PAINTERLY SHADING OCEAN SURFACE

A Thesis

by

ZHAO YAN

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

Chair of Committee,	Ergun Akleman
Committee Members,	John Keyser
	Philip Galanter
Head of Department,	Timothy McLaughlin

December 2015

Major Subject: Visualization

ABSTRACT

In this research study we explored the topic of bringing aesthetic characteristics from a seascape oil painting into 3D computer graphics. By applying the idea of barycentric shaders, we proposed a new shading workflow that guarantees to obtain the desired look-and-feel with a streamlined process.

We used the artwork from the renowned romantic artist Ivan Aivazovsky as our primary visual reference. First, we implemented a simulation tool with artistic control on the platform of a commercial software package to create a procedural ocean animation that matches visual storytelling. We then analyzed the characteristics of Aivazovsky's seascape paintings, which were then used as the guidelines for recreating animation in 3D computer graphics.

In the shading stage, we implemented a rendering architecture based on the idea of barycentric algebra. We redefine shader functions as parametric functions that satisfy the partition of unity, a concept that is widely used in geometric modeling. Our new framework allows computation separately on the front-end shader and back-end shader. The front-end shader is only used to compute color based on the incoming illumination. The back-end shader is barycentric shader which use the images generated by front-end shader as control parameters. Regardless of how illumination is computed, this framework guarantees a consistent style. As a result, a short animation with style consistent with Aviazovsky's seascape oil painting is created to demonstrate the artistic intention.

ACKNOWLEDGEMENTS

I would like to thank Dr.Ergun Akleman, my committee chair, for providing me continuous guidance and support while I was working on my thesis. I would also like to thank the rest of my committee, Prof. Philip Galanter and Dr. John Keyser for their expertise and wise guidance they have put into this thesis.

I would like to thank my parents for their support, trust and encouragement through the years. Most of all, their unconditional love gives me the opportunity to allow this curious heart to explore the new worlds.

NOMENCLATURE

2D	Two-dimensional
3D	Three-dimensional
CG	Computer Graphics
NPR	Non-Photorealistic Render

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

1. INTRODUCTION

1.1 Motivation

The motivation of this research comes from the simulation of mesmerizing beauty that waves exhibit in nature. As an important subject for painting and illustrations, waves have been depicted by many artists. In this thesis, we focus on one of the seascape paintings of the Russian Romantic painter Ivan Konstantinovich Aivazovsky, who created more than 3000 seascape paintings through his lifetime and is considered one of the most prominent marine artists of the 19th century [1].

Another important motivation comes from the critical need for an effective workflow to match computer-generated imagery with traditional artwork. In the industry of computer animation and visual effects, the look development phase of a project will typically rely on artists who use hand drawn images to convey their visual concepts. Thus, the real art direction problem is very close to that of matching the style of a painting.

1.2 Goals in Stylistic Rendering of Ocean Surface

In this work, we are interested in the stylistic rendering of ocean waves. As we mentioned earlier, ocean waves have always been an important subject in paintings and illustrations. There also exists a wide varity of styles in depicting waves. Consider, for example, one of the most famous woodprint paintings, "The Great Wave Off Kanagawa" by Japanese artist Katsushika Hokusai (see figure 1.1). The Great Wave Off Kanagawa demonstrates four types of stylization that exist in depicting waves.

1. **Stylized Wave Shape:** As shown in the image(see figure 1.1), artists stylize the wave shapes to obtain a new aesthetic. There has been existing work on stylizing wave shapes. For instance, Jay Allen Faulkner developed a method for creating stylized wave shapes using Bezier curves [6].

2. **Stylized Wave Motion:** This particular still image(see figure 1.1) does not demonstrate motion; however, the stylized wave motion needs to correspond to stylized shapes. If there is no consistency between stylized characters and animation, the resulting animations will not look visually coherent. To solve this problem, Sarah Beth Eisinger recently implemented the principles of hand-drawn animation to create artistic effects motion [4].

3. **Stylized Perspective:** Artists almost never use correct perspective [17]. Even correct looking perspective can only be correct locally. Katsushika Hokusai's image(see figure 1.1) also demonstrates such perspective irregularities that was studied by Jonathan H. Kiker. He also developed a method to composite 3D digital work in traditional paintings using local perspective.

4. **Stylized Colors & Look-and-Feel:** This image(see figure 1.1) also demonstrates that colors do not have to be realistic to obtain an interesting result. As far as we know there exists no formalized way to obtain stylized colors & look-and-feel to create a stylized depiction of ocean waves. In this thesis, therefore, we focus on the creation of stylized color and tone. We develop a scientific approach to bring the visual characteristics from Aviazovsky's seascape paintings into 3D digital works (see figure 1.2).

(a) The Great Wave Off Kanagawa [14].

Figure 1.1: Example of tradtional art works depicting ocean scene: "The Great Wave at Kanagawa" is a Japanese woodcut print by Katsushika Hokusai

Figure 1.2: Primary visual reference "Stormy Sea" by Aivazovsky

Figure 1.3: Examples of animation feature films with ocean scene: “Cars 2” is produced by Pixar Animation Studios and released by Walt Disney Pictures.

1.3 Complexity in Matching Traditional Art Work

In recent years, many works have been done in the industry of computer animation and visual effects to create ocean scenes through computer algorithms. Such works include animation feather film "Cars 2" by Pixar Animation Studios (see figure 1.3). That film included a computer generated ocean animation with a realistic shading and rendering. However from my point of view, the artistic style of the ship character and the realistic ocean rendering are not visually coherent. This also motivates us to propose a more effective workflow into the process of rendering CG ocean.

Although a large number of public-shared shaders are available in different commercial software packages, matching the style of traditional artworks is still a difficult task. The complexity is brought by the physical concept that is rooted in traditional shader development. However in production, the decision-making process of artist creation is purely based on an artist's personal preference; thus it is likely that an inconsistency exists between artworks and physical laws. For instance, in the oil painting "Shadow Dance"(see figure 1.4), the blue channel of the region in shadow

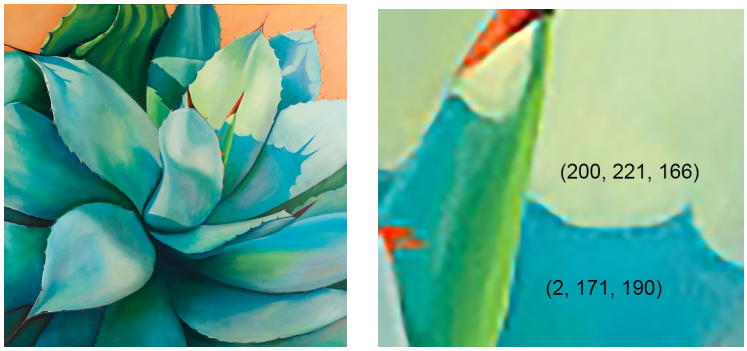


Figure 1.4: Oil painting that demonstrates conceptually inconsistency

is brighter than that under full illumination which is not logically expected.

In this case, ad-hoc solutions will be introduced to circumvent problems by highly qualified technical directors as an accepted practice. However, such "one-off" solutions are usually time-consuming and hard to be streamlined. Therefore, it is still not an easy task for artists to quickly go through simulations and rendering iterations and converge on the desired result.

The rest of the thesis is organized as follows. Section 2 summarizes previous research related to ocean simulation and rendering technology. Section 3 presents an artistic analysis of the visual reference, and then discusses the shading methods to match the visual characteristics in computer graphics. Section 4 presents our approach of simulating a procedural ocean surface, as well as the implementation of shading methods. Section 5 presents the conclusion and potential future works.

2. PREVIOUS WORK

In this chapter, we will review some of the previous works related to the ocean surface simulations and renderings in computer graphics.

2.1 Ocean Surface Simulation

In the field of computer grahics, the previous methods of simulating ocean surfaces can be classified into two main categories: functionally based approaches and physically based approaches.

2.1.1 Functionally Based Methods

Simulating the ocean surface through functionally based approaches has been studies with a long history; generally these approaches can be classified into two main categories.

The first category is to model the surface of water by parametric functions in order to simulate the transportation of waves. Fournier and Reeves [9] first present their method based on Gerstner Wave Model [10] in which particles of water move in a circular or elliptical stationary orbit. The Gerstner Wave Model was originally developed long before computer graphics to model ocean water in oceanography. Parallelled by Fournier's work, Peachey [23] uses a height field to compute waves without breaking crests. Through his method, foam and spray can also be generated by integrating with a particle system. Ts'o and Barsky [28] later use Gerstner Wave Model combined with wave-tracing to simulate refracting waves, but they approximated the appearance of wave crest using the tension property of beta-spline. Hinsinger et al. [12] compute deep water surface waves at interactive frame rates. They compute surface points on an adaptive mesh defined by a projection from the

camera position which is then transformed into screen space. This allows them to filter waves that would not be visible and to focus sampling on any area of the computed surface for enhanced image quality in that region.

The second category of the functionally based approaches include works by Mastin [19] and Tessendorf [27]. The basic idea is to synthesize ocean waves as cyclical height field using the data derived from oceanography observation.

Mastin [19] uses Fourier synthesis to model the ocean surface. This approach is efficient in modeling still waves; However, the main drawback is that it does not support the wave animation very well since this model doesn't embed a time parameter to describe the propagation of the wave trains.

Tessendorf [27] presents a method to produce the wave surface animation through calculating the evolution of the spectrum in the frequency domain. This model sums up a large numbers of sinusoidal curves and is solved by Fast Fourier Transformation (FFT). The user can adjust the area of the synthesized surface region by connecting wave tiles since FFT is cyclical.

2.1.2 Physically Based Methods

Most of the physically based approaches in simulating motion of fluids are achieved by solving the non-linear, inviscid, and incompressible Navier-Stokes equations.

2.1.2.1 Simulation of Liquids

One of the most commonly used approach to simulate liquids is called Smoothed Particle Hydrodynamics(SPH) [21]. This method considers the volume of fluid consists of discrete regions; in the center of each region lies a particle with momentum and mass. Particles are moving inside of the fluid due to both external and internal forces such as pressure, strain and gravity. By using SPH, the problem of simulating liquids is converted to solve Navior-Stokes Equations for discrete particles. SPH is

very useful in the situation where there is significant splashes or explosions.

Foster and Metaxas [8] modify the classic marker and cell (MAC) [11] method and apply in solving full 3D Navier-Stokes Equation to obtain realistic fluids behavior. Stam [26] presents the stable semi-Lagrangian methods for solving the Navier-Stokes Equations at a decent computational price. By combining particles and implicit surfaces, Foster et al. [7] introduce a hybrid liquid model for simulating liquids. Enright et al. [5] modify the hybrid liquid model by applying the particle level-set approach. Their solution is most commonly used to produce highly realistic motion of complex water surface. However, the drawback of this method is lacking in small scale features and expensive in computation. More recently, Yuksel et al. [29] introduce a novel concept of wave particles to approximate the solution of the wave equation by storing wave trains on 2D particles.

2.1.2.2 Simulation of Breaking Waves

By using a linear approximation to solve the N-S equations, Kass and Miller [16] construct a height field to simulate a water surface without breaking crests. An interesting work on simulating breaking waves is done by Mihalef et al. [20]. Their approach involves computing a full 3D N-S solution on multiple 2D slices of a breaking wave and then combining the slices to form a 3D wave form. The initial conditions for their 2D crest shapes can be defined by an animator from a pre-computed library, therefore eliminating the computations for wave propagation to the point of breaking. Work on interactive frame rates also comes into focus when Miller et al. [22] uses SPH(as cited before) to simulate fluids. Using particles allows them to neglect mass conservation and the convection terms of the N-S equations, which greatly simplified their computations and increases their frame rate. More recently, Irving et al. [15] uses fluid cells to simulate large bodies of liquids by coupling a 3D N-S solution

for the air-water boundary while using a simpler 2D height field approach for the remaining volume.

2.2 Ocean Surface Shading and Rendering

The optical phenomenon of the ocean surface has been well studied. Fundamentally, the color of the ocean has mainly been contributed by several basic optical properties of water. These properties include specular reflection, refraction, fresnel reflectivity and transmissivity. Besides, light is also scattered and absorbed by the water volume below the surface due to water molecules and organic matter.

One of the earliest efforts to render ocean surface in computer graphics is from Fournier et al. [9]. They present a method based on the assumption that the final color of ocean is mainly derived from the reflection of the skydome. They use an environment map and ray-tracing render scheme to compute water reflections and combine highlight specular in post compositing processes.

Peachey et al. [23] presents a novel method to render water spray using the particle system. Particles are simulated by a separated program which uses the same procedural wave surface model in the final result as input. Each of the particles are allocated a data structure to store their positions, velocities and mass information. Then the information of the particles is written into the same scene description file which contains all the other surfaces in the scene and use Z-depth information to decide whether the particles are obscured or not during rendering time.

Ts'o and Barsky [28] presents an optimized algorithm for rendering the ocean surface by using texture maps and Fresnel reflection. Based on the Fresnel's Law and surface normal, a ratio parameter is calculated to determine what fractions of refracted and reflected color should be assigned to a shading point in the rendering phase. They also used an approximation of Fresnel's Law to reduce computational

cost.

3. METHODOLOGY

In this chapter, we present our simulation and shading approach in creating our final animation. First, we conduct a visual analysis to abstract the essential characteristics in Aivazovsky's seascape paintings for us to match in digital work. Based on our analysis, we propose the detailed simulation and shading approach to match each of the characteristics.

3.1 Visual Analysis

The purpose of visual analysis is to provide guidelines for creating digital work in CG that conveys the same look-and-feel of Aivazovsky's seascape paintings. The result of this analysis yields a list of essential characteristics from Aivazovsky's paintings, which can be categorized as follows:

- Wave Shape and Form
- Water Shading Components
- Color Distribution

For each aspect, we choose one specific painting from Aivazovsky for analysis; based on the analysis, we present a solution for each of the characteristics. Since the goal of this research is to explore the possibility of bringing aesthetic characteristic from oil painting into 3D computer graphics, the analysis only emphasizes the artistic style of Aivazovsky's paintings rather than proposes a physically accurate wave model.

Figure 3.1: An preliminary sketch from Aivazovsky to demonstrate how wave crest in his drawing resemble a sine trochoid curve in shape [13]

.

3.1.1 Wave Shape and Form

First analysis is based on wave shape and form. We use Aivazovsky's preliminary sketches as our primary visual reference(see Figure 3.1). The reason we choose sketch work instead of oil painting works for analysis is that sketches emphasize more on the overall composition of image rather than detailed application of color. The shape of wave can be better revealed by using sketch works.

// The complexity of Aivazovsky's wave shape cannot easily be summarized by one simple model. The shape we found that can best represent Aivazovsky's painting is called "trochoid"(see Figure 3.2). Trochoid shape is derived from sinusoidal curve and is described by the trail of a fixed point on a circle as it rolls along a straight line. The difference is that trochoid curve has a narrowing in peaks compared to the sinusoid. This narrowing or steepening of the peak becomes more pronounced as the wave amplitude increases(see Figure 3.1).

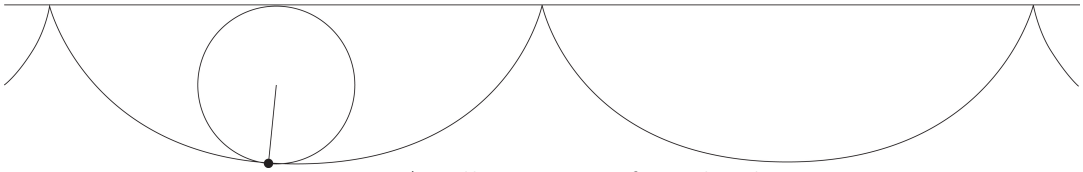


Figure 3.2: An illustration of trochoid curve

Figure 3.3: An example from Aivazovsky to demonstrate color variation from wave trough to crest, highlight specular and translucency effect

3.1.2 Shading Components

In the early idea of shade tree [2], which has evolved into modern industry standard, a complete procedural shader is described as a network of separate shading modules. Each shading module or component in that network should generate control parametric for its down-stream function component as input. This subsection is about dissecting procedural ocean shader into potential shading components. This analysis is related to how the artist applied color and tones to depict the translucent wave appearance. The purpose of this analysis is to identify the essential characteristics in order to match them respectively in the production stage. We choose Aivazovsky's oil painting "Stormy Sea"(see figure 3.3) as our visual reference.

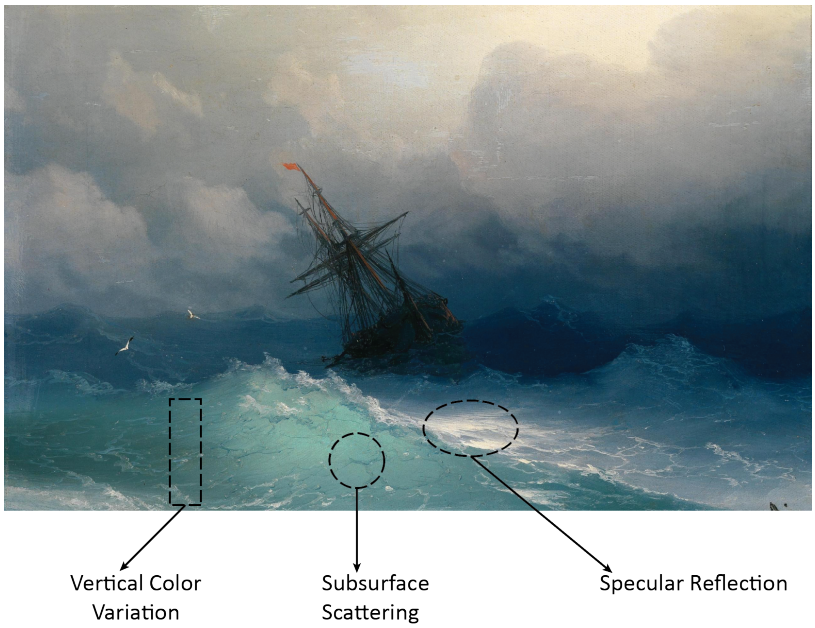


Figure 3.4: An illustration shows shading component of Aviazovsky's painting "Stormy Sea"

In the painting "Storm Sea"(see figure 3.3), we can divide the sea into three regions as foreground, mid-ground and background based on the distance from point position in 3D space to the viewer. Each region displays a different color distribution. Therefore, a control image of depth information will be needed to create separate color distribution. Besides, bright sunlight comes through a foggy atmosphere and casts a very strong specular reflection on the water surface. This transparent look of waves against sunlight is caused by subsurface scattering effect. Moreover, diffuse color of wave varies from dark green at wave trough to a light green at crest which brings the need to generate wave height information as control parameters in order to achieve color variation in vertical direction.

Based on the above analysis, we can summarize shading components into the following four categories, they are

- Non-Specular Component: Non-Specular component provides a single param-

eter that is a sum of all non-specular reflected light that is reaching to a given point. This component should take a sum of illumination as input and return a real number color value as output.

- Fresnel Component: Fresnel component is used to compute the Fresnel reflection/refraction contributions of a certain shading point given incident ray direction and surface normal as input.

- Specular Component: Specular component is to handle all specular reflected illumination for a given view direction. In our case, specular component should only consider the mirror reflection of sun light.

- Depth Component: Depth Component provides depth information about the distance from a view-point to the shading point. In our case, depth component should also provide the distance information of a shading point to its rest point before the simulation.

3.1.3 Color Distribution

In this subsection, we will discuss color and tones of Aivazovsky's painting. As is discussed before, in our primary reference waves can be divided into three separate regions according to their distance to the viewer in the 3D space. In this analysis, we will extract the darkest and brightest color from each region and create a color ramp for each region using the color values we sampled correspondingly.

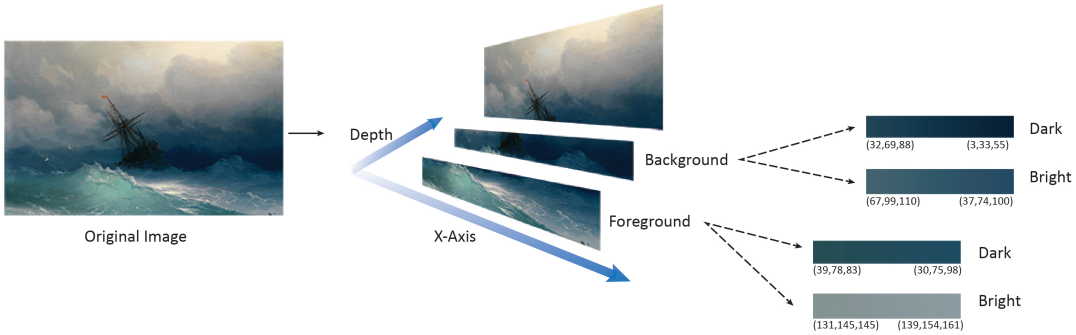
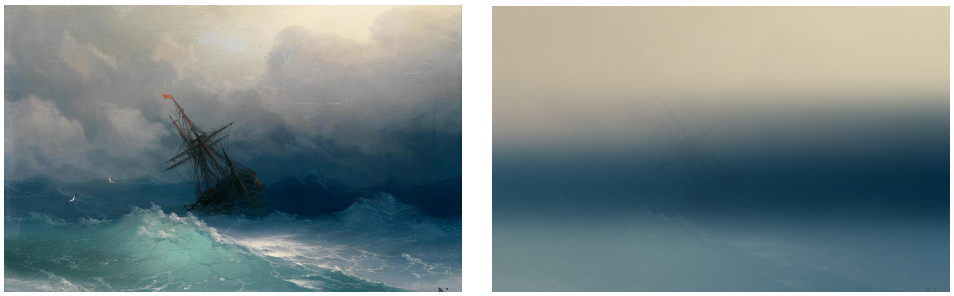


Figure 3.5: Color distribution analysis for oil painting "Stormy Sea"

As is shown in figure 3.5, if we cut the original painting into different layers and do a rough layout in 3D space, we can discover the fact that in the depth direction, color of the sea tends to grow darker and bluer as its depth value become larger. Similarly, color value also grows larger in the blue channel along the X-Axis for all three layers.

From left to right in 2D painting or along the X-Axis in 3D space, the darkest and brightest color for each layer still has a subtle change, however, we can ignore those changes for now as we focus on the analysis of overall color distribution. The brightest and darkest colors we detected from foreground is (131, 145, 5) and (39, 78, 83) respectively. Similarly, for the background the brightest and darkest colors is (67, 99, 110) and (32, 69, 88). We created color ramps based on those values on the right side of figure 3.5. By combining ramp layers and blurring the sharp transaction between layers, we can roughly obtain an image that shares the same look-and-feel with the original painting. This phenomenon of obtaining the same look-and-feel by pasting color ramps together is actually one of the most important inspirations for our shading approach (see figure 3.6).



(a) (b)

Figure 3.6: Two image showing how can we obtain the same look-and-feel by combining color ramps with the same color distribution

3.2 Methodology

In this section, we will discuss in detail on the simulation and shading approach in creating our final animation. By applying barycentric algebra on texture images, our new approach guarantees the style of resulting images is consistent with original painting.

3.2.1 Gestner Wave Model

The method we use to simulate wave surface is called Gerstner Wave Model [9]. The original Gerstner Wave Model is defined in two dimensional space. Let us orient our 2D world coordinates so that X-Axis represents the level of sea at rest while Y-Axis is pointing upwards.

Gerstner model assumes wave is a parametric surface defined by a set of particles. The trail of each particle forms a circle shape around its rest position(x_0, y_0).

The equations of motion for such a point is

$$x = x_0 + a \times sin(kx_0 - wt)$$

$$y = y_0 + a \times cos(kx_0 - wt)$$

Looking at equations above as function for 2D point (x_0, y_0) for a given t. The shape of this function is a trochoid. This trochoid curve can be viewed as the trail generated by a point P at a distance a from the center of a circle of radius $\frac{1}{k}$ rolling along the X-Axis. $P = (x_0, y_0)$ denotes point rest position, t is current time. The variable k represents the number of cycles per unit time, therefore is related to the period variable T, by $w = 2 * \pi/T$. The term in the parenthesis represents the wave phase, $\theta = kx0 - wt$.

Here the product $s = ka$ provides a measure of the sharpness of crest peaks(see figure 3.8), when $s = 0$, trochoid curve completely resembles a sinusoid curve while $s = 1$ gives the sharpest trochoid curve in a reasonable limit. After s exceeds 1, parametric curve will be self-intersected. Only at low values of s does the function give a wave form comparable to that of a sinusoid(see figure 3.7).

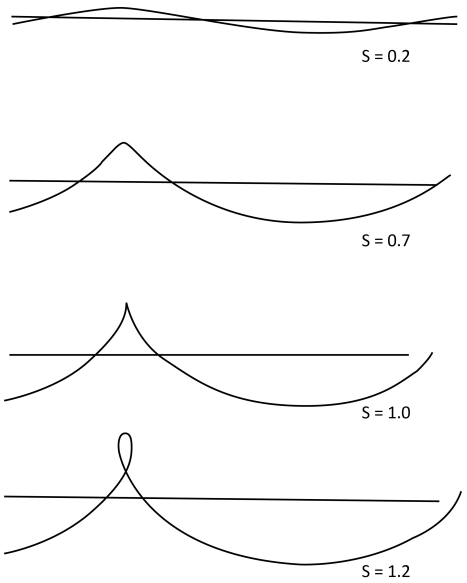


Figure 3.7: An illustration show different shapes of Gerstner wave according to different s values

The original Gerstner Wave is defined in two dimensional space which is sufficient for the use of 2D animation. However, in our case we need to transform the equation into three dimensional space. Another drawback of Gerstner Wave Model is that control parameters are not very intuitive, therefore it is not an easy task for an artist who may not have mathematical training to obtain desired simulation results very quickly. For our simulation software, we want to give the artist full control over the following parameters: (see figure 3.8):

- a: Amplitude
- l: Wave Length
- $\vec{d}$: Transportation Direction
- v: Velocity
- s: Sharpness
- n: Simulation Cost

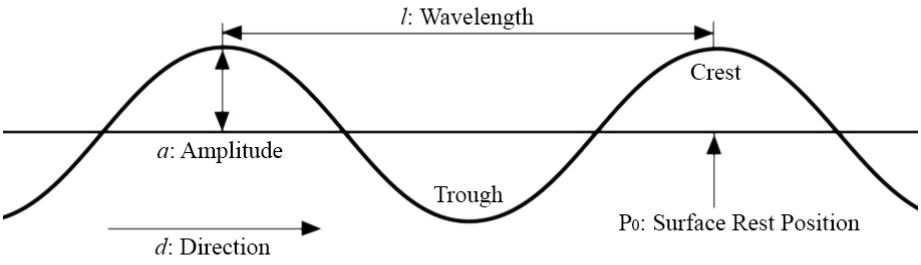


Figure 3.8: Wave parameters

First, we need to transform Gerstner Wave equation from two dimensional space to three dimensional space . We use right-hand coordinate system, so that the plane defined by X-Axis and Z-Axis represents the parametric sea surface at rest while the Y-Axis is pointing upwards. For a given point on the surface as $P_0 = (x_0, y_0, z_0)$, Gerstner Wave Model is transformed to the following 3D equations:

$$D_0(t) = s \times a \times sin(\frac{2\pi}{L}(\vec{d} \cdot p_0) - \frac{2\pi t}{v})$$

$$D_1(t) = s \times a \times cos(\frac{2\pi}{L}(\vec{d} \cdot p_0) - \frac{2\pi t}{v})$$

Where function $D_0(t)$ represents the displacement amount projected onto XZ plane given the time t, $D_1(t)$ is the displacement amount in Y-Axis. In the above function, v is a non-negative real number that denotes the speed of wave train, a is a scalar controls the amplitude of wave, s is the parameter controls the sharpness of wave. In the above function, we define the direction of wave as vector $\vec{d} = (d_0, d_1, d_2)$. Then we denote the new position after displacement as a triple of real numbers $P' = (x, y, z)$ where

$$x = x_0 + D_0(t) \times d_0$$

$$y = y_0 + D_1(t) \times d_1$$

$$z = z_0 + D_0(t) \times d_2$$

For the design of an intuitive user interface, we expose control of parameters directly to the user. These parameters include direction d, amplitude a, sharpness s, wave length l and velocity v. We use system clock for the time input(see Figure 3.9).

Figure 3.9: User interface design of wave simulator

The next step is to add multiple waves with different wave lengths and velocities to get our final simulation. The reason to use multiple waves instead of one is that one single wave shape cannot give us enough randomness and complexity that we observed in nature. Let us denote the final displacement as:

$$D(t) = \sum_{i=0}^{M} \omega_i D_i(t)$$

where d_s is a single wave displacement amount with the given amplitude, wave length, velocity and sharpness. ω_i's displace weight that satisfy partition of unity, i.e:

$$\sum_{i=0}^{M} \omega_i = \mathbf{1}$$

.

Since our method does not require physical accuracy, we don't need to use oceanographical data to drive our simulation. Instead, to simplify our computation, we set sharpness parameter s to a same constant value for every simulation and define amplitude parameter a to be directly proportional to wave length parameter l. The distribution of wave length in one simulation roughly resembles a log-normal distri-

bution where the wave length value specified by user is used as mean value.

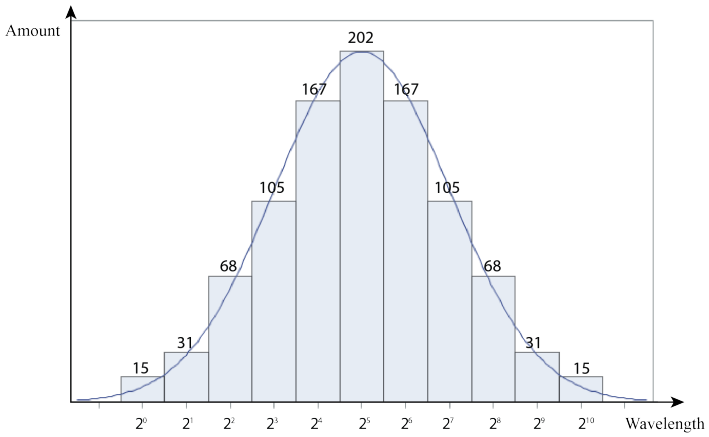


Figure 3.10: Distribution of wave length

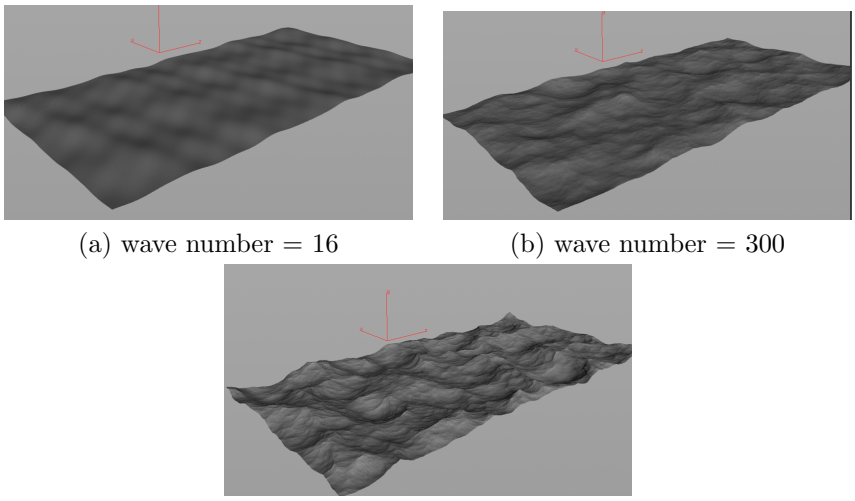
The computing cost of simulation is directly proportional to the wave number. The increase in wave number can improve the wave's look significantly by adding more details, however, as an expected trade-off, the simulation will be more time-consuming. In our case, we used 1000 Gerstner waves for our final simulation(see figure 3.11).

In the next subsections, we will introduce our render architecture based on the concept of barycentric algebra.

3.2.2 Shading Architecture

In visual narrative, the term look-and-feel refers to the unique expressive style for defining the world and its characters in the story. In the initial stage of CG production process, one of the most important tasks is to identify the desired look-and-feel of final result based on artistic reference.

From our point of view, the complexity of obtaining desired look-and-feel with traditional shading framework comes from two facts. Firstly, color in traditional



(a) wave number = 16 (b) wave number = 300

(c) wave number = 1000

Figure 3.11: Result of using different wave number

shade tree frameworks is represented by n-tuples positive real number. However, multiplication and addition operation to create new color are closed over all real numbers, not only positive real numbers. Because of this mismatch, shader network does not guarantee mathematical consistency.

Another reason is that for most off-the-shelf shading tools, colors is computed under a physically based framework. Within this framework, for instance, the color obtains under full illumination should always be larger than lack of illumination. However, when it comes to artistic creation, an artist's decision is made purely based on personal preference, therefore it is likely that inconsistency exist between physical concept and hand drawn images.

A shader designer's job is to replicate these inconsistencies in order to obtain the desired look-and-feel. In pracice, fully qualified technical director regularly need to employ ad-hoc solutions to meet special requirements. However, making those "one-off" solution is time-consuming and hard to be streamlined. Ad-hoc solutions, on the other hand, often create more problems than they solve, so we end-up with a

partial yet complicated solution to a very simple problem.

For mathematical consistency, we need a formal algebra that guarantees that, when operating on n-tuples of positive real number, we produce only n-tuple of positive real number. Fortunately, such algebras exist and are known as barycentric algebra [24] [3]. These are already familiar in computer graphics, as they are widely used in geometric modeling applications as barycentric coordinates.

However, with barycentric operators alone, we cannot obtain non-polynominal functions such as exponential, logarithm or cosine. These functions are needed for shader implementations that are related to geometry and are used to compute global or local illumination. On the other hand, the combination of colors can be handled entirely by barycentric operations. Therefore, we separate shaders into two types, as front-end shaders to do illumination calculation, and back-end shaders to mix colors, we will have a shading architecture that provides the best of both worlds.

- Front-End Shader: These are vector algebraic shaders that are used to compute and manipulate geometry related information such as displacement, normal vector or $cos\theta$. These shaders can be constructed like classical shaders using any function or operation. In other words, we can use the full power of shade trees concept in this level and allow manipulation with negative and complex numbers. These shaders only produces parameters to back-end shaders to compute colors.

- Back-End Shader: These are barycentric shaders that are constructed only by barycentric operations therefore guarantee that from color we can only obtain colors. These shaders are used to compute colors based on parameters that are passed from front-end shaders. Regardless of how parameters are computed in front-end shader, a given back-end shader guarantees to provide the same

style.

In the next subsection, we will discuss the five types of front-end shaders we designed.

3.2.3 Front-End Shader

We developed five different types of front-end shaders as shown in figure 3.12, they are

- Specular Shader
- Non-Specular Shader
- Spray Shader
- Fresnel Shader
- Depth Shader

All of the front-end shaders listed above are responsible for generating render passes which will be used as control parameters for back-end shading.

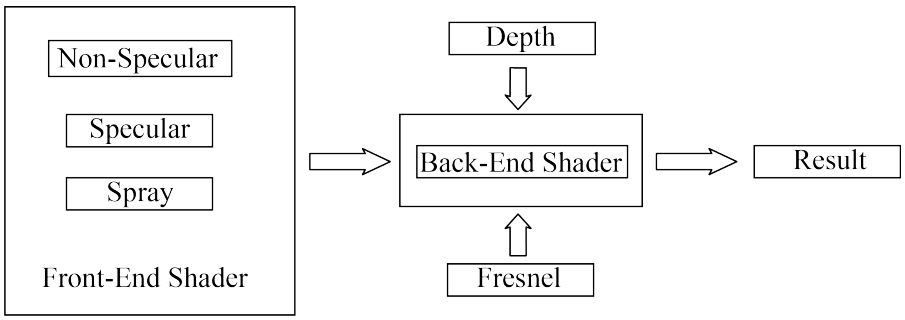


Figure 3.12: Shading architecture

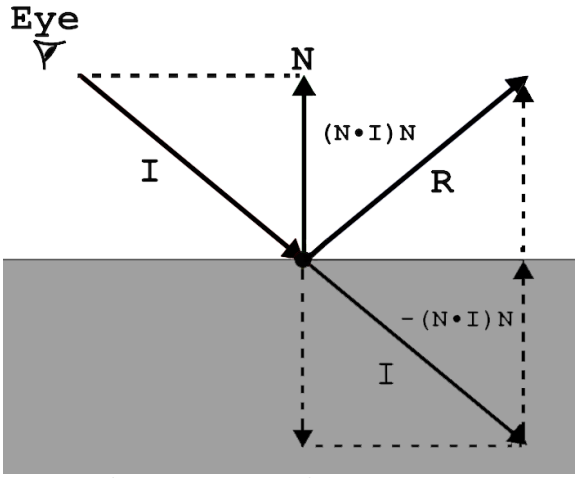


Figure 3.13: An illustration demonstrating how to compute a standard reflection ray R for an incoming ray I.

3.2.3.1 Specular Reflection Shader

The first front-end shader is specular reflection. The purpose of creating specular reflection shader is to mimic the effects of mirror reflection of sunlight in the original painting. According to the law of reflection, as shown in figure 3.13 the angle between incidence ray and the surface normal equals that between reflected ray and the surface normal. The incident, surface normal and reflected rays are on the same plane. The reflected ray R for an incoming ray I is computed as (see Figure 3.13):

$$\vec{R} = \vec{I} - 2\vec{N}(\vec{N} \cdot \vec{I})$$

In the production stage, virtual spotlight is placed to cast reflections on ocean surface to emulate the light source from sky in the painting. The result of specular front-end shader is shown in figure 3.14.

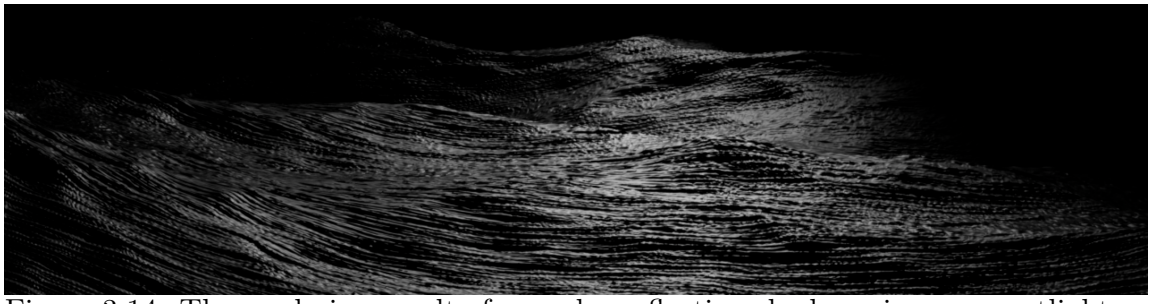


Figure 3.14: The rendering result of specular reflection shader using one spotlight as light source

3.2.3.2 Non-Specular Shader

Non-specular reflection describes the optical phenomenon that a light ray from a light source hits a surface and is reflected in all directions as is opposed to specular reflection. Non-specular reflection shader calculates the color given the lump sum of illumination from the scene. In optics, Lambert's cosine law says that the luminous intensity from an ideal diffuse reflective surface is directly proportional to the cosine of the angle between the surface normal and incident light ray [18]. We adapted Lambert's law to measure the intensity of illumination that reaches a certain point.

In order to estimate illumination intensity from an incoming light source, we uses the parameter $cos(\theta)$. For every shading point P on a 3D surface. $cos(\theta)$ is calculated as: (see Figure 3.15):

$$cos(\theta) = \vec{N} \cdot \vec{N_L}$$

where $\vec{N}$ is the surface normal at P and $\vec{N_L}$ is the outgoing light ray direction.

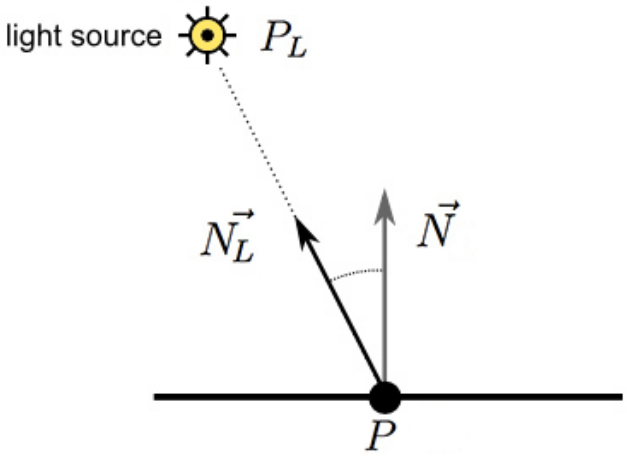


Figure 3.15: An illustration showing the relationship between shading point P, light source P_L, surface normal $\vec{N}$ and outgoing light ray $\vec{N_L}$.

However $cos(\theta)$ is a number between -1.0 and 1.0. So instead of using $cos(\theta)$ directly, we use shading parameter $C = \frac{cos(\theta)+1}{2}$ to remap $cos(\theta)$ from the range $[-1, 1]$ to $[0, 1]$. The next step is to create a color ramp using the brightest and darkest color we detected from water regions in the painting(see figure 3.17). Without loss of generality, we use a real number to represent color value instead of n-tuples. When we compute non-specular color for the surface point P, we first caculate its shading parameter C, then the shading parameter C to the color value by using color ramp. In practice, we realized that more than two colors can be used when construct the color ramp in order to better match our reference. Other modifications of parameter C is also acceptable, such as using C^2 instead of C to increase contrast in the rendering. The result of the non-specular front-end shader is shown in figure 3.16.

3.2.3.3 Fresnel Shader

Fresnel effect describes the observation that the amount of reflectance you see on a surface depends on the viewing angle. In computer graphics, fresnel shader allows

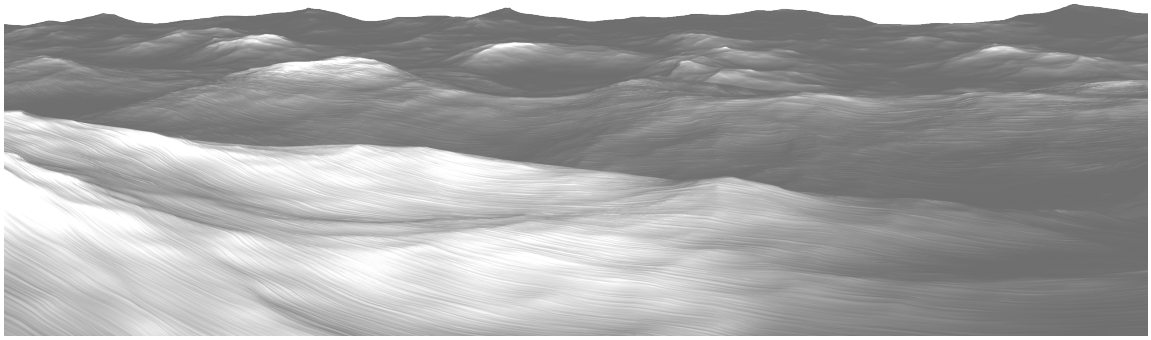


Figure 3.16: Non-Specular shader result

Figure 3.17: Remap function

reflection, specularity and other attributes to vary according to the viewing angle of the 3D surface. In practice, we used Schlick Approximation to compute fresnel.

According to Schlick's model, the specular reflection coefficient F can be approximated by:

$$F(\theta) = F_0 + (1 - F_0)(1 - \cos(\theta))^5$$

$$F_0 = (\frac{\eta_1 - \eta_2}{\eta_1 + \eta_2})^2$$

where θ is the angle between the viewing direction $\vec{V}$ and the half-angle direction $\vec{H}$, which is halfway between the incident light direction $\vec{L}$ and the viewing direction $\vec{V}$, hence $\cos\theta = (\vec{H} \cdot \vec{V})$. And η_1, η_2 are the indices of refraction of the two medias at the interface, and F_0 is the reflection coefficient [25]. One of the interfaces is usually air, meaning that η_1 can be very well approximated as 1. Rendering result using

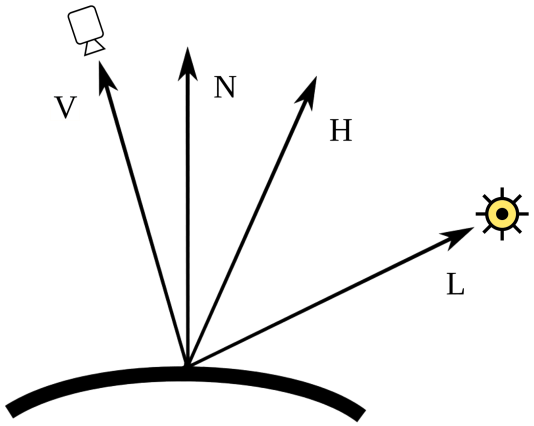


Figure 3.18: An illustration showing relevant parameters in Schlick Approximation

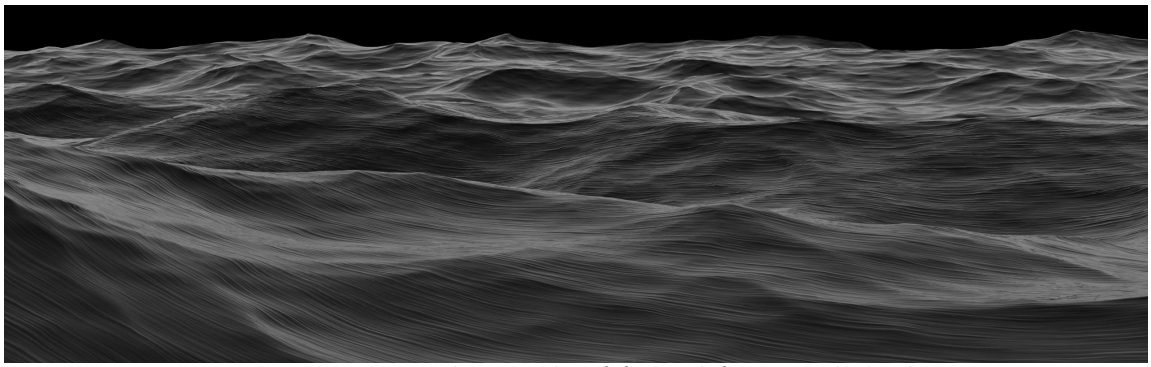
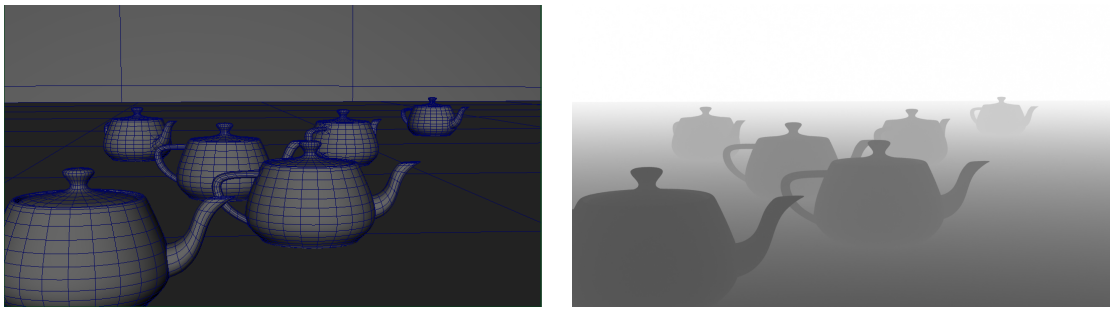


Figure 3.19: Render result of fresnel front-end shader

fresnel front-end shader is shown in figure 3.19.

3.2.3.4 Depth Shader

Depth front-end shader is used to generate depth control images for back-end shading. Depth image contains the distance information of how far object in 3D scene is away from the camera(see Figure 3.20).



(a) A screenshot of the 3D scene. (b) How the Z-Depth pass looks.

Figure 3.20: A depth pass example for a specific 3D scene.

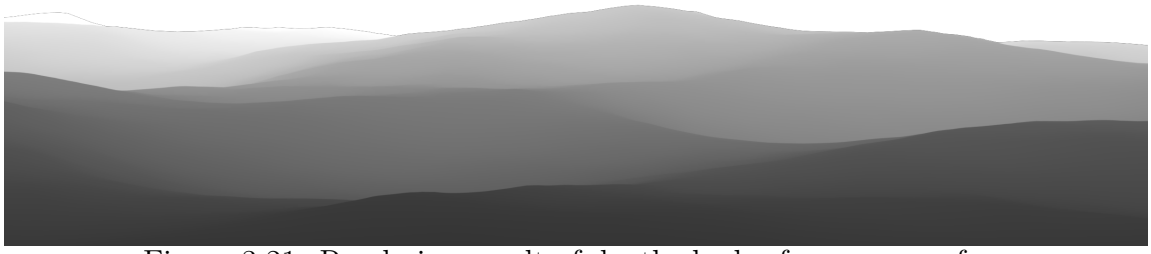


Figure 3.21: Rendering result of depth shader for ocean surface

The method we use to generate depth pass is illustrated in figure 3.22, where Z_s is the distance between a specific shading point P and the camera, which is computed as:

$$Z_s = (0, 0, 1) \cdot (P - P_C)$$

where P is the location of the current shading point. P_C is the location of the virtual camera from which we render the scene. And $(0, 0, 1)$ is a unit vector facing the Z-Axis of the camera (see Figure 3.22).

The range of Z_s is $[0, +\infty]$; therefore, we need to remap Z_s to $[0, 1]$ in order to display it properly in image space. We use *Clamp* function to achieve this remapping which will be discussed in the next chapter.

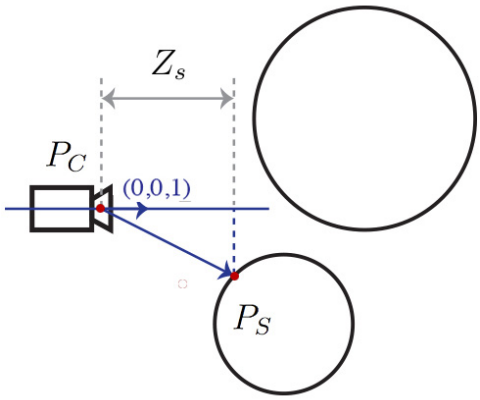


Figure 3.22: An illustration shows how depth value is computed

3.2.4 Back-End Shader

Our rendering architecture allows computation of color separately on front-end shader and back-end shader. Front-end shader is used to generate control parameters while back-end shader is based on barycentric algebra to preserve mathematical consistency.

3.2.4.1 Barycentric Algebra

Barycentric algebra is defined by a set of operations as

$$(x, y) \rightarrow t_0 x + t_1 y \quad where \quad t_0 \geq 0,\ t_1 \geq 0,\ t_0 + t_1 = 1$$

.

The property $t_0+t_1 = 1$ is called partition of unity. One application of barycentric algebra in geometry is barycentric coordinates system(see figure 3.23). Consider a triangle T defined by its three vertices, P_0, P_1, and P_2. For every point P located inside this triangle can be represented as a unique combination of the three vertices. In other words, for each P there is a unique sequence of three numbers, t_0,t_1,t_2 such

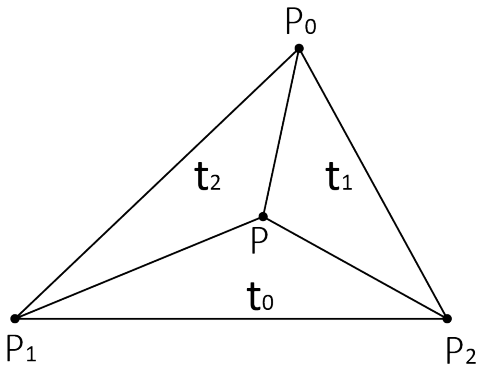


Figure 3.23: Barycentric coordinate system

that

$$t_0 + t_1 + t_2 = 1$$

and

$$P = t_0P_0 + t_1P_1 + t_2P_2$$

Here $t_0 + t_1 + t_2 = 1$ satisfies the restriction of partition of unity. If $t_0, t_1, t_2 > 0$, we say point P is inside the convex hull defined by P_0, P_1, and P_2. Please note that barycentric operation doesn't restrict the operands to the number of three, in fact we can use any number of operand in barycentric operation as long as they satisfies partition of unity.

3.2.4.2 Barycentric Shader

Using a Barycentric algebra in shaders does not require a significant conceptual change. In fact, we view barycentric shaders as if they are barycentric operations on texture images. We restrict shader operations only to the form

$$C = \sum_{i=0}^{M} \omega_i C_i \quad \text{where} \quad \sum_{i=0}^{M} \omega_i = 1 \quad \text{and} \quad \omega_i \geq 0$$

where the C and C_i's are colors, i.e. n-tuples of positive real numbers. Partition of unity is satisfied by the property that the weights ω_i are all positive and sum to 1, which guarantees that result color C stay inside of the convex hull defined by the colors C_i.

In our case, barycentric shader starts with non-specular color. We assume resulting non-specular colors are computed as a weight average of control colors. In the stage of back-end shading, the original non-specular color is directly used as weight parameter ω . We need to define control parameter C based on the depth parameter in order to carry out computation.

Here we need to further restrict both control parameters and weight parameters of color as n-tuple of positive real numbers between 0 and 1 for conceptual simplicity. The restriction of maximum number to 1 is not a problem since any set of real numbers can always be mapped into $[0, 1]$. For our purpose we use $Clamp$ function to re-map parameters, which is defined as follows:

$$Clamp(t, \max, \min) = \begin{cases} 1 & \text{if } \max \leq t \\ \dfrac{t - \min}{\max - \min} & \text{if } \min \leq t \leq \max \\ 0 & \text{otherwise.} \end{cases}$$

We define the barycentric operation Mix on colors by linear bezier interpolation as

$$Mix(C_0, C_1, t) = C_0(1 - t) + C_1 t \quad \text{where} \quad 0 \leq t \leq 1$$

Control colors for barycentric shader is obtained by assigning textures onto ocean surface. For instance, the texture representing bright tone of ocean is created as in figure 3.24, where C_0, C_1, C_2 and C_3 is the brightest color sampled from four corners in the painting. As is mentioned in the section of Visual Analysis, these colors are

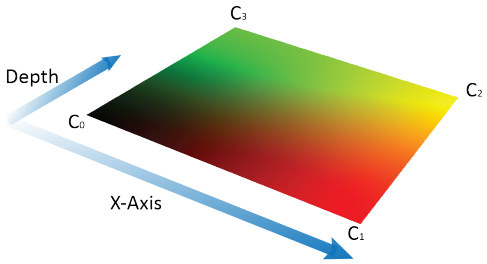


Figure 3.24: A visual demonstration of bilinear interpolation. Here in the depth direction, we interpolate the value of green channel between color C_0 and C_3, similarly in X-Axis, red channel is interpolated between color C_0 and C_1.

(131, 145, 145), (139, 154, 161), (37, 74, 100) and (32, 69, 88). For each shading point P, we map it from the global space to uv space as $P : (x, y, z) \rightarrow (t_0, t_1)$ by the depth value generated by depth front-end shader. Here t_0, t_1 are remapped to the range of $[0, 1]$ by

$$t_0 = Clamp(x, x_{max}, x_{min})$$

$$t_1 = Clamp(z, z_{max}, z_{min})$$

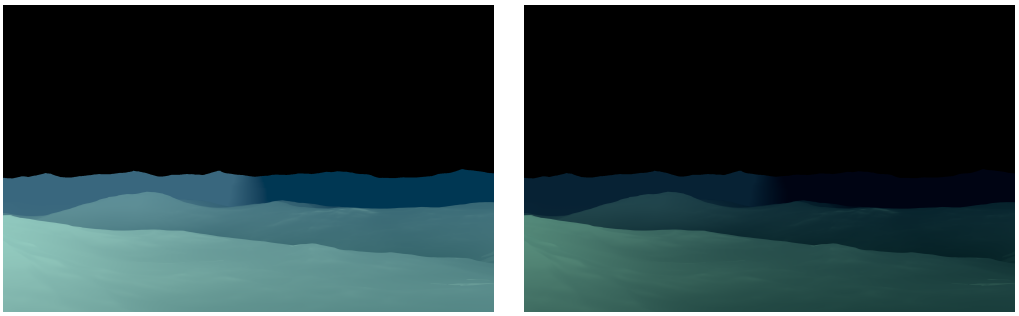
where x_{max} is the largest value detected in X-Axis, x_{min} is the smallest value in X-Axis. Similarly, z_{max} is the largest value detected in Z-Axis while z_{min} is the smallest value in Z-Axis. The final color in control image I for shading point P is obtained by bilinear interpolation as

$$C_{01} = Mix(C_0, C_1, t_0)$$

$$C_{23} = Mix(C_2, C_3, t_0)$$

$$C_{final} = Mix(C_{01}, C_{23}, t_1)$$

Since Mix operation is closed only in positive real numbers, our approach guar-



(a) bright tone (b) dark tone

Figure 3.25: Rendering result of assigning texture on ocean

antees the elimination of mathematical inconsistency. We applied the same method to create the control image for the dark tone of ocean rendering. The result of bright tone and dark tone of ocean surface is shown in figure 3.25

3.2.4.3 Application on Texture Image

We assume the resulting non-specular color are computed as a weight average of control colors. Therefore, a function that describes shading can simply be given as a weight average of a set of control images as $I = \sum_{i=0}^{M} \Omega_i I_i$ where I_i's are control images and Ω_i's weight images that satisfy partition of unity, i.e $\sum_{i=0}^{M} \Omega_i = \mathbf{1}$, where $\mathbf{1}$ is a white image and I is the final rendering (see figure 3.26).

Figure 3.26 provides an illustration demonstrates how the final non-specular color of foreground water is computed as weighted average of control images provided by front-end shaders as $I = I_0\Omega_0 + I_1\Omega_1$. Note in our case, we choose the non-specular color from front-end shader as control parameter to simplify our computation, denoted as weight image Ω_0. The other one is well defined as $\Omega_1 = \mathbf{1} - \Omega_0$ since partition of unity $\Omega_0 + \Omega_1 = \mathbf{1}$ must be satisfied for a legal operation. In this case, obtaining $\mathbf{1} - \Omega_0$ is easy, which is just the inverse image of Ω_0. Image I_0 and I_1 is the color gradient obtained in the previous subsection.

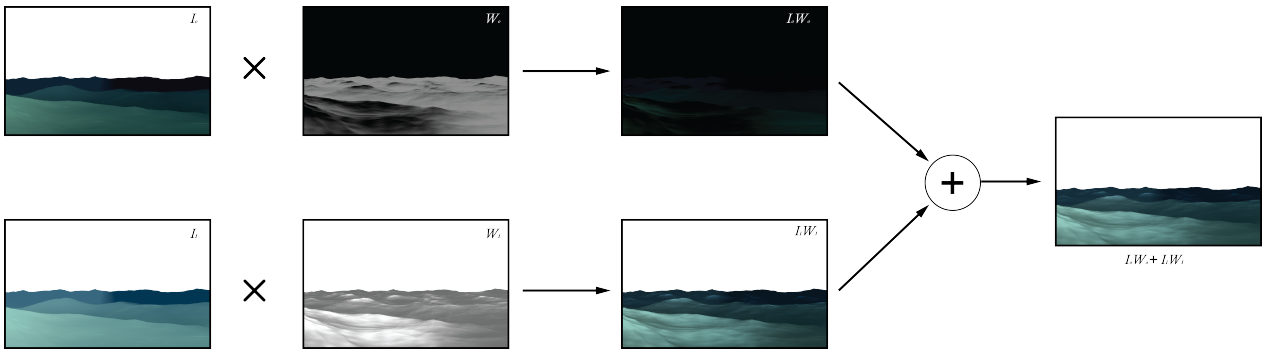


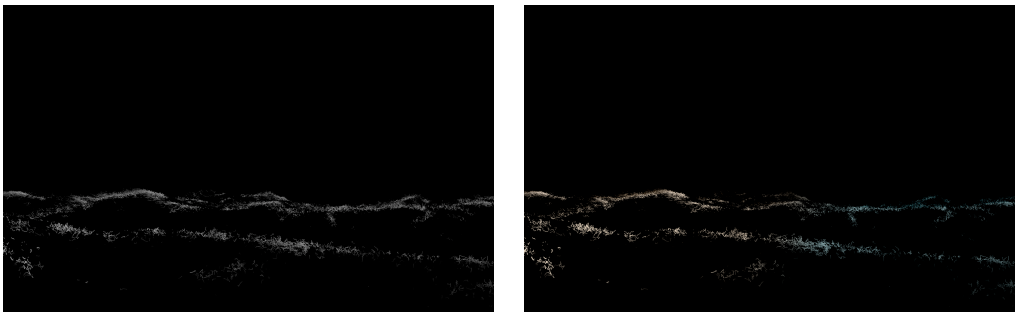
Figure 3.26: An example demonstrating the concept of control and weight images. I_0 and I_1 are control images, Ω_0 and Ω_1 are weight images that satisfy partition of unity. $I = I_0\Omega_0 + I_1\Omega_1$ is the final rendering obtained by weighted average of two control images.

3.2.4.4 Hierarchy for Handling Multiple Parameters

As we have discussed earlier, we can obtain five control parameters from front-end shaders. They are non-specular, specular, fresnel, spray and depth. Among them, the depth parameter is used to create texture images while other control parameters will be used to obtain our final water color. In order to manage these control parameters, we need to create a hierarchy of barycentric operations. In practice, this hierarchy starts from the most important parameter, non-specular color. As an intermediate result, image I_1 is obtained as a weighted average of non-specular and specular. The weighted image for mixing specular reflection and non-specular color is the fresnel parameter since fresnel describes the ratio of specular reflection that can be received by the viewer. Therefore, our first layer of barycentric operation can be described as

$$I_1 = (1 - \Omega_{fresnel})I_{diffuse} + \Omega_{fresnel}I_{specular}$$

On top of I_1, we mix between spray parameter and I_1 to get the final water color. Hence the equation of final image is extended into the form



(a) front-end shading result (b) back-end shading result

Figure 3.27: Spray rendering result

$$I = (1 - \Omega_{spray})((1 - \Omega_{fresnel})I_{diffuse} + \Omega_{fresnel}I_{specular}) + \Omega_{spray}I_{spray}$$

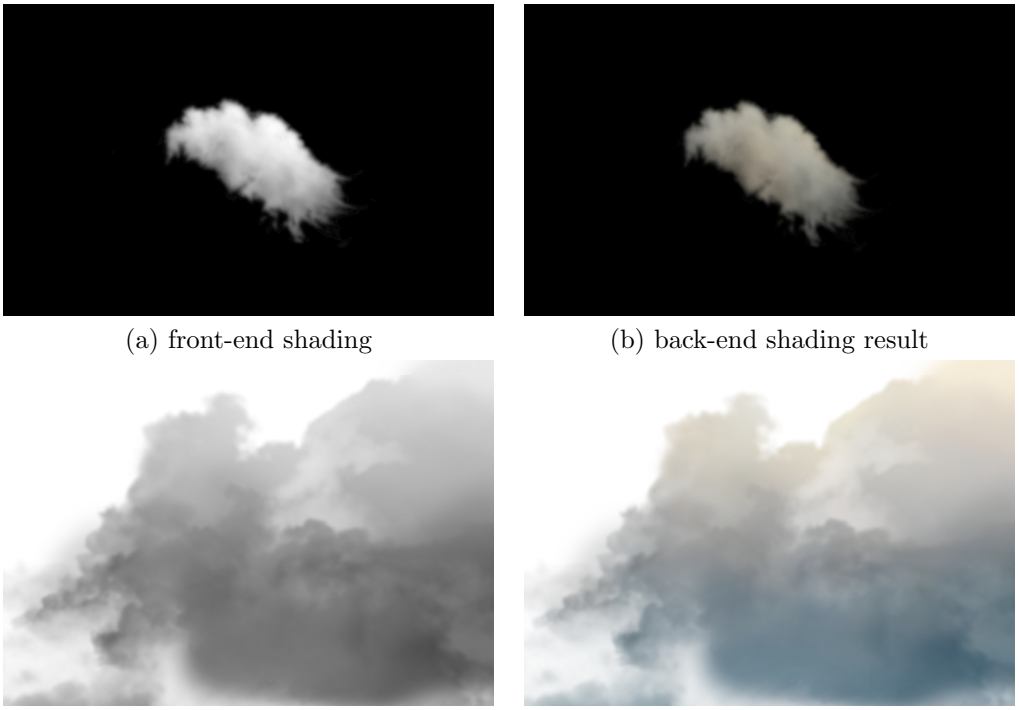
where Ω_{spray} is the alpha channel of spray rendering.

3.2.4.5 Spray

The spray effect is simulated through a customised particle system. To mimic brush strokes as shown in the painting, we pre-model several brush geometries and stamp each of the geometries to a certain particle when that particle is created. The front-end shader of particle geometries is a contant white shader while in the back-end, a similar strategy is applied as we sample both brightest and darkest colors from the water spray in the painting and use those colors as control colors to obtain the final spray color. The result of water spray rendering is shown in figure 3.27

3.2.4.6 Cloud

In our production, we use the default volumetric rendering tool in the commercial software package to obtain the front-end shading result of the cloud. Our final sky background is obtained by combining different layers of cloud rendering. Each piece



(a) front-end shading (b) back-end shading result

(c) front-end shading result (d) back-end shading result

Figure 3.28: Figure shows two groups of cloud rendering results

of cloud rendering is passed into back-end shaders to be processed with the corresponding bright and dark colors according to its position in image space. Figure 3.28 shows the rendering result of cloud.

4. IMPLEMENTATION AND PROCESS

In this chapter, we will talk about how we implement the wave simulation and shading architecture by using commercial software packages.

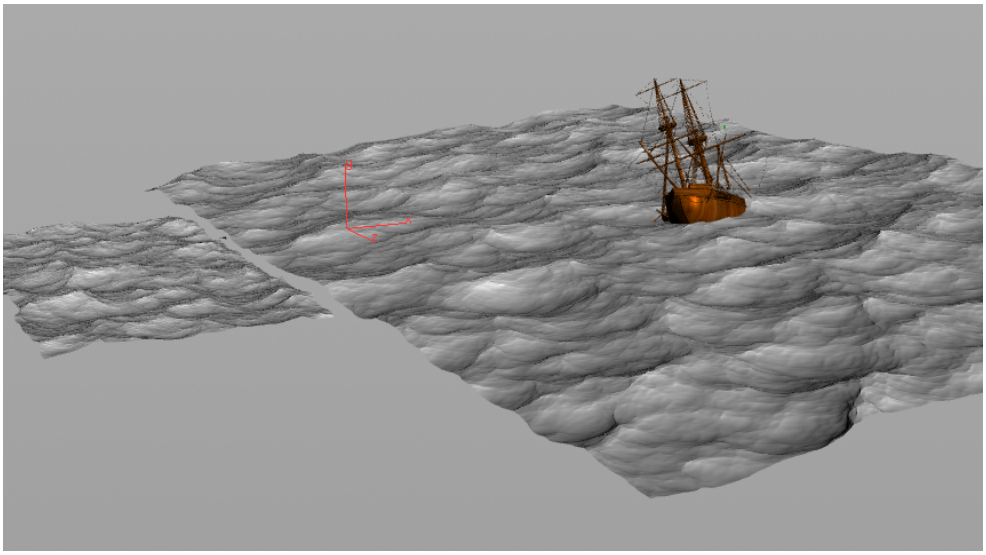
4.1 Simulation and Control

We first implement a 2D Gerstner wave simulator for a prototyping test. We use Java and Processing as our platform for this purpose. The display and user interface is rather primitive, since our main goal is to test sharpness coefficient $s = ka$. The sharpness coefficient has valid ranges between 0 and 1 and for values exceeding 1, our simulator generates a self-intersected wave curve.

The 3D wave simulator is coded in C++ with Houdini Development Kit and we use GCC as our standard compiler. The architecture of software respects the principles of object-oriented design(OOD), with one class devoted as a pure interface to communicate with Houdini and another class responsible for wave pose computation. In order to boost our simulation speed, the Intel TBB library is employed for multi-treading capability. For the design of the user interface, we do not specify any scene description format since each parameter is treated as data input directly from the default Houdini UI.

4.2 Ship Modeling and Layout

Our 3D ship model is provided by the Nautical Archaeology Program at Texas A&M University. The original model is built in the commercial 3D package of Autodesk Maya. A simple texture is created and assigned to the ship model to improve the look. In order to match the feel of depth as shown in Aivazovsky's painting, we place two separate pieces of ocean surface into 3D scene(see figure 4.1). However,



(a) Layout

Figure 4.1: 3D scene layout

for the depth of front-end shader, we compress the depth information of two pieces of parametric surfaces into a single image to simplify the process of managing data files.

4.3 Shader Implementation and Management

Our front-end shader is implemented using the Houdini VEX Shading language. For each front-end shader, we create a different set of virtual CG lights to illuminate the ocean surface according to our needs. The product of each front-end shader is an image file that only contains information produced by that shader. Multiple render passes created during the front-end shading stage also give us a lot of flexibilty in the back-end shading process.

Back-end shaders are implemented in Foundry Nuke. Foundry Nuke is a digital compositing software and assembles all the control parameters together by using barycentric operations over texture images. At this stage, the user will still have artistic control over the final result by adjusting parameters in 2D image process-

ing algorithms. Another reason we choose to use Nuke for implementing back-end shaders is that Nuke is a node-based digital compositing software. It is straightforward to construct a tree-style back-end shader in Nuke as discussed in the previous chapter.

4.4 Result

The final result of this thesis is a short animation with the style consistent with our primary reference. Figure 4.2 shows a comparison between the original artwork and our final animation.

(a) reference painting

(b) rendering result

Figure 4.2: Comparison between original art work and our final rendering

5. CONCLUSION AND FUTURE WORK

The main goal of this research lies in two aspects. First is to provide a tool for creating procedural ocean wave animation that can simulate wave shapes observed in both nature and artwork. Second is to propose a shading architecture that addresses the issues of mathematical inconsistency and conceptual inconsistency which are brought by the practical problem of matching digital work with traditional art.

We spanned 2D Gerstner wave equation to three dimensional space and re-designed the representation of parameters in wave equation. Wave shape is governed by poses given wavelength, time, amplitude and sharpness parameters. The final displacement is represented as a weighted sum of independent waves in order to bring randomness and complexity into the wave simulation.

The shading architecture that is developed for matching traditional artwork is proved to be effective in handling inconsistencies. We first created several independent front-end shaders based on our need of shading components. This allows for a great deal of control, however it can be slow at times. The resulting colors of back-end shaders are more predictable than using the classical shading approach since barycentric operations are closed in positive real numbers. The illumination has proved to be irrelevant to our shading approach. Even if we use a weight image which is completely irrelevant to the geometry, our final result will still be acceptable by using the correct control colors.

5.1 Future Work

Opportunities for future works could include using a better model to depict wave shapes. In this research study, we use the Gerstner Wave model. It is flexible and

relatively easy to implement, but Gestner Wave Model is not capable of simulating breaking waves that are often observed in seascape paintings.

Another line of research could be spanned to advanced rendering techniques, such as using brush stroke effects to express water sprays in the oil painting. Currently in this thesis, we use a particle system to mimic water spray.

REFERENCES

[1] Ivan Konstantinovich Aivazovsky and Hovhannes Aivazian. Biography of ivan aivazovsky. 1881.

[2] Robert L Cook. Shade trees. *ACM Siggraph Computer Graphics*, 18(3):223–231, 1984.

[3] Gábor Czédli and Anna B Romanowska. An algebraic closure for barycentric algebras and convex sets. *Algebra Universalis*, 68(1-2):111–143, 2012.

[4] Sarah Beth Eisinger. Applying hand-drawn effects design principles to the creation of 3d effects. Master's thesis, Texas A&M University, College Station, 2013.

[5] Douglas Enright, Ronald Fedkiw, Joel Ferziger, and Ian Mitchell. A hybrid particle level set method for improved interface capturing. *Journal of Computational Physics*, 183(1):83–116, 2002.

[6] Jay Allen Faulkner. Beauty waves: an artistic representation of ocean waves using bezier curves. Master's thesis, Texas A&M University, College Station, 2007.

[7] Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, pages 23–30. ACM, 2001.

[8] Nick Foster and Dimitri Metaxas. Realistic animation of liquids. *Graphical Models And Image Processing*, 58(5):471–483, 1996.

[9] Alain Fournier and William T Reeves. A simple model of ocean waves. In *ACM Siggraph Computer Graphics*, volume 20, pages 75–84. ACM, 1986.

[10] Franz Gerstner. Theorie der wellen. *Annalen der Physik*, 32(8):412–445, 1809.

[11] Francis H Harlow, J Eddie Welch, et al. Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *Physics of fluids*, 8(12):2182, 1965.

[12] Damien Hinsinger, Fabrice Neyret, and Marie-Paule Cani. Interactive animation of ocean waves. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 161–166. ACM, 2002.

[13] K. Hokusai. The great wave at kanagawa (from a series of thirty- six views from mount fuji). Polychrome Woodblock Print, Ink and Color on Paper, ca. 1830-32. H. O. Havemeyer Collection (JP1847), Accessed October 7, 2006 from http://www.metmuseum.org/Works of Art.

[14] K. Hokusai. The great wave at kanagawa (from a series of thirty- six views from mount fuji). Polychrome Woodblock Print, Ink and Color on Paper, ca. 1830-32. H. O. Havemeyer Collection (JP1847), Accessed October 7, 2006 from http://www.metmuseum.org/Works of Art.

[15] Geoffrey Irving, Eran Guendelman, Frank Losasso, and Ronald Fedkiw. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. *ACM Transactions on Graphics (TOG)*, 25(3):805–811, 2006.

[16] Michael Kass and Gavin Miller. Rapid, stable fluid dynamics for computer graphics. In *ACM SIGGRAPH Computer Graphics*, volume 24, pages 49–57. ACM, 1990.

[17] Jonathan Kiker. Using local information for compositing cg into traditional art. Master's thesis, Texas A&M University, College Station, 2009.

[18] Klett, Eberhard Witwe, Detleffsen, Christoph Peter, et al. *IH Lambert... Photometria sive de mensura et gradibus luminis, colorum et umbrae.* sumptibus viduae Eberhardi Klett, 1760.

[19] Gary A Mastin, Peter A Watterberg, and John F Mareda. Fourier synthesis of ocean scenes. *Computer Graphics and Applications, IEEE*, 7(3):16–23, 1987.

[20] Viorel Mihalef, Dimitris Metaxas, and Mark Sussman. Animation and control of breaking waves. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 315–324. Eurographics Association, 2004.

[21] Joe J Monaghan. Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics*, 30:543–574, 1992.

[22] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 154–159. Eurographics Association, 2003.

[23] Darwyn R Peachey. Modeling waves and surf. In *ACM SIGGRAPH Computer Graphics*, volume 20, pages 65–74. ACM, 1986.

[24] AB Romanowska and JDH Smith. On the structure of barycentric algebras. *Houston Journal of Mathematics*, 16(3):431–448, 1990.

[25] Christophe Schlick. An inexpensive brdf model for physically-based rendering. In *Computer Graphics Forum*, volume 13, pages 233–246, 1994.

[26] Jos Stam. Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128. ACM Press/Addison-Wesley Publishing Co., 1999.

[27] Jerry Tessendorf et al. Simulating ocean water. *Simulating Nature: Realistic and Interactive Techniques. SIGGRAPH*, 1, 2001.

[28] Pauline Y Ts'o and Brian A Barsky. Modeling and rendering waves: wave-tracing using beta-splines and reflective and refractive texture mapping. *ACM Transactions on Graphics (TOG)*, 6(3):191–214, 1987.

[29] Cem Yuksel, Donald H House, and John Keyser. Wave particles. In *ACM Transactions on Graphics (TOG)*, volume 26, page 99. ACM, 2007.