shapes as before. Based on this assumption, it is easy to show that e^{-ar} and t/r are qualitatively similar for all positive a values.

3.4 Specular Highlights

Reflection is the return of light back into the same medium after striking a boundary between two mediums. Reflection happens based the object's reflective properties. As per law of reflection for specularity, the angle of incidence of a light ray is equal to the angle of reflection. The shininess of an object depends on its material properties. A shiny object like a ball bearing is more reflective than a piece of cloth. The reflected ray is calculated as $\vec{r} = \vec{v} + 2(\hat{n} \cdot \vec{v})\hat{n}$ where $\vec{v}$ is the incident ray and $\hat{n}$ is the surface normal.



(a) Calculating specular factor.



(b) Specularity factor decreases as θ increases.

Figure 3.12: Calculating specularity.

Let the incident ray from eye position $\mathbf{p}_E$ to shading position $\mathbf{p'}_S$ be $\vec{v}_S$.

$$\vec{v}_S = \mathbf{p'}_S - \mathbf{p}_E$$

The surface normal $\vec{n}_S$ at a shading point can be calculated approximately from the height field information by finding the difference between the heights of the previous and next shading points.

$$\vec{n}_S = Height(\mathbf{p'}_{S-1}) - Height(\mathbf{p'}_{S+1})$$

The normalized surface normal $\hat{n}_S$ is,

$$\hat{n}_S = \frac{\vec{n}_S}{|\vec{n}_S|}$$

The reflected ray $\vec{r}_S$ is calculated with $\vec{v}_S$ and $\hat{n}_S$ as,

$$\vec{r}_S = -\vec{v}_S + 2(\hat{n}_S \cdot \vec{v}_S)\hat{n}_S$$

Let a light ray from shading position $\mathbf{p'}_S$ to light $\mathbf{p}_L$ be $\vec{l}_S$. It is calculated as,

$$\vec{l}_S = \mathbf{p}_L - \mathbf{p'}_S$$

$$\hat{l}_S = \frac{\vec{l}_S}{|\vec{l}_S|}$$

With reflected ray $\vec{r}_S$ and light ray $\hat{l}_S$, the specularity factor s would simply be cosine of the angle between the two vectors θ_r. As shown in 3.12, as the angle between $\vec{l}_S$ and $\vec{r}_S$ increases, the specularity decreases. Hence, a surface normal pointing more towards a light source would have higher specularity. Specularity factor s can be computed by taking the dot product between the two vectors as follows,

$$\mathbf{s} = F(\vec{r}_S \cdot \hat{l}_S)$$

where $\hat{l}_S$ is the light ray, $\vec{r}_S$ is the reflected ray and function F() can be used to control the cosine curve using one or more controls.

With the equation above, the darker regions will have a stronger base color while brighter regions will be influenced more by the light source.

3.5 Integrated equation for Shading

In this section, all the calculations from previous sections will be integrated into one single equation. The final color $\mathbf{c}_S$ at a shading point $\mathbf{p}'_S$ is the blending of base color with the colors from specular highlights and scattering.

The amount of specular and scattering highlights can further be controlled using coefficients $\mathbf{k}_s$ and $\mathbf{k}_f$ respectively. These coefficients decide how much of specular and scatter highlights are visible. If they both are 0, only the diffuse illumination is visible. The above equation can be modified as,

$$\mathbf{c}_S = (1 - \mathbf{k}_s)\,(\mathbf{c}_B + ((1 - \mathbf{k}_f) + \mathbf{k}_f\,\mathbf{f})\,\mathbf{d}\,\mathbf{c}_D\,\mathbf{c}_L) + \mathbf{k}_s\,\mathbf{s}\,\mathbf{c}_S\,\mathbf{c}_L$$

where:

$\mathbf{c}_B$ is the base color

$\mathbf{c}_L$ is the light color

$\mathbf{c}_D$ is the diffuse color

$\mathbf{c}_S$ is the specular color

$\mathbf{d}$ is the diffuse factor

$\mathbf{s}$ is the specularity factor

$\mathbf{f}$ is the forward scattering factor

$\mathbf{k}_s$ is the specularity coefficient

$\mathbf{k}_f$ is the forward scatter coefficient

If the illumination is high at a point, light color is more prominent. As illumination decreases, the impact of light is reduced and hence the base color becomes more visible. Specular and scatter coefficients $\mathbf{k}_s$ and $\mathbf{k}_f$ can be artistically controlled by the user to get the required results. These coefficients decide the material properties. If $\mathbf{k}_s$ is 1, it means the object is fully specular and if $\mathbf{k}_f$ is 1, the object is translucent. They can be adjusted to get a wide range of interesting images.

This equation is similar to the Phong shading equation [19], as the reflected light by a surface material is a combination of ambient light (base color), light reflected by rough surfaces (diffuse scattering) and by shiny surfaces (specular highlights). Figure 3.13 shows an illustration of the results of our model using height fields,

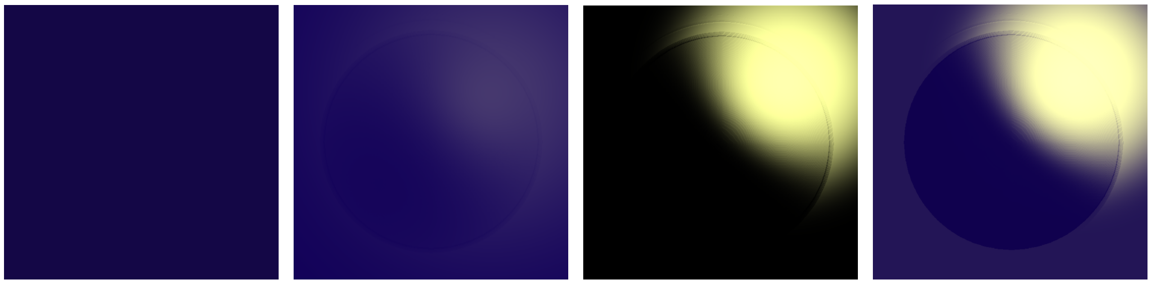
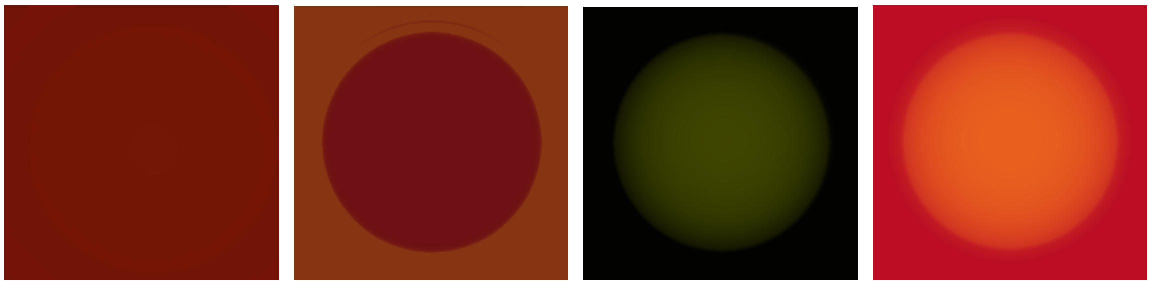

(a) Ambient. (b) Diffuse. (c) Specular. (d) Combination.

Figure 3.13: Results of our model taking a Phong-like shading approach for specular objects.

In addition to these shading layers, we include light scattered by translucent surfaces (forward scattering) as well. Forward scatter is visible when the light is in the back of the object as shown in Figure 3.14. In this scenario, the light is placed directly in the back of the sphere height field.



(a) Ambient. (b) Diffuse. (c) Forward Scatter. (d) Combination.

Figure 3.14: Results of our model to demonstrate forward scattering for translucent objects.

4. IMPLEMENTATION AND RESULTS

4.1 Integrated Model Workflow

This section discusses about the overall workflow of the system. The result of this model would be the rendered output that is qualitatively consistent with $\cos\theta$ shading. The model requires only a Height Map Image as required input. In case of environment illumination, an Environment Map must be provided too. In addition, an Alpha Image can also be given which can be helpful to remove any artifacts and render a clear image.

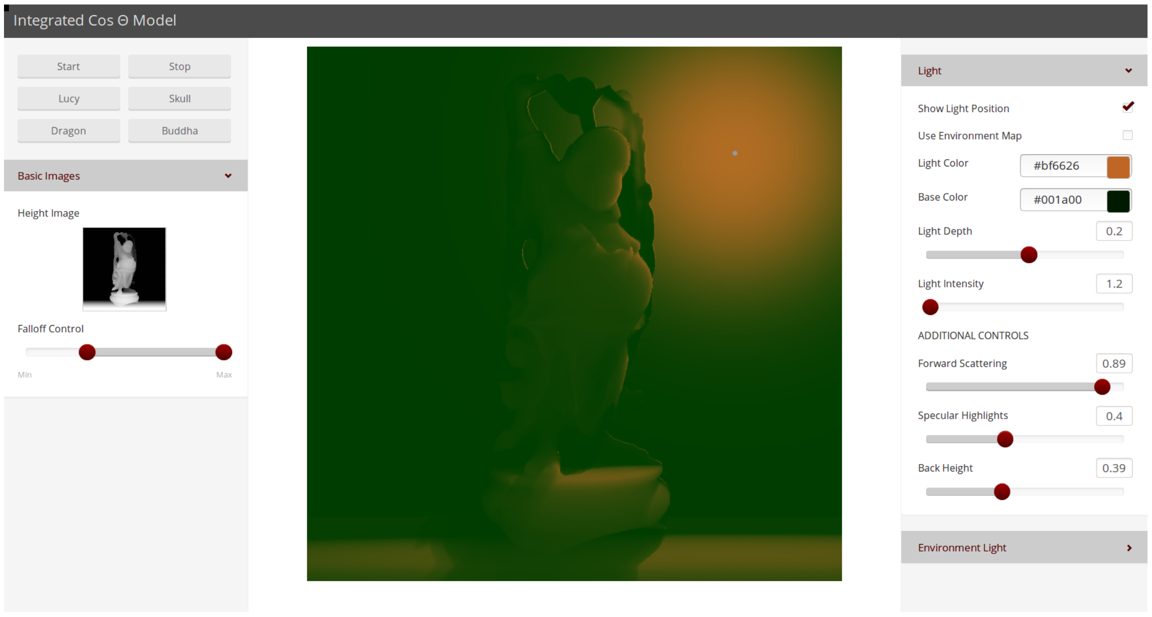
4.2 Framework Design

The implementation framework for this integrated model has been extended from Xiong's web framework [7]. Xiong's thesis elaborates about the different User Goals and Product requirements that was taken into consideration for designing this tool. She also talks about Component Design and the importance of having a Responsive Interface so that it can be accessed from various mobile or desktop devices. This model follows the same look and style for creating controls and rendering images. Implemented using WebGL and Javascript, the rendering is done real-time with interactive controls. Figure 4.1 shows the web interface of this tool.

4.3 Rendering

The rendering of a final image with this model involves different components: Height Field, Light or Light Map, lighting controls and shading coefficients. In the following sections, these components will be explained in detail.

Similar to Xiong's framework, rendering is done using orthographic projection, which means the direction of projection is normal to the projection plane [20]. The direction of projection is parallel to z-axis and the projection is done on the (x, y) plane at $z = 0$.



Show Light Position ✓
Use Environment Map ☐
Light Color #bf6626
Base Color #001a00
Light Depth 0.2
Light Intensity 1.2
Forward Scattering 0.89
Specular Highlights 0.4
Back Height 0.39

Figure 4.1: User interface of the integrated tool.

The camera is positioned on the +z side, pointing towards -z direction. Since the Center of Projection is at ∞, no perspective calculations are required here. Hence, a camera ray from the camera to a shading position will always be $(0, 0, -1)$.

4.3.1 Illumination Computation

This section explains how the illumination algorithm is implemented. For calculating the illumination at a shading point $\mathbf{p}_S$, we need the height information at that point and all points along the line segment $\mathbf{L}$ from $\mathbf{p}_S$ to light position $\mathbf{p}_L$. As explained in Algorithm 1, the illumination $\mathbf{d}$ is the inverse of the length of the line segment $\mathbf{L}$ in shadow. In the next section, calculating the light position $\mathbf{p}_L(\mathbf{x}_L, \mathbf{y}_L, \mathbf{z}_L)$ in canvas coordinates is discussed.

Let the shading point be $\mathbf{p}_S(\mathbf{x}_S, \mathbf{y}_S, \mathbf{z}_S)$, where $\mathbf{x}_S$ and $\mathbf{y}_S$ are the current shading positions (i.e. pixel position) along x and y axes respectively, and $\mathbf{z}_S$ is the height at point $\mathbf{p}_S$ which is obtained from height map $\mathbf{h}_{map}$.

$$\mathbf{z}_S = \mathbf{h}_{map}(\mathbf{x}_S, \mathbf{y}_S)$$

The shading position $\mathbf{p}_S$ is then moved along its normal to be inside the actual height boundary as shown in Figure 3.7). The new shading point is defined as,

$$\mathbf{p'}_S = \mathbf{p}_S - \mathbf{d}\hat{n}_S$$

where $\mathbf{d}$ is the distance by which $\mathbf{p}_S$ is moved along the normal $\hat{n}_S$ and $\mathbf{d} > 0$. Since we use orthographic projection, the normal vector is parallel to the z-axis and simply $\hat{n}_S(0,0,1)$. In that case, the modified shading point is now $\mathbf{p'}_S(\mathbf{x}_S, \mathbf{y}_S, \mathbf{z}_S - \mathbf{d})$.

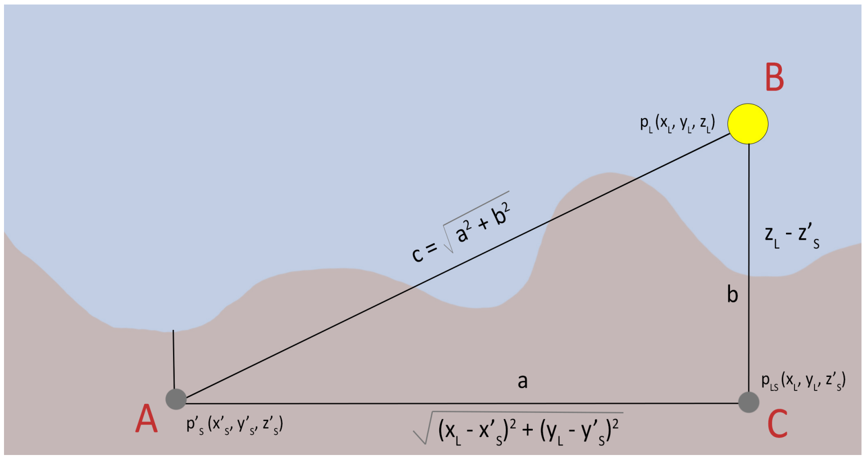


Figure 4.2: Computing increment vector based on shading position $\mathbf{p'}_S$ and light position $\mathbf{p}_L$.

As per Figure 4.2, a right triangle ABC is drawn with $\mathbf{p'}_S(\mathbf{x'}_S, \mathbf{y'}_S, \mathbf{z'}_S)$ and $\mathbf{p}_L(\mathbf{x}_L, \mathbf{y}_L, \mathbf{z}_L)$ as two of its vertices adjacent to hypotenuse $\mathbf{c}$. The lengths of the sides can be calculated as,

$$\mathbf{a} = \sqrt{(\mathbf{x}_L - \mathbf{x'}_S)^2 + (\mathbf{y}_L - \mathbf{y'}_S)^2}$$

$$\mathbf{b} = \mathbf{z}_L - \mathbf{z'}_S$$

$$\mathbf{c} = \sqrt{\mathbf{a}^2 + \mathbf{b}^2}$$

An increment vector $\vec{inc}$ is needed to travel along the line segment $\mathbf{L}$. and it depends upon a given step size. Let the step size be $\mathbf{S}_{step}$. The increment vector $\vec{inc}(\mathbf{inc}_x, \mathbf{inc}_y, \mathbf{inc}_z)$ is then,

$$\mathbf{inc}_x = \frac{(\mathbf{x}_L - \mathbf{x'}_S)}{\mathbf{S}_{step}}$$

$$\mathbf{inc}_y = \frac{(\mathbf{y}_L - \mathbf{y'}_S)}{\mathbf{S}_{step}}$$

$$\mathbf{inc}_z = \frac{(\mathbf{z}_L - \mathbf{z'}_S)}{\mathbf{S}_{step}}$$

Next, the upper and lower limits along all dimensions need to be set so that we start traveling from $\mathbf{p'}_S$ along the line segment $\mathbf{L}$ and stop at $\mathbf{p}_L$. The following pseudo code is used for calculating the lower limit $\mathbf{p}_{low}(\mathbf{x}_{low}, \mathbf{y}_{low}, \mathbf{z}_{low})$ and upper limit $\mathbf{p}_{high}(\mathbf{x}_{high}, \mathbf{y}_{high}, \mathbf{z}_{high})$,

```
if $\mathbf{x}_L < \mathbf{x'}_S$ then $\mathbf{x}_{low} = \mathbf{x}_L; \mathbf{x}_{high} = \mathbf{x'}_S$
else $\mathbf{x}_{low} = \mathbf{x'}_S; \mathbf{x}_{high} = \mathbf{x}_L$
if $\mathbf{y}_L < \mathbf{y'}_S$ then $\mathbf{y}_{low} = \mathbf{y}_L; \mathbf{y}_{high} = \mathbf{y'}_S$
else $\mathbf{y}_{low} = \mathbf{y'}_S; \mathbf{y}_{high} = \mathbf{y}_L$
if $\mathbf{z}_L < \mathbf{z'}_S$ then $\mathbf{z}_{low} = \mathbf{z}_L; \mathbf{z}_{high} = \mathbf{z'}_S$
else $\mathbf{z}_{low} = \mathbf{z'}_S; \mathbf{z}_{high} = \mathbf{z}_L$
```

Finally, to calculate the illumination factor $\mathbf{d}$ for shading point $\mathbf{p'}_S$, we travel from $\mathbf{p'}_S$ in increments of $\vec{inc}$ to reach $\mathbf{L}$. The lower and upper limits $\mathbf{lim}_{low}$ and $\mathbf{lim}_{high}$ help in making sure we don't go beyond line segment $\mathbf{L}$. Here is the pseudo code used to implement this concept,

```
$\mathbf{p}_i(\mathbf{x}_i, \mathbf{y}_i, \mathbf{z}_i) = \mathbf{p}'_S$
$\mathbf{s} = 1$
while $\mathbf{p}_{low} \leq \mathbf{p}_i \leq \mathbf{p}_{high}$ do
    if $\mathbf{z}_i < Height(\mathbf{p}_i)$ then
        $\mathbf{s} = \mathbf{s} + 1$
        end
    $\mathbf{p}_i = \mathbf{p}_i + \vec{inc}$
    end
    $\mathbf{d} = \frac{1}{\mathbf{s}}$
    return $\mathbf{d}$
```

4.3.2 Lights

The light mode can be chosen by the user based on their artistic goal. If the user needs only a single light source, point illumination can be used and if there is a necessity for an environment illumination, an environment map can be attached. Under Light Controls, this option can be set, as shown in Figure 4.3.

Figure 4.3: Checkbox for Lighting mode in user interface.

4.3.2.1 Single Light Source

In case of a single light source, the illumination algorithm as defined in Algorithm 1 only calculates the illumination factor once at a shading point. With that, the final color $\mathbf{c}_S$ at a point $\mathbf{p}_S$ is calculated by mixing the light color and base color. A single point light is represented as a dot on the screen. The light can be interactively controlled along the x and y dimensions using the mouse. To move the light in the z direction, a Light Distance control slider is provided in the interface.

The mouse coordinates differ from the canvas coordinates in terms of origin position and scaling. For shading calculations, the mouse position corresponding to a light position must be converted to it's corresponding position on the canvas [7]. The position should be normalized in order to support different aspect ratios. Let the origin of the canvas coordinate at the top left corner be $\mathbf{p}_{C0}(\mathbf{x}_{C0}, \mathbf{y}_{C0})$ and the lower right corner be $\mathbf{p}_{C1}(\mathbf{x}_{C1}, \mathbf{y}_{C1})$. In order to map the mouse coordinates to canvas coordinates, the following computation is done:

$$\mathbf{x}_L = \frac{\mathbf{x}_M - \mathbf{x}_{C0}}{\mathbf{x}_{C1} - \mathbf{x}_{C0}} + 0.5$$

$$\mathbf{y}_L = -\frac{\mathbf{y}_M - \mathbf{y}_{C0}}{\mathbf{y}_{C1} - \mathbf{y}_{C0}} + 0.5$$

where $\mathbf{p}_M(\mathbf{x}_M, \mathbf{y}_M)$ are the mouse coordinates and $\mathbf{p}_L(\mathbf{x}_L, \mathbf{y}_L)$ are the canvas coordinates that correspond to them. So, the canvas coordinate in 3D will now be $\mathbf{p}_L(\mathbf{x}_L, \mathbf{y}_L, \mathbf{z}_L)$, where $\mathbf{z}_L$ is the distance of the point light source along the z-axis. This conversion is explained in detail in Xiong's thesis [7].

In a practical scenario where the scaling ratio between the canvas and mouse coordinates is 1, the light position calculation will simply be,

$$\mathbf{x}_L = \mathbf{x}_M + 0.5$$

$$\mathbf{y}_L = -\mathbf{y}_M + 0.5$$

Figure 4.4 shows examples of lighting using a single point light at different positions and intensities.

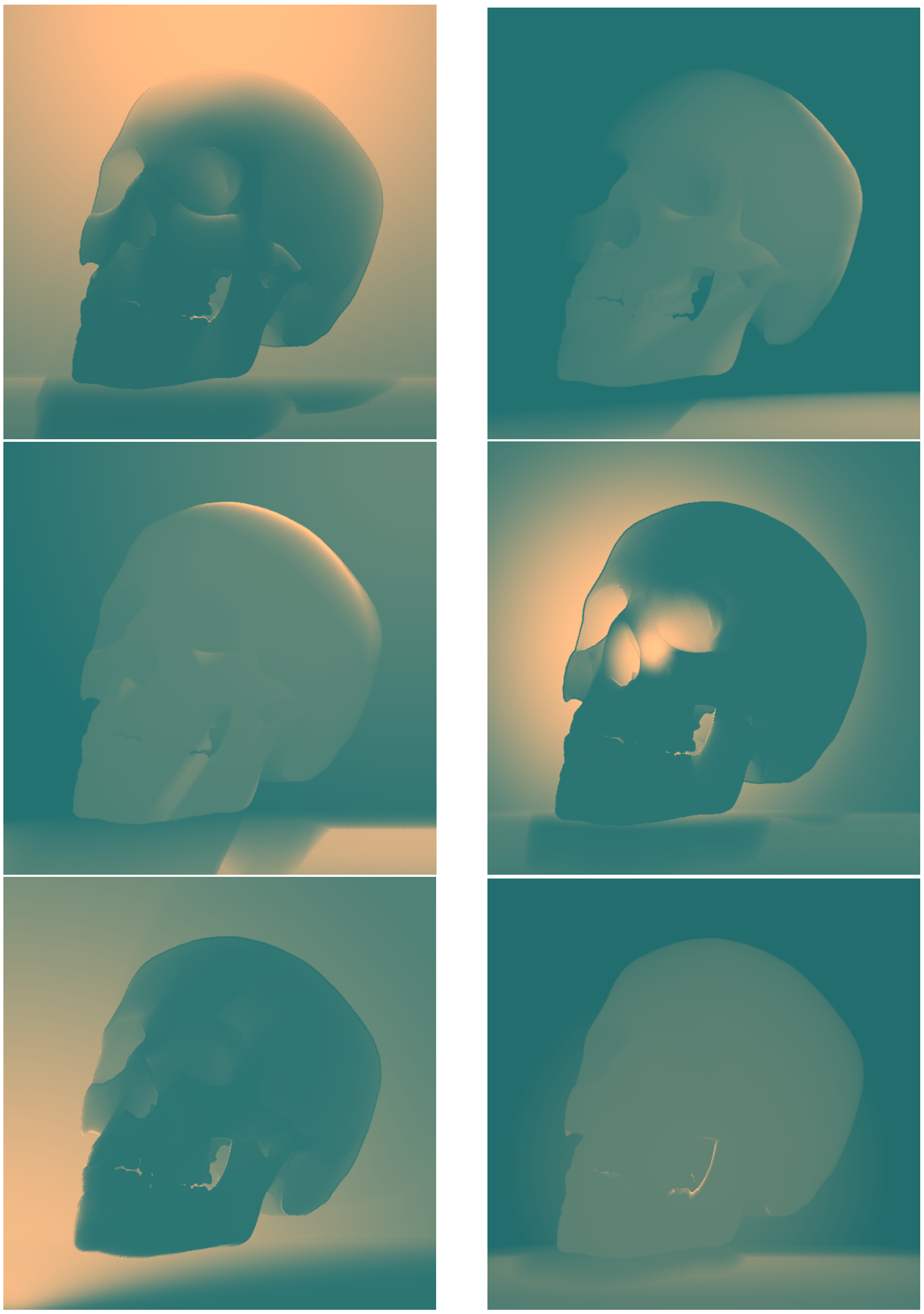


Figure 4.4: Examples of bas-reliefs created using a point light.

4.3.2.2 Environment Light

When environment lighting setup is enabled, each point on the environment map contributes to the shading at point $\mathbf{p}_S$ instead of one single point light. The illumination algorithm as defined in Algorithm 1 calculates the illumination contribution by each point on the environment dome i.e. each point on the dome is assumed to be a light source. The final color $\mathbf{c}_S$ is then the mixing of illumination from all points on the environment dome.

The environment map can also be controlled by the user using mouse controls. Similar to setting the point light position for a single light source as described in the previous section, the rotation of an environment map can be controlled by moving the mouse positions on the canvas. Let a 2D texture map $\mathbf{t}(\mathbf{u}, \mathbf{v})$ that will be mapped onto the sphere have a width and height of $\mathbf{width}$ and $\mathbf{height}$ respectively. The values for the coordinates range between $0 \leq \mathbf{u}_i, \mathbf{v}_j \leq 1$ and are calculated as,

$$\mathbf{u}_i = \frac{\mathbf{i}}{\mathbf{width}}$$

$$\mathbf{v}_j = \frac{\mathbf{j}}{\mathbf{height}}$$

Let the environment sphere have its center $\mathbf{p}_{L0}(\mathbf{x}_{L0}, \mathbf{y}_{L0}, \mathbf{z}_{L0})$ placed at the center of the canvas and radius of r. The canvas coordinates $\mathbf{p}_L(\mathbf{x}_L, \mathbf{y}_L)$ calculated above are added as offsets to the spherical coordinates,

$$\theta = \mathbf{u}_i + \mathbf{x}_L$$

$$\phi = \mathbf{v}_j + \mathbf{y}_L$$

where $\mathbf{t}(\mathbf{u}_i, \mathbf{v}_j)$ are the uv-coordinates on a 2D environment texture map for point (i, j) on the map. The i values are between $0 \leq \mathbf{i} \leq \mathbf{width}$ and j values are between $0 \leq \mathbf{j} \leq$

$\mathbf{height}$.

During implementation, we align the y-axis of the sphere is with the z-axis of the canvas. Hence, the light coordinates on the sphere $\mathbf{p'}_L(\mathbf{x'}_L, \mathbf{y'}_L, \mathbf{z'}_L)$ will now be calculated as follows:

$$\mathbf{x'}_L = \mathbf{r}\sin\theta\sin\phi$$

$$\mathbf{y'}_L = \mathbf{r}\cos\phi$$

$$\mathbf{z'}_L = \mathbf{r}\cos\theta\sin\phi$$

In Figure 4.5, we can see some of the images rendered using an environment sphere with a rainbow-colored texture mapped onto it.

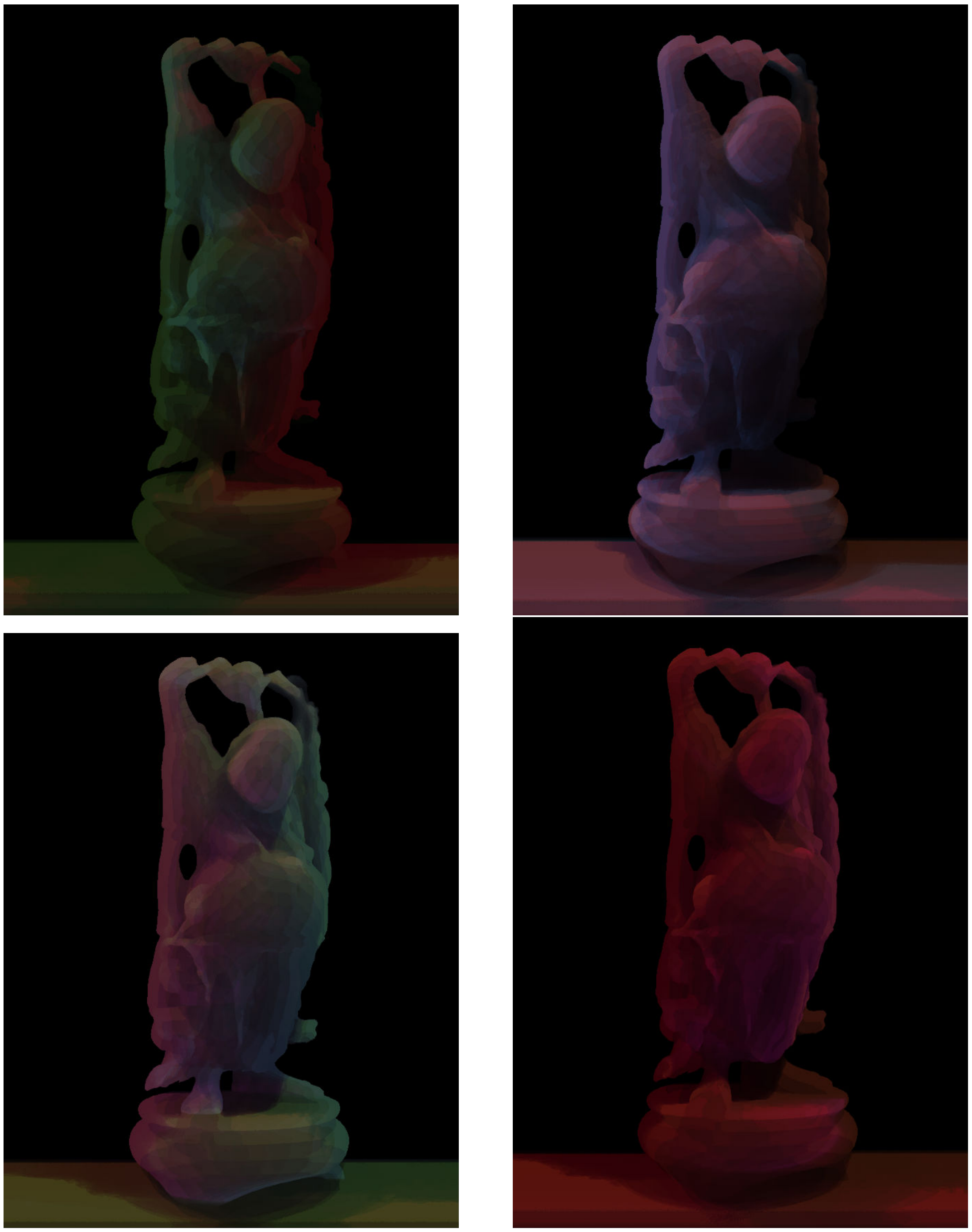


Figure 4.5: Examples of bas-reliefs created by rotating a rainbow environment map.

4.3.3 Soft Shadows

In the previous sections, we discussed about how the illumination algorithm is implemented and how the light positions are calculated for a single light source and an environment dome. As per Figure 3.7, diffuse $\mathbf{d}$ calculated using Algorithm 1 holds the illumination factor at each shading point $\mathbf{p}_S$. Since the algorithm calculates illumination based on the length of line segment $\mathbf{L}$, longer the length of $\mathbf{L}$, the more $\mathbf{p}_S$ is in shadow. In other words, larger hindrance for the light source leads to darker regions. Figure 3.6 shows how this works. The areas closer to region B is darker because the height field blocks most of the light that would reach the region. But region A is more exposed, so it would have a higher $\mathbf{d}$ value. As we move to the left from region B towards region C, the illumination factor gradually increases because it becomes more and more exposed to light, hence producing a soft gradient of shadow. Hence, illumination and soft shadow computation is integrated into the same algorithm producing results qualitatively consistent with $\cos\theta$ shading. Since this model only takes into account direct light sources and not light reflected off of scene objects, indirect illumination is not possible here. Diffuse can be rendered separately as shown in Figure 4.6 by simply using $\mathbf{d}$ as the color value,

$$\mathbf{c}_S(r, g, b, w) = (\mathbf{d}, \mathbf{d}, \mathbf{d}, 1)$$

4.3.4 Base Color and Light Color

With base color $\mathbf{c}_B$, diffuse color $\mathbf{c}_D$ light color $\mathbf{c}_L$, the illumination can be blended together to create more visually appealing results. Figure 4.7 shows the effect of having a base color for the object.

These colors can be chosen in the interface using color pickers as shown in Figure 4.8. The light intensity $\mathbf{I}_L$ is another factor that can be directly controlled by the user with a

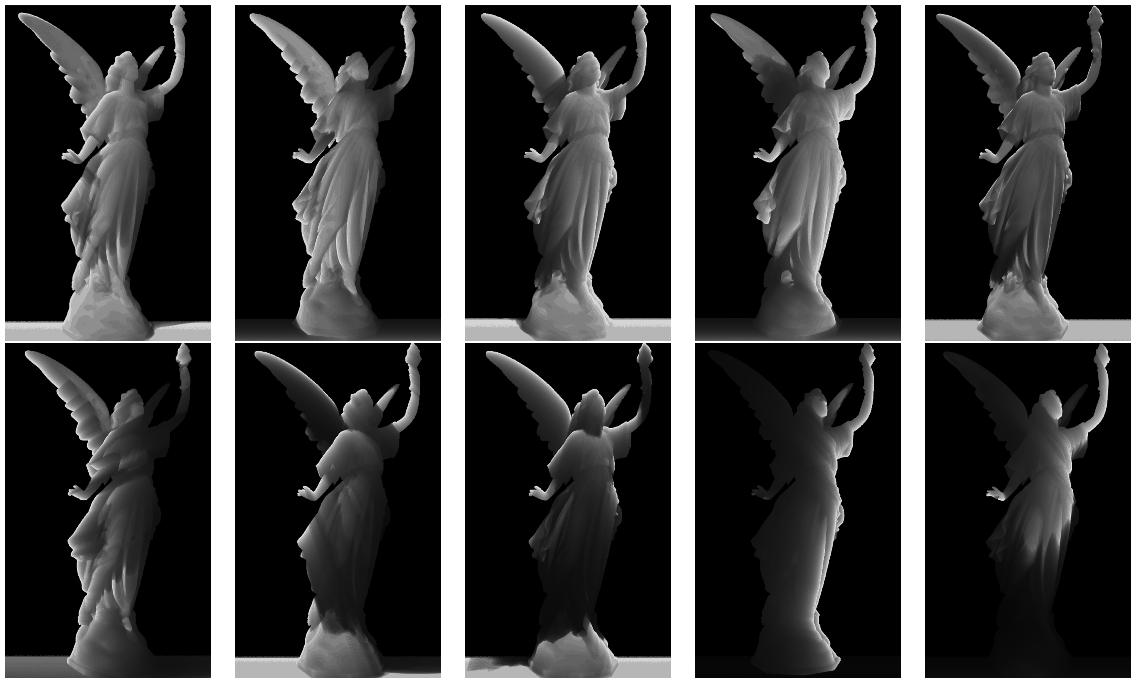


Figure 4.6: Examples of bas-reliefs of the diffuse and shadow using different light positions.

slider (Figure 4.9).

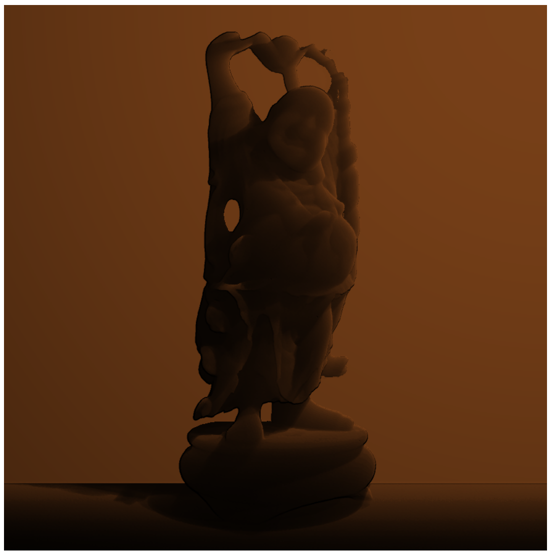
$$\mathbf{c}_S(r, g, b, w) = \mathbf{c}_B + \mathbf{d}\,\mathbf{I}_L\,\mathbf{c}_D\,\mathbf{c}_L$$

Figure 4.10 some examples with different lighting and color setups.

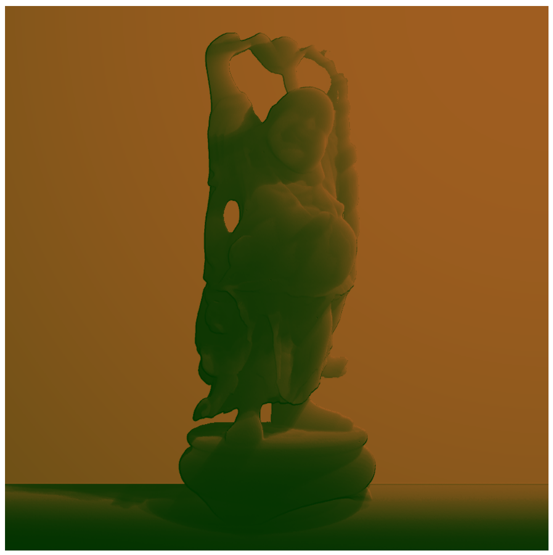
4.3.5 Specular Highlights and Forward Scattering

The eye position $\mathbf{p}_E$ for orthographic projection is at ∞, so the viewing direction from eye position to a shading point $\mathbf{p'}_S$ would be parallel to z-axis and hence, the incident ray $\vec{v}_S$ from $\mathbf{p'}_S$ to $\mathbf{p}_E$ is simply $(0, 0, 1)$. The light direction $\hat{l}_S$ from shading position $\mathbf{p'}_S$ to light position $\mathbf{p}_L$ is,

$$\hat{l}_S = \frac{\mathbf{p}_L - \mathbf{p'}_S}{|\mathbf{p}_L - \mathbf{p'}_S|}$$

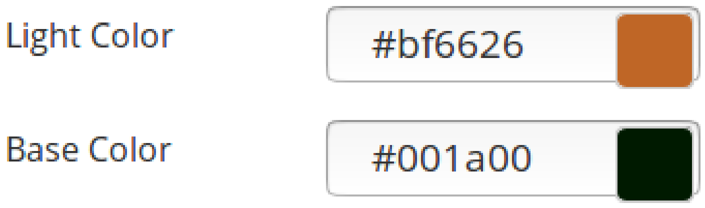


(a) Render with no base color.



(b) Render with a green base color.

Figure 4.7: Images showing results with and without blending a base color.



Light Color #bf6626 🟧

Base Color #001a00 ⬛

Figure 4.8: Color pickers in user interface.

4.3.5.1 Computation of surface normal

Height fields don't have any normal map information as they store only the relative height at each point. In order to calculate the surface normal approximately at a shading point, we find the difference between the heights before and after that point. The surface normal $\vec{n}_S(\mathbf{x}_N, \mathbf{y}_N, \mathbf{z}_N)$ is calculated as,

$$\mathbf{x}_N = \mathbf{h}_{map}(\mathbf{x}_{S-1}, \mathbf{y}_S).x - \mathbf{h}_{map}(\mathbf{x}_{S+1}, \mathbf{y}_S).x$$

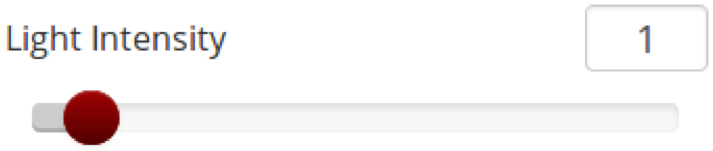


Figure 4.9: Intensity slider in user interface.

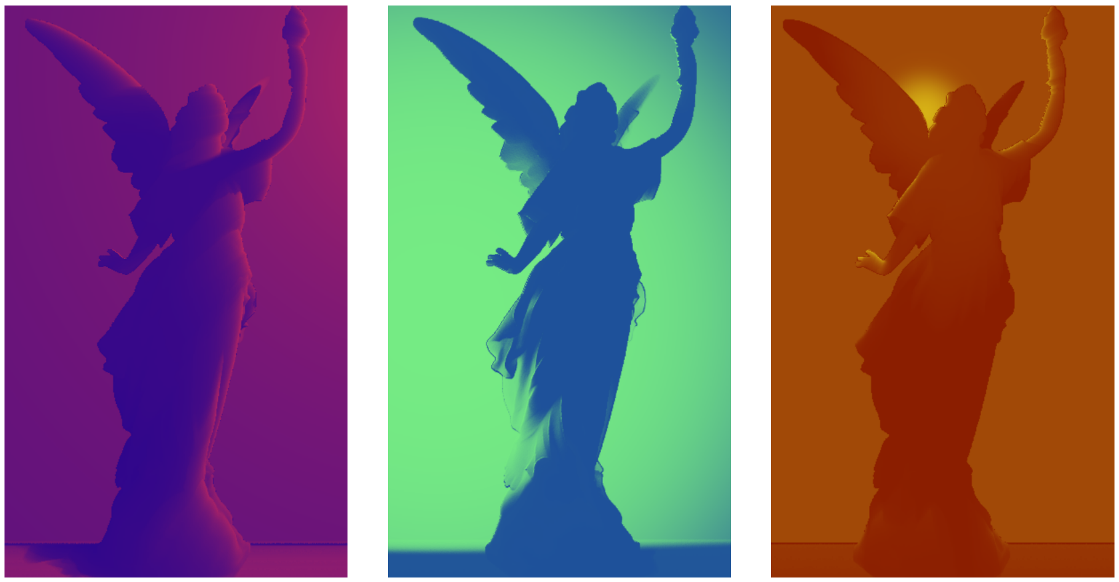


Figure 4.10: Examples of bas-reliefs with different base and light colors.

$$\mathbf{y}_N = \mathbf{h}_{map}(\mathbf{x}_S, \mathbf{y}_{S-1}).y - \mathbf{h}_{map}(\mathbf{x}_S, \mathbf{y}_{S+1}).y$$

$$\mathbf{z}_N = 1$$

The surface normal is then normalized,

$$\hat{n}_S = \frac{\vec{n}_S}{|\vec{n}_S|}$$

4.3.5.2 *Specularity*

With the surface normal $\hat{n}_S(\mathbf{x}_n, \mathbf{y}_n, \mathbf{z}_n)$ and incident ray $\vec{v}_S$, we can compute the reflected ray $\vec{r}_S$. The reflection equation can be simplified as follows,

$$\vec{r}_S = -\vec{v}_S + 2(\hat{n}_S \cdot \vec{v}_S)\hat{n}_S$$

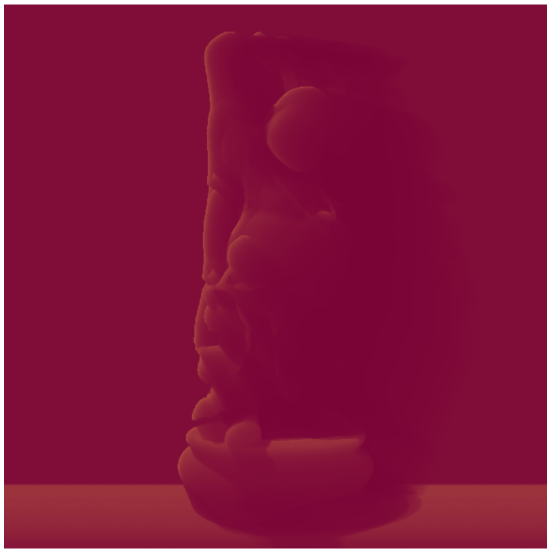
$$\vec{r}_S = -(0,0,1) + 2((\mathbf{x}_n, \mathbf{y}_n, \mathbf{z}_n) \cdot (0,0,1))(\mathbf{x}_n, \mathbf{y}_n, \mathbf{z}_n)$$

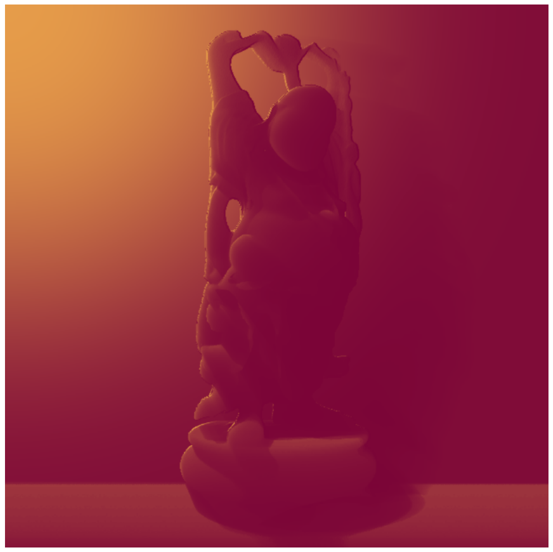
$$\vec{r}_S = (0,0,-1) + 2\mathbf{z}_n(\mathbf{x}_n, \mathbf{y}_n, \mathbf{z}_n)$$

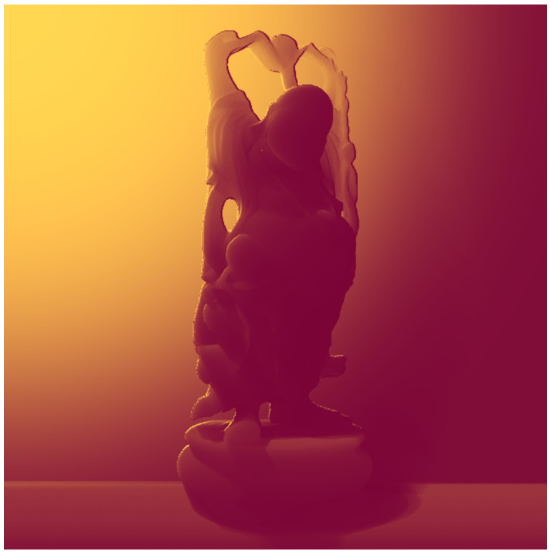
The dot product of the reflected ray $\vec{r}_S$ and light ray $\hat{l}_S$ would be the specularity factor,

$$\mathbf{s} = \vec{r}_S \cdot \hat{l}_S$$

Some examples for renders with different specular coefficients are shown in Figure 4.11


(a) $k_s = 0$


(b) $k_s = 0.25$


(c) $k_s = 0.5$

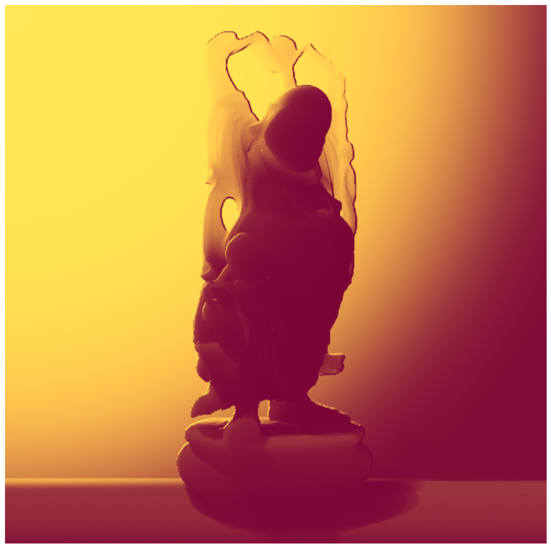

(d) $k_s = 0.9$

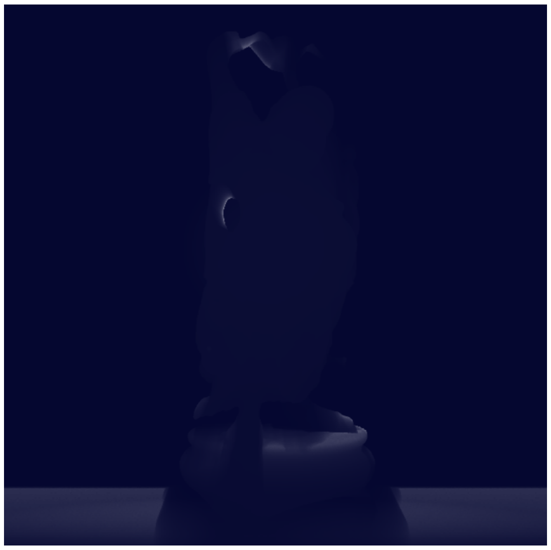
Figure 4.11: Examples of bas-reliefs with a front light and different specular coefficients $\mathbf{k}_s$.

4.3.5.3 Forward Scattering

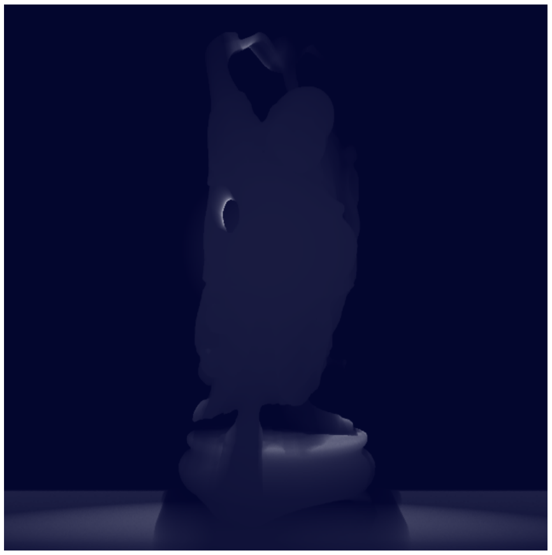
For forward scattering, we need the viewing direction in order to determine the amount of scatter. For orthographic projection, the eye direction $\hat{v}'_S$ from eye position $\mathbf{p}_E$ to shading point $\mathbf{p}'_S$ would be $(0, 0, -1)$. The dot product of the viewing direction $\hat{v}'_S$ and light ray $\hat{l}_S$ would be the scatter factor,

$$\mathbf{f} = \hat{v}'_S \cdot \hat{l}_S$$

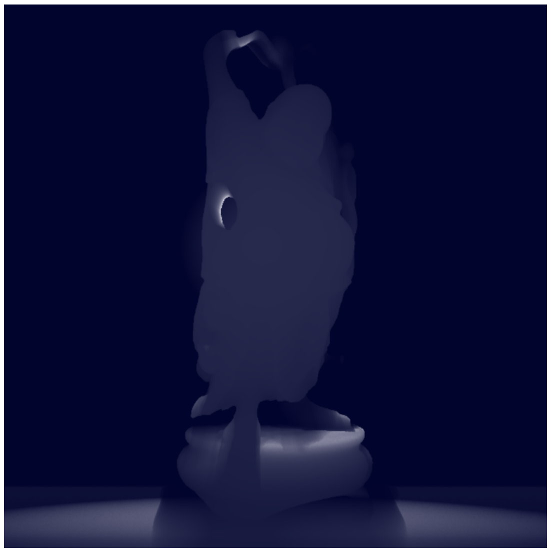
Some examples for renders with different forward scattering coefficients are shown in Figure 4.12



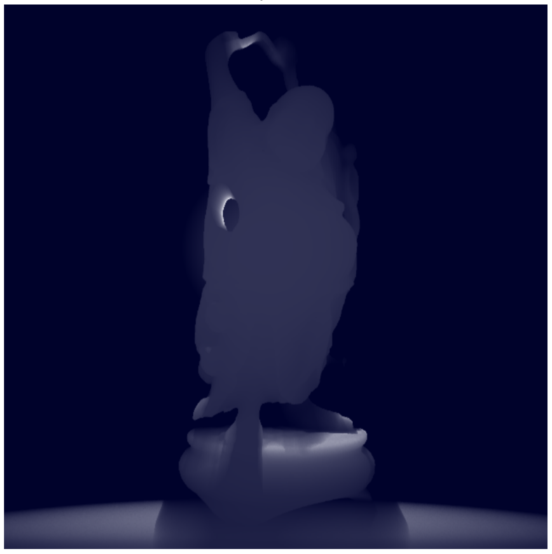
(a) $k_f = 0.1$



(b) $k_f = 0.4$



(c) $k_f = 0.75$

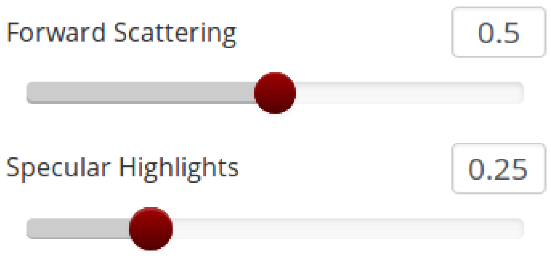


(d) $k_f = 1$

Figure 4.12: Examples of bas-reliefs with a back light and different forward scatter coefficients $\mathbf{k}_f$.

4.3.6 Falloff

The user can control the falloff of the light source using slider controls (Figure 4.13). The controls can be used to provide a range that decides if the light source has a sharp or gradual falloff. Let s_{min} and s_{max} be the minimum and maximum values for specularity f_{min} and f_{max} are minimum and maximum values for scattering. Then the falloff values can be modified as,


Forward Scattering 0.5

Specular Highlights 0.25

Figure 4.13: Slider controls for setting Specular and Forward Scattering .

$$s = \frac{s - s_{min}}{s_{max} - s_{min}}$$

$$f = \frac{f - f_{min}}{f_{max} - f_{min}}$$

Here are some examples of renders with different falloffs in Figure 4.14.

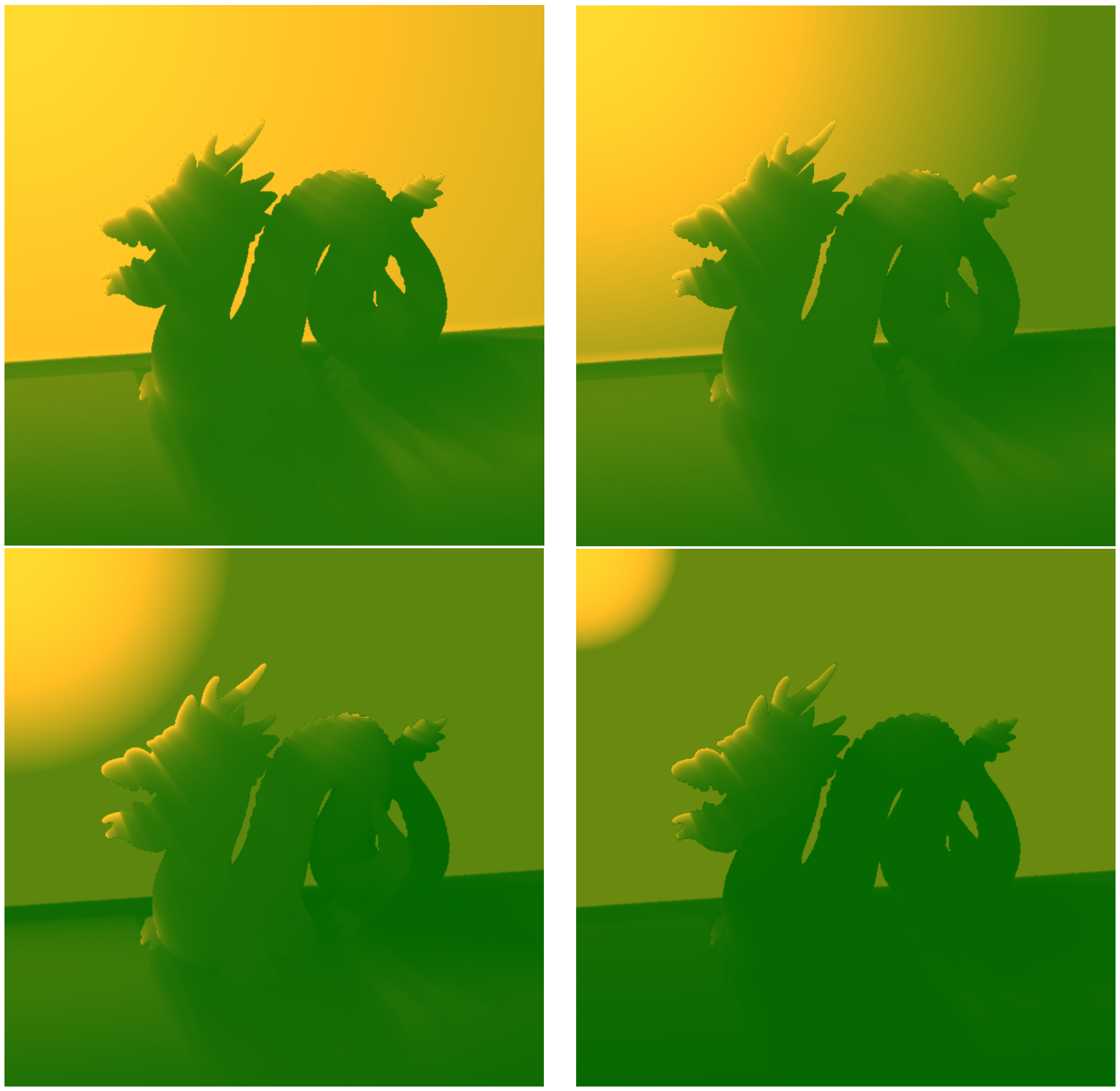


Figure 4.14: Examples of bas-reliefs with different falloff values.

4.3.7 Final Shading Computation

Using the above computations, the forward scattering and specularity contributions can be computed. The scatter and specularity coefficients can be controlled in the interface using sliders as shown in Figure 4.13. These settings will give the users some artistic control over basic material properties like the amount of reflectivity and translucency.

For a translucent object like skin, the scatter effect would be high while specularity is minimum. On the other hand, a burning candle would have some amount of reflective quality while being translucent. A shiny object like metal is fully opaque and hence will only have specularity. The equation will be modified as,

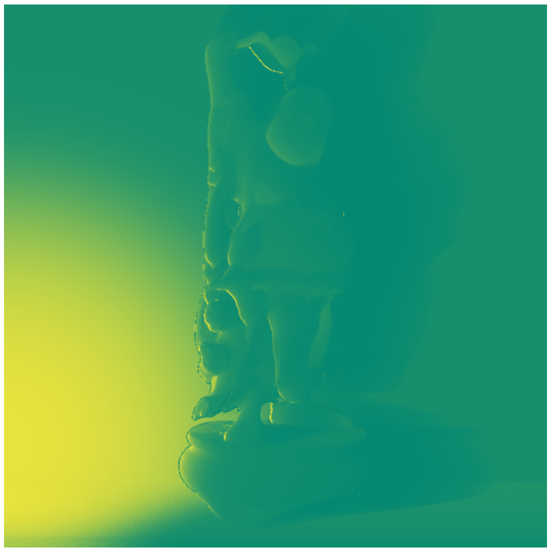
$$\mathbf{c}_S = (1 - \mathbf{k}_s)\,(\mathbf{c}_B + ((1 - \mathbf{k}_f) + \mathbf{k}_f\,\mathbf{f})\,\mathbf{d}\,\mathbf{I}_L\,\mathbf{c}_D\,\mathbf{c}_L) + \mathbf{k}_s\,\mathbf{s}\,\mathbf{I}_L\,\mathbf{c}_S\,\mathbf{c}_L$$

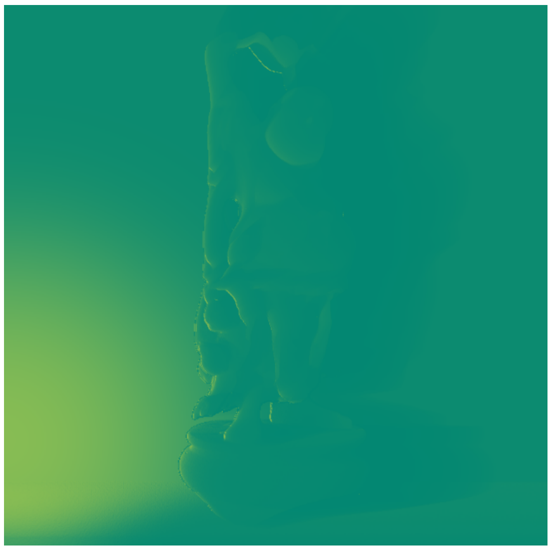
where $\mathbf{k}_f$ and $\mathbf{k}_s$ are the coefficients of scatter and specularity respectively.

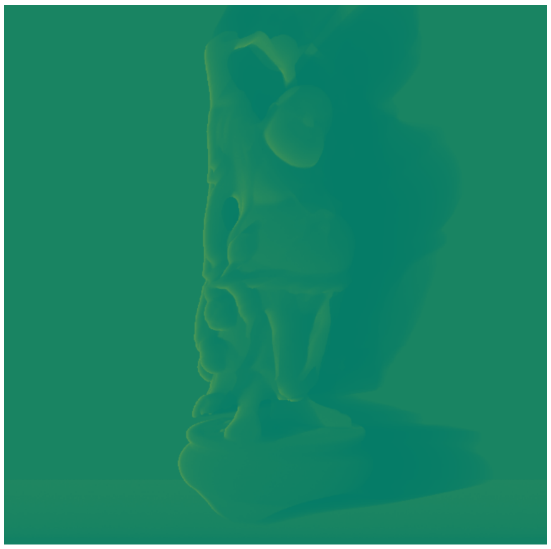
A simple trick is used for the final color calculation using the alpha channel value w. w hold the weights of each color being blended together using the above equation. In case of an environment light, each light sample will contribute to the color as well as intensity, thereby increasing the overall brightness of the output. So, as a final step, we divide the color with w to get desirable results.

$$\frac{\mathbf{c}_S(r, g, b, w)}{w} = \mathbf{c}_S(r/w, g/w, b/w, 1)$$

Figure 4.15 shows some examples of objects with different types of material property settings. Figure 4.16 shows some stylized renders of the Stanford Dragon using a rainbow environment map with varying material properties.


(a) $s = 0.5, f = 0$


(b) $s = 0.15, f = 0.5$


(c) $s = 0, f = 0$

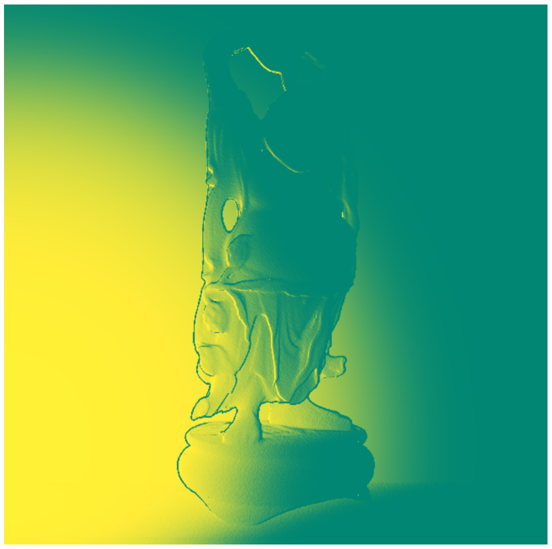

(d) $s = 0.8, f = 1$

Figure 4.15: Examples of bas-reliefs using a front light and different specular and forward scattering properties.

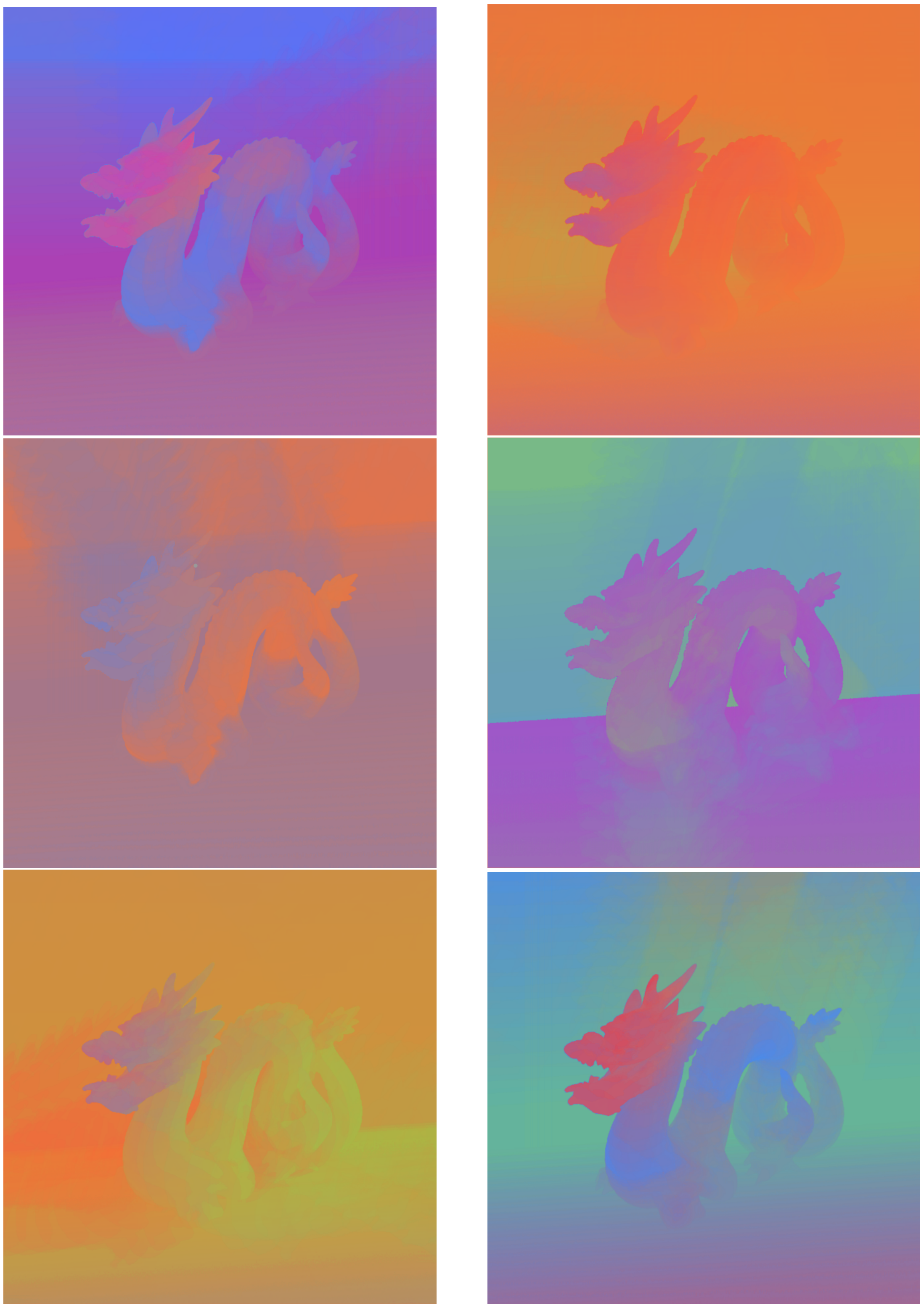


Figure 4.16: Examples of stylized renders using a rainbow environment map.

5. CONCLUSION AND FUTURE WORK

We presented an integrated model for computing direct illumination, scattering and soft shadows qualitatively consistent with the $\cos\theta$ techniques. Light scattering and transmission are computed using an integrated function similar to the BSDF function. The implementation of this model is really simple. The model computes direct illumination, soft shadows, specular highlights and subsurface scattering in a real-time web environment. By interactively moving the lights and using various user controls, the required artistic goal can be obtained.

This model can easily be included in a rendering system to replace models to obtain these three effects. Height fields are nothing but depth information stored in a 2D image. The integrated function introduced here can be extended to work in a 3D environment too, where the height fields would be the replaced with actual depth information. It can be integrated with modern ray tracing algorithms to obtain qualitatively consistent effects, and hence it has a lot of future development scope. As an improvement to the current system, the color values for materials can be replaced with actual texture maps to obtain more meaningful results similar to real-life.

REFERENCES

[1] S. F. Johnston, "Lumo: illumination for cel animation," in *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*, pp. 45–ff, ACM, 2002.

[2] H. Bezerra, B. Feijó, and L. Velho, "An image-based shading pipeline for 2d animation," in *Computer Graphics and Image Processing, 2005. SIBGRAPI 2005. 18th Brazilian Symposium on*, pp. 307–314, IEEE, 2005.

[3] C. Shao, A. Bousseau, A. Sheffer, and K. Singh, "Crossshade: shading concept sketches using cross-section curves," *ACM Transactions on Graphics*, vol. 31, no. 4, 2012.

[4] M. Okabe, G. Zeng, Y. Matsushita, T. Igarashi, L. Quan, and H.-Y. Shum, "Single-view relighting with normal map painting," in *Proc. Pacific Graphics*, pp. 27–34, 2006.

[5] C. Toler-Franklin, A. Finkelstein, and S. Rusinkiewicz, "Illustration of complex real-world objects using images with normals," in *Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, pp. 111–119, ACM, 2007.

[6] Y. Wang, *Qualitative Global Illumination of Mock-3D Scenes: Available electronically from hdl.handle.net/1969.1/157921.* PhD thesis, Texas A&M University, 5 2014.

[7] Y. Xiong, *Mock-3D Web Application: Interactive Lighting, Rendering and Shading for 2D Artwork.* PhD thesis, Texas A&M University, 2016.

[8] J. Wu, R. R. Martin, P. L. Rosin, X.-F. Sun, Y.-K. Lai, Y.-H. Liu, and C. Wallraven, “Use of non-photorealistic rendering and photometric stereo in making bas-reliefs from photographs,” *Graphical Models*, vol. 76, no. 4, pp. 202–213, 2014.

[9] D. Sỳkora, L. Kavan, M. Čadík, O. Jamriška, A. Jacobson, B. Whited, M. Simmons, and O. Sorkine-Hornung, “Ink-and-ray: Bas-relief meshes for adding global illumination effects to hand-drawn characters,” *ACM Transactions on Graphics (TOG)*, vol. 33, no. 2, p. 16, 2014.

[10] J. Wu, R. R. Martin, P. L. Rosin, X.-F. Sun, F. C. Langbein, Y.-K. Lai, A. D. Marshall, and Y.-H. Liu, “Making bas-reliefs from photographs of human faces,” *Computer-Aided Design*, vol. 45, no. 3, pp. 671–682, 2013.

[11] L. Governi, M. Carfagni, R. Furferi, L. Puggelli, and Y. Volpe, “Digital bas-relief design: A novel shape from shading-based method,” *Computer-Aided Design and Applications*, vol. 11, no. 2, pp. 153–164, 2014.

[12] T. Ritschel, T. Thormählen, C. Dachsbacher, J. Kautz, and H.-P. Seidel, “Interactive on-surface signal deformation,” in *ACM Transactions on Graphics (TOG)*, vol. 29, p. 36, ACM, 2010.

[13] B. Gooch and A. Gooch, *Non-Photorealistic Rendering*. Natick, MA, USA: A. K. Peters, Ltd., 2001.

[14] “Stanford computer graphics laboratory.” https://graphics.stanford.edu/.

[15] “Example: Creating a depth shader.” http://help.autodesk.com/view/MAYAUL/2017/ENU/?guid=__files_GUID_7BE4A6D9_7869_4065_A16B_D6D8D2D3B10E_htm. Autodesk®Maya®Help, available under a Creative Commons Attribution-NonCommercial-Share Alike license. Copyright ©Autodesk Inc.

[16] F. O. Bartell, E. Dereniak, and W. Wolfe, “The theory and measurement of bidirectional reflectance distribution function (brdf) and bidirectional transmittance distribution function (btdf),” in *1980 Huntsville Technical Symposium*, pp. 154–160, International Society for Optics and Photonics, 1981.

[17] K. D. Forbus, “Qualitative process theory,” *Artificial intelligence*, vol. 24, no. 1, pp. 85–168, 1984.

[18] H. W. Jensen, S. R. Marschner, M. Levoy, and P. Hanrahan, “A practical model for subsurface light transport,” in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 511–518, ACM, 2001.

[19] B. T. Phong, “Illumination for computer generated pictures,” *Communications of the ACM*, vol. 18, no. 6, pp. 311–317, 1975.

[20] J. D. Foley, A. Van Dam, S. K. Feiner, J. F. Hughes, and R. L. Phillips, *Introduction to computer graphics*, vol. 55. Addison-Wesley Reading, 1994.