



HOKKAIDO UNIVERSITY

Title	A Study on Efficient Rendering of Reflections on Complex Surfaces
Author(s)	Podee, Namo
Degree Grantor	北海道大学
Degree Name	博士(情報科学)
Dissertation Number	甲第14626号
Issue Date	2021-06-30
DOI	https://doi.org/10.14943/doctoral.k14626
Doc URL	https://hdl.handle.net/2115/82421
Type	doctoral thesis
File Information	Podee_Namo.pdf



A Study on Efficient Rendering of Reflections on Complex Surfaces

Namo Podee

A thesis presented for the degree of
Doctor of Information Science and Technology

The Graduate School of Information Science and Technology

Hokkaido University

March 2021

Thesis Supervisor: Prof. Yoshinori Dobashi

A Study on Efficient Rendering of Reflections on Complex Surfaces

Namo Podee

Abstract

Computer graphics has become a popular technique and is widely used in many applications including video games, virtual reality, and architectural designs. One of the important goals in these applications is to synthesize realistic images efficiently. Reflections on reflective surfaces with highly complexity structures, such as sparkles of the sun dancing on ocean surfaces or glints of street lights catching on wet pavements, are tiny but essential to the realism and quality of the synthetic images. The reflections visually describe the surfaces' detail and material to the viewer. However, these reflections are difficult to render accurately in a limited time. In Chapter 1, I explain the fundamentals and the relation between computer graphics and complex surfaces. Then, I discuss our objective and challenge in rendering the complex surfaces. Next, I present our approaches and their overall contributions. At the end of the chapter, I briefly explain my thesis outline and its structure, including each chapter's detail.

This study is relying on many rendering and computing architectures. First, the rasterization-based rendering method excels at interactive rendering. Second, general-purpose computing on GPU is a high-performance parallel computing method. Last, the illumination model defines the calculation of interactions between light and surfaces. In Chapter 2, I explain this background knowledge in detail.

The computer graphics research field has a long history of progressing in both result quality and process performance. There are also many research focusing on solving relevant topics to my work. In Chapter 3, I review the history of computer graphics research and the relevant research in multiple categories: reflection on smooth surfaces, glint and caustic, and anti-aliasing researches. I discuss the contributions and limitations of each study in detail in this chapter.

Real-time glittery surface rendering is my first approach to solve complex surface

reflection rendering in real-time. This work focuses on reproducing glint and sparkle on glittery reflective surfaces in an efficient manner. This work’s main challenge is the fact that a reflection in each pixel is too complicated to be smoothly approximated. The proposed method efficiently renders the reflections from glittery surfaces on a static model and supports multiple light source types. The method stores numerous tiny surface orientations in one variable using slope space categorization and bit field variables. In Chapter 4, I explain the method in detail, including its result and limitation.

Real-time spatio-temporal reflection anti-aliasing is my second approach to the problem. This work aims at creating high-quality reflections of the Sun and Moon on moving water surfaces. The main challenges are complicated reflections, and its movement causes unstable reflections. The proposed method solves the problems by efficiently integrating reflections in-between the rendering frames to stabilize the result. The technique also approximates the water surfaces, that are too complicated to be rendered, by a smooth function for an efficient reflection computation. There are also two follow-up methods: adaptive supersampling and micro-reflection rendering. Adaptive supersampling improves my anti-aliasing method by preserving the sub-pixel geometric detail removed by the smooth function. Micro-reflection rendering increases the reflection quality of a simple surface, which the above process did not consider yet. In Chapter 5, I explain the primary method and the follow-up methods in detail. Then, I show its result and discuss its contribution and limitation.

My proposed methods successfully generate reflections on glittery surfaces and moving water surfaces in real-time. However, there are some limitations and undiscovered potential in our techniques. In Chapter 6, I summarize the contributions and limitations of the proposed methods then explain what I plan to improve in the future.

Contents

1	Introduction	1
1.1	Fundamental Terminology	1
1.1.1	Computer Graphics	2
1.1.2	Photorealistic Rendering	3
1.1.3	Offline and Real-time Rendering	5
1.1.4	Rasterization Based Rendering	5
1.1.5	Local and Global Illumination	5
1.1.6	Normal Distribution Function	7
1.1.7	Complex Surface	7
1.2	Our Objective and Challenge	7
1.3	Our Approaches and Contribution	9
1.4	Organization	11
2	Background Knowledge	12
2.1	Rasterization Based Rendering	12
2.1.1	Graphics Rendering Pipeline	12
2.1.2	Anti-aliasing	14
2.2	General-purpose computing on GPU	14
2.3	Illumination Model	15
3	Related Work	16
3.1	History of Computer Graphics Rendering Research	16
3.1.1	Photorealistic Rendering Research	17
3.1.2	Real-time Rendering Research	17
3.1.3	Efficient Path Tracing	19

3.2	Relevant Research on Complex Surface Reflection	21
3.2.1	Reflection Rendering	22
3.2.2	Glint and Caustic Rendering	23
3.2.3	Screen Space Anti-aliasing	24
3.2.4	Analytic Anti-aliasing	26
4	Real-time Glittery Surface Rendering	28
4.1	Proposed Method	29
4.1.1	Flake Maps	29
4.1.2	Construction of Flake Maps	30
4.1.3	Rendering	30
4.2	Results	31
4.3	Discussion	32
5	Real-time Spatio-temporal Reflection Anti-aliasing	34
5.1	Proposed Method	36
5.1.1	Spatial-aliasing detection	36
5.1.2	Temporal-aliasing wave rendering	38
5.1.3	Spatio-temporal-aliasing wave rendering	39
5.1.4	Circular Gaussian approximation	44
5.1.5	Elliptical Gaussian approximation	46
5.1.6	Aliasing wave transition	49
5.2	Implementation	50
5.3	Results	51
5.4	Discussion	53
5.5	Real-time Adaptive Supersampling	61
5.5.1	Proposed Method	61
5.5.2	Results	65
5.5.3	Discussion	66
5.6	Real-time Micro-specular-highlight Rendering	68
5.6.1	Proposed Method	69
5.6.2	Results	72
5.6.3	Discussion	75

6 Conclusion and Future Work	76
6.1 Conclusion	76
6.2 Future Work	78
Appendix A Real-time Spatio-temporal Reflection Anti-aliasing	92
A.1 Per-pixel NDF	92
A.2 RDF approximation accuracy	93
A.3 Elliptical Gaussian approximation	94
Achievement	96
Acknowledgements	98

List of Figures

1.1	The result of the photorealistic rendering of the virtual classroom. The classroom model by courtesy of Christophe Seux	4
1.2	Comparisons between local and global illumination of Frank Meinl's Crytek Sponza scene.	6
1.3	An example of complex surface reflection. The image contains reflec- tions of multiple streetlights on water surfaces.	8
1.4	These images represent the NDF of three surface types. We show NDF with an image, with the horizon and vertical axis represents the polar and azimuthal angle of directions, respectively. We plot a higher probability direction with brighter color and vice versa. . . .	8
2.1	A simplified flowchart of graphics rendering pipeline.	13
3.1	Progression of adaptive sampling and reconstruction research.	21
3.2	A part of reflection rendering research progression.	22
3.3	A part of glint and caustic rendering research progression.	23
3.4	A part of screen space anti-aliasing research progression.	25
3.5	A part of analytic anti-aliasing research progression.	27
4.1	Left and center figures show front and back of the hemisphere, which is divided into evenly 16 spaces. Right figure is an example of 16 bit field that represent 4 vectors, which are pointed to 13, 14, 15 and 16 labeled spaces, we store these vector bit field as 0000000000001111. .	29
4.2	Flakes map generated by using a normal map as a reference. Flakes map mip level 0 is generated directly from normal map data then generate higher mip level data from the current level.	30

4.3	These two scenes have the same directional light and the same four flakes in each pixel. The left image is without any jittering; the left ball focuses all of its four normals onto the center of 4 labeled spaces, which result in 4 strong specular lights. The right image is rendered with a random offset; it distributed four normal vectors in each labeled space, create the glittery and uniformly lighting.	31
4.4	The result image of our glittery rendering method. The high heel is rendered in real-time with randomly generated flakes map. The result show glittery lighting on its body as it is being shined by a directional light and a light from environment map.	32
4.5	Rendered images from different flakes maps: A use 4 vectors on the top of hemisphere flakes map, B use A flakes map with one more random vector, C is generated from a metallic flakes normal map, D is generated from an isotropic noise normal map.	33
5.1	The problem in rendering a reflection on dynamic surfaces. Typically a single reflection ray (solid black) is generated for each frame, but it often misses the light source, which causes temporal aliasing. We address this problem by using the ray (dashed black) shot in the previous frame. The light source is sampled by using multiple interpolated rays (red) between them to reduce the temporal aliasing.	35
5.2	The outline of our methodology.	37
5.3	The projected wave is aligned to a 2x2 screen. If any wave in any projected direction has higher spatial frequency or lower wavelength than this wave then it will cause aliasing.	38
5.4	Our method uses a simple geometric analysis to find the contribution of light between the current reflection and the previous reflection. . .	39
5.5	The outline of our spatio temporal waves rendering method.	40
5.6	Our NDF is a normal distribution function of spatial aliasing waves. Our RDF is a reflection distribution function of the NDF. It is defined on a light plane perpendicular to the direction to the center of the light source.	41
5.7	Transformation from NDF to RDF.	43

5.8	Precomputation for the line integral, which is an integral of a convolution of the light disk and the circular Gaussian that is moving on the line segment.	45
5.9	We further approximate our RDF, which is defined as an elliptical Gaussian distribution, as a combination of multiple weighted circular Gaussian distributions. In this example, we approximate RDF as 3-weighted circular Gaussian distributions.	47
5.10	A, B, and C circular Gaussian kernels are used to approximate the RDF. Each circular Gaussian kernel has its own integral segment, which can be calculated by offsetting the original integral segment with the vector from the center of the RDF to the circular Gaussian kernel.	48
5.11	Comparison with/without transition regions.	50
5.12	The preview of our four experiment scenes.	51
5.13	Here is the reflection result from the simple wave scene. The reference uses 8 spatial samples and 4 temporal samples.	54
5.14	Here is the reflection result from the smooth detail wave scene. The reference uses 4 spatial samples and 8 temporal samples.	55
5.15	Here is the reflection result from the medium detail wave scene. The reference uses 8 spatial samples and 8 temporal samples.	56
5.16	Here is the reflection result from the high detail wave scene. The reference uses 8 spatial samples and 8 temporal samples.	57
5.17	Here is the comparison of our approximation with different numbers of circular Gaussian kernels (without a fallback method), the Feline method, and the convolution. The reference uses 8 spatial samples. The RMSE of (b) to (f) are 0.0209, 0.0195, 0.0181, 0.0191, and 0.0183 respectively. (d) has the best performance for capturing and ability to compute time integral but (f) is better at capturing the reflection near the horizon. Thus, our method uses both (d) and (f).	58
5.18	A bar chart of RMSE comparison data.	59
5.19	A bar chart of our survey data.	59
5.20	The overview of our method.	62

5.21	The comparison result from the sunset scene.	66
5.22	The internal result of our method.	67
5.23	An overview of our method.	70
5.24	A curved surface patch inside a pixel and four samples of reflection vectors at the corners.	71
5.25	Definitions of the light plane and the reflection space.	71
5.26	The comparison result in the ocean scene.	74
5.27	The comparison result in the rings scene.	75
A.1	Wave beginning and wave ending in the image is spatial position of p_1 and p_2 respectively.	92
A.2	The relative error of NDF to RDF transformation. Height and color shows the relative error while the other axis represent NDF variance in X and Z axis. Color from blue to green to yellow represents the relative error from low to high.	93
A.3	The error ϵ (Eq. A.5) for three circular Gaussian kernels. The hori- zontal axis is the major axis standard deviation σ_y	95

List of Tables

5.1	RMSE of the result of each method when compared to the references.	53
5.2	The computational time comparison.	66
5.3	RMSE of the result of each method when compared to the references.	73

Chapter 1

Introduction

Computer graphics is one of the most critical technologies for humankind. It enables us intuitive communication with a machine, which serves us in numerous ways. Photorealistic rendering is one of the essential tools in computer graphics. It can visually express data as a photorealistic image or animation, which is instinctively understandable by us. The more we understand, the better we can make our decision based on the data. However, to create a photorealistic result, we need to spend the machine's expensive computational cost to simulate countless physics phenomena. We need to calculate how light interacts with surfaces and enters our eyes, which becomes more costly for more complex light and surface. High computational cost means less interactivity for the user, which is crucial for real-time application. However, complex lights and surfaces can significantly increase the fidelity of the result.

In this research, we study a method that efficiently renders reflections of complex surfaces. We approach the problem by improving the interactive method's quality to reproduce complex surface reflections in real-time.

This chapter will introduce some fundamental terminologies, our objective, challenge, our approaches, our contribution, and this thesis organization.

1.1 Fundamental Terminology

Before we talk about our study, we need to explain the essential keywords of computer graphics related to our work.

1.1.1 Computer Graphics

Computer graphics is the methodology of creating, manipulating, visualizing, and analyzing virtual content. It creates interactions with a virtual world, which is data inside a computer. For simplicity, we divide computer graphics into five major subfields: modeling, animation, rendering, imaging, and system and interface. Firstly, the modeling subfield studies virtual object creation methods by using user-interface and mathematical methods. Secondly, the animation subfield explores virtual object movement by creating a motion from changing a virtual object's position, rotation, scale, and distortion according to time parameters. Thirdly, the rendering subfield studies image synthesis methods. The method creates an image of virtual objects from geometries, animations, lights, cameras, and other necessary data. Fourthly, the imaging subfield studies creation and manipulation of digital images. It concerns a method to physically acquire an image and digitally process a 2-dimensional image. Lastly, the system and interface subfield studies processors, input devices, and file formats.

Computer graphics plays a vital role in many industries, from movie, game, and product design to health care. It is needed in many industries because it is a human tool to create digital products from data. Movie, television, and commercial sector found their usage in generating virtual objects and blending them with real-life footage to create a scene that is impossible or difficult to arrange in real life. This illusion synthesis also became the primary motivation that made many breakthrough pieces of research in computer graphics. The video game industry had been born after advancements in computer graphics; the improvement of rendering methods by a revolutionary graphics processor, which can generate an image of virtual objects within a fraction of a second, enables the real-time interaction between humans and machines. In other words, the user can send input and see a result from the machine immediately. This real-time interaction permits immersive game programs to be processed on the machine. Product designers used to draw their designs on paper and then build prototypes to see and examine them. If there are any changes, the designers have to redo the whole process again, which takes a long time. Computer graphics helps these designers to be able to create and simultaneously see their products. The application improves their productivity by significantly reducing their

design iteration time. Doctors traditionally analyze CT-scans by reading images of slices of an object one by one, which is very unintuitive and difficult to understand. Computer graphics can render images of a 3-dimensional volumetric model of a scanned object so that doctors be able to freely manipulate the model and even simulate a surgery within a computer. This method helps the doctors gain more understanding of inspected data and then treat their patients correctly.

1.1.2 Photorealistic Rendering

Photorealistic rendering is one of the most active research areas within the rendering sub-field of computer graphics. It studies methods to generate an image of virtual objects that are as similar as possible to what humans have seen in real-life. Many industries that need to reproduce real-life aesthetics in a virtual world, such as using computer-generated imagery in the film industry, must rely on photorealistic rendering. Figure 1.1 shows an example of photorealistic rendering.

The history of photorealistic rendering starts from scan-line rendering [82] in 1967; the method that can determine any visible surfaces and draw them on a display monitor. However, it can only produce a simple graphical image. Even with state-of-the-art techniques based on the scan-line method, there are still many limitations that prevent its result from reaching the exact copy of the real-life visual quality. Later in 1982, the ray casting method [60] was introduced. It is a method that casts rays from a virtual camera and checks intersections between rays and surfaces. If there is an intersection, then there is a visible surface. This method cannot produce a photorealistic image, but it became a fundamental of another technique called a ray tracing method [79]. The ray tracing method uses the ray casting method to determine visible surfaces. It then casts another set of rays from the intersected point to light sources to create paths between a virtual camera or eyes to any light sources. This algorithm imitates real-life light interactions. In real-life, a light source emits numerous photons from itself. If emitted photons hit a surface, then it might have reflected and hit another surface. This bouncing happened as photons travel across many objects and finally enter our eyes, which results in our visual perception. The algorithm tries to search for these massive photon paths in a virtual world and create an image just like how we perceive the world with our

eyes. It can be called a light transport simulation. It can produce a high-quality photorealistic image at the expense of computational time since there is an almost infinite number of photon paths that need to be calculated. The quality of the result depends on the number of processed paths.

The ray tracing based method is not suitable for any interactive application, where computational time is limited. On the other hand, a scan-line based method or a rasterization based method thrives in the field of an interactive application because of its performance despite the lower quality. Fortunately, various light transport approximation techniques can improve the realism of the result at the expense of more computational time. Thus, the photorealistic rendering problem is a trade-off between the quality of a result and the computational time.

Light transport problems were practically impossible to be solved interactively because of their complexity. However, the advances made in general-purpose graphics processing units (GPGPU) now mean a massive amount of threads on a GPU can solve these problems in interactive time.



Figure 1.1: The result of the photorealistic rendering of the virtual classroom. The classroom model by courtesy of Christophe Seux

1.1.3 Offline and Real-time Rendering

There are mainly two types of rendering when categorized by its application: offline rendering and real-time rendering.

Offline rendering is a rendering that stores the rendering result for later usage. An example of this is a rendering in film, which spends a long computation time to create animation data as a video file.

Real-time rendering takes an input from the user and show the rendering result within a few tens of millisecond to keep interactivity between the user and the application. An example of this would be a video game, where the user pushes a button and the game has to react to the input and show a result on the screen at that instant. If the game takes too long to show the result, the user will feel a delay between the input and the screen; this will cause an unpleasant feeling due to the interrupted flow of the action and reaction.

The computational time limitation depends on the type of application. Offline rendering doesn't need interactivity; thus, it can utilize a ray tracing based method to spend an expensive computational time to create a high-quality result. The real-time rendering needs to compute within a fraction of a second, which is what a rasterization based method excels at interactivity. Its application type can decide the trade-off between computational time and quality.

1.1.4 Rasterization Based Rendering

Rasterization is a method of converting vector data shape to a series of pixels of that shape. Rasterization rendering utilizes the conversion to convert 3D polygons to a series of pixels on the 2D screen. The rasterization method is very efficient. Thus, it is the most popular choice for any real-time rendering application. We will explain more in the next chapter.

1.1.5 Local and Global Illumination

Illumination is a result of an interaction between light and surface, which is what we study in computer graphics. It is usually categorized into two groups due to its complexity: local illumination and global illumination.

Local illumination is a contribution of direct light on a surface. Direct light comes directly from the light source; it doesn't hit any other surface, so its property doesn't change from its source. This lighting type is easier to compute than indirect lighting, which requires higher computational cost; thus, both offline and real-time rendering always uses it.

Global illumination is a contribution of both direct and indirect light on a surface. Indirect light is a light that hits and bounces off other surfaces before illuminate on the target surface. Indirect light usually has less contribution than direct light because of energy absorption by the other surfaces, but it is an essential factor for the result's realism. Unfortunately, global illumination or specifically indirect light has a high computational cost because we need to consider the light bouncing off the other surfaces, which increases the dimensional of the computation. Thus it is challenging to use global illumination in real-time rendering without a heavy approximation or pre-computation techniques.

Figure 1.2 shows examples of local and global illumination effect.

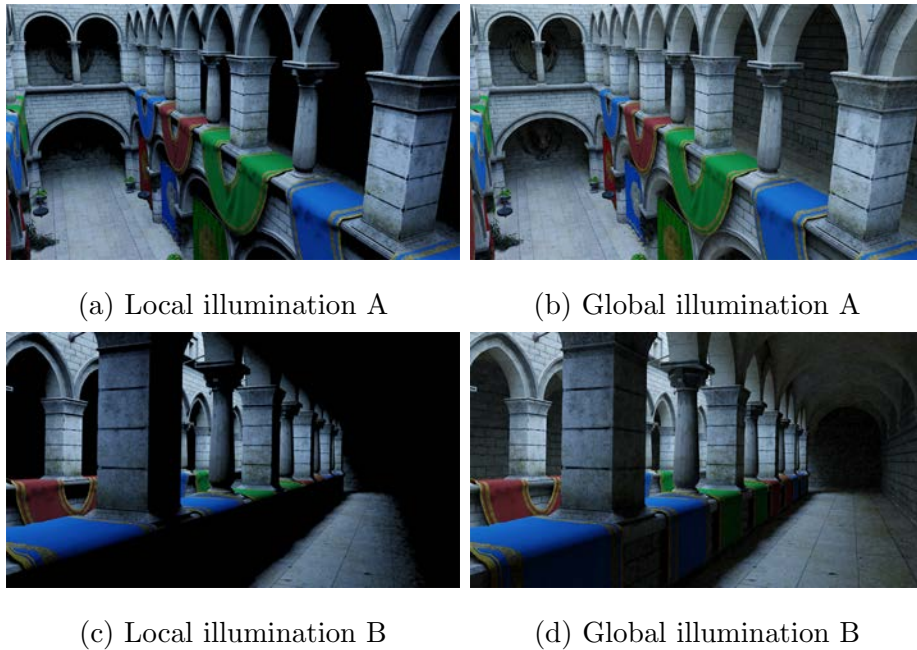


Figure 1.2: Comparisons between local and global illumination of Frank Meinl's Crytek Sponza scene.

1.1.6 Normal Distribution Function

Normal distribution function (NDF) is a function that describes the distribution of orientation of a tiny surface (microfacet) within one area, which usually is an area of a surface within a pixel. It can also be looked at as a probability function that defines the probability of each surface orientation within an area. Computer graphics relies on NDF to define a geometric structure of a surface, which is too tiny to represent with polygons, then use it to compute the surface illumination.

1.1.7 Complex Surface

We define a complex surface as a surface with an NDF that is difficult to be approximated by a smooth function. Figure 1.3 shows an example of complex surface reflection. The NDF is smooth when the material of the surface has some roughness, or the normal variation is uniformly distributed. A smooth function like the Gaussian function can easily fit the smooth NDF. However, if the surface has low roughness and non uniformly distributed NDF, its NDF will have a unique distribution shape. The unique distribution shape is difficult to be approximated by a smooth function and thus more troublesome to process. The conventional method encourages the use of a smooth function to approximate the actual NDF because of its simple computation and noiseless result. Figure 1.4 shows examples of multiple type of NDF.

1.2 Our Objective and Challenge

Our study aims to reproduce the photorealistic reflection of complex surfaces in real-time. Reflection of surfaces is one of the most important aspects of photorealistic rendering. It directly contributes to the realism and quality of the rendering result. However, the reflection of the complex surface still lacks in real-time rendering application. Despite the fact that there are many complex surfaces in our real-world such as an ocean, metallic flake paint, and street on a rainy day. (See Fig. 1.3) By accurately reproducing the effect, we can improve the quality and realism of a real-time rendering, including its application. Our study should enable new kinds of realistic material to be used in real-time rendering applications.



Figure 1.3: An example of complex surface reflection. The image contains reflections of multiple streetlights on water surfaces.

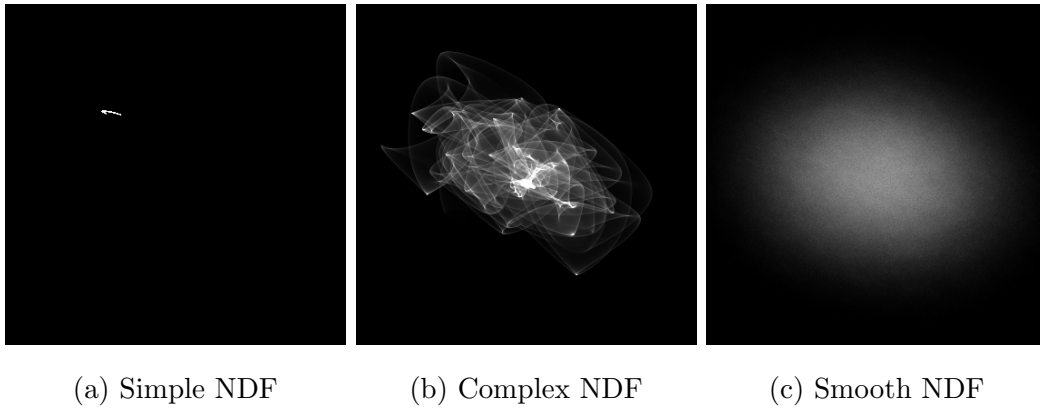


Figure 1.4: These images represent the NDF of three surface types. We show NDF with an image, with the horizon and vertical axis represents the polar and azimuthal angle of directions, respectively. We plot a higher probability direction with brighter color and vice versa.

The challenge of rendering complex surfaces is that we cannot use the conventional method to compute an accurate reflection without losing the detail. The traditional method efficiently computes a reflection by approximating a surface NDF as a smooth function, which can be used in multiple ways to find the surface's illumination or reflection.

For example, we can find how much energy of a point light is illuminated on a reflective surface by looking at how much of an area on the surface perfectly reflects the light into our eye. A ratio of any specific orientation of a surface is defined by NDF. Thus we can just use its value at that perfect reflected orientation; however, if we increase the difficulty by considering an area light instead of a point light, which is not present in reality. The area light has its own surface that emits light energy onto the surface. We can find its accurate reflection by solving an integral of each point on the light source's surface, similar to how we did in the point light case. This integration can be computationally expensive if it is done naively. However, we can find this integral more efficiently or analytically if the NDF is a smooth function, which is convenient to process.

Unfortunately, a complex surface NDF cannot be approximated as a smooth function because of its sharp and complex feature. These sharp and complex features are the source of glints and sparkles that create a beautiful look of complex surface reflection. Thus we cannot ignore them and forcefully use a smooth function approximation. The smooth function approximation will remove all of these effects and details from the surfaces. We can find an accurate reflection by naively solving the integral, but it is inefficient and too expensive for real-time application. Hence, we need to come up with a new method that can solve this problem efficiently.

1.3 Our Approaches and Contribution

We approach the problem of complex surface reflection rendering by introducing new rendering methods based on rasterization rendering. The rasterization rendering method is very efficient and excels at real-time rendering; however, it can't reproduce the reflection of complex surfaces without using expensive computational time by using a naive integration approach yet.

Our study expands the possibility of rasterization rendering with three different rendering methods: glittery surface rendering, spatio-temporal reflection anti-aliasing, adaptive supersampling, and micro-specular-highlight rendering. These methods suit each different scenario, but all of them aim at solving the reflection of complex surfaces at each level.

Glittery surface rendering is our work in progress study. Glittery surface is difficult to accurately render due to its complicated structure within each rendering pixel. Our glittery surface rendering stores multiple orientations of a surface in one variable and use it to create sparkles and glints on reflection surfaces. The technique is still not evaluated with any reference result. But the result is interesting and should be useful for the reader.

Spatio-temporal reflection anti-aliasing improves the quality of reflection of complex surfaces in motion by considering and integrating reflections in both temporal and spatial space of the surface. We focus on the reflection of water surfaces, which are complex surfaces. We compute a contribution of reflection in an interval time between two rendering frames instead of only the current time to help improve the quality of the result. We also approximate an NDF of the surface as a Gaussian function to deal with reflection in spatial space.

Adaptive supersampling solves the problem of our spatio-temporal reflection anti-aliasing. Our previous method uses the smooth function approximation to calculate the reflection in spatial space. However, as previously stated, this approximation causes a loss in reflection detail. Adaptive supersampling follows a similar approach as our previous method; then, it effectively adds all the important reflection detail afterward. The detail rendering process is naturally expensive. Thus our method evaluates the importance of each pixel and computes an accurate reflection adaptively in an efficient parallel process.

Micro-specular-highlight rendering polishes the reflection of simpler surfaces, which are ignored in our previous methods. Our previous methods only deal with reflections in temporal space and highly dense reflections in spatial space. Our third method focuses on less dense reflection in spatial space and improves its quality without losing the sharpness like the other conventional methods. Our approach chooses a normal direction with the brightest reflection within a pixel, unlike the

traditional method, which uses a normal direction at the center of the pixel. With the brightest reflection, our method appropriately weights the result according to the light source visibility. The result shows that our approach improves the quality and stability of reflections on less dense, complex surfaces.

Our contributions are:

- Introduce glittery surface real-time rendering by using multi-normal-per-pixel mipmapping.
- Introduce real-time anti-aliasing method for ocean wave reflection by considering both temporal and spatial reflection space.
- Introduce real-time adaptive supersampling method for ocean wave reflection.
- Introduce real-time micro reflection rendering method by choosing the best normal vector of the surface of each pixel.

1.4 Organization

Here we explain how our thesis organized:

- In the first chapter, we already introduced our research, basic terminology, and contributions.
- In the second chapter, we will introduce more practical background knowledge.
- In the third chapter, we will introduce a history of photorealistic rendering research and focus on research relevant to our study.
- In the fourth and fifth chapter, we will explain our proposed methods in detail. We will talk about the challenge, contributions, methodology, result, and evaluation.
- In the sixth chapter, we will conclude our whole study with the benefit of our study and what is next for the study.
- Also, in the appendix, we will show all the experiment data and mathematics solutions that are too long to be included in its chapter.

Chapter 2

Background Knowledge

This chapter explains essential knowledge for three of our research topics. We already describe some basic terminology in the previous chapter, but now we will get into more practical and relevant issues.

2.1 Rasterization Based Rendering

Our methods use the rasterization based rendering method because we focus on real-time rendering. The rasterization based rendering method specializes in real-time rendering because of its high-efficiency algorithm and specialized hardware, a graphics processing unit.

Rasterization is a method of converting shape data to a raster image. A raster image is the shape's information but in the form of a series of pixels suitable for displaying on a monitor or storing in an image format. We use rasterization in 3D rendering to convert 3D shape data, usually a triangle, to a pixel data series in our render target. Rasterization also interpolates the data between vertex points (corner points) of shape data at each pixel. However, rasterization is just a part of the graphics pipeline.

2.1.1 Graphics Rendering Pipeline

Graphics rendering pipeline is a conceptual model or a rendering architecture. It generalizes the architecture, so the implementation and interface between hardware and software are more straightforward. A graphics processing unit (GPU) or a

graphics card is designed to execute this workflow efficiently.

The pipeline consists of multiple computational stages, but we can simplify them into three parts: application, geometry, and rasterization.

The application part is your main program, where you process your data, user input, or user interface. This part also responsible for preparing the data and setting for rendering on a GPU and sending an execution signal to the GPU. The data is mainly geometry data, camera data, and lighting data.

The geometry part is executed on a GPU. It transforms geometry and other necessary data to 2D screen space coordinate before passing them to rasterization. The transformation code is often called a vertex shader. Vertex is a point data of geometry. We only transform the vertex of geometry and the vertex data at this stage, so the computation is not expensive. The execution code of this stage is called vertex shader.

The rasterization part is also on a GPU. It converts the data from the previous stage to the series of pixels of the render target. The conversion is finding which pixels are inside the geometry shape, which usually is a triangle. The conversion also finds the other data of each pixel by interpolating data at each vertex. After that, the system computes the color of each pixel by executing the assigned shader code. The color computation program is often called a pixel shader.

There is one major problem with this system that is related to our research. The rasterization is simple when a whole pixel is inside or outside of a triangle. But what if only a part of it is inside. This situation causes an aliasing problem. Figure 2.1 shows an example of graphics rendering pipeline.

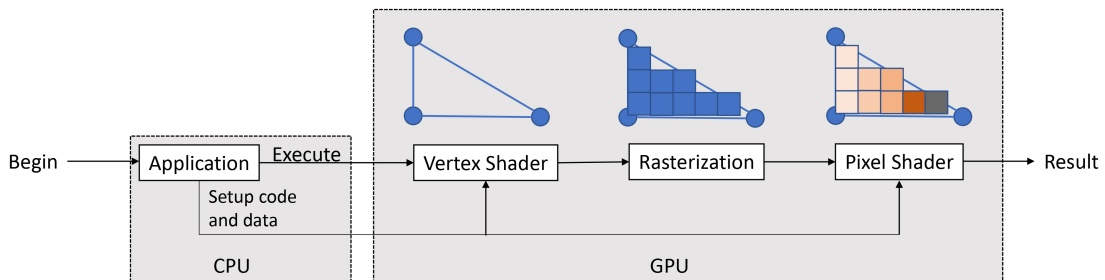


Figure 2.1: A simplified flowchart of graphics rendering pipeline.

2.1.2 Anti-aliasing

Aliasing is a problem when we get a wrong result by under-sampling a signal. In this case, we only sample once per pixel to see if it is inside, outside, or on the edge of a shape. There is a rule that governs this decision, which is called the rasterization rule. But not the less we cannot get a correct answer from just one sample. This error creates an artifact at the edge of the triangle. Our method in Chapter 5 also deals with aliasing, but it is a different kind of aliasing. This aliasing here is about hidden-surface-determination while our aliasing is from the lighting. There are many methods to solve this problem. We will explain these methods in Chapter 5.

2.2 General-purpose computing on GPU

General-purpose computing on graphics processing units or GPGPU uses a GPU to compute a general problem instead of computing a graphics pipeline. A GPU is a specialized circuit for computing and generating an image sent to a display device. The circuit's structure is designed for computing an intensive task by using a massive number of compacted processors to work on the same task but with different data simultaneously. Fortunately, the image generating task can be divided into multiple small independent tasks. Thus, the circuit can process them in parallel, which results in a tremendous efficiency improvement over a traditional processing unit. It was initially only used for accelerating the scan-line rendering process. However, we can also use it to process other general tasks, such as ray tracing. With a GPU, we can process a simple ray tracing program and obtain a result in an interactive time. Our research heavily relies on GPGPU to precompute a data for our algorithm and execute our process that is not aligned with the traditional rendering pipeline.

Path tracing and ray tracing generate a photorealistic image by tracing numerous paths from eyes to light sources. Since these paths are generally independent of each other, they can be computed separately and simultaneously on each thread on a GPU, containing many threads. A standard path tracing algorithm is an exceptionally separable algorithm, and therefore a GPU can immensely improve its performance. However, there are still some problems regarding the usage of a GPU in path tracing.

2.3 Illumination Model

The illumination model or shading model is a computational model of a surface's appearance under a light source's influence. Researchers invented many illumination models for different applications, but we can categorize them into two groups: photorealistic and non-photorealistic models. The photorealistic model imitates real-life light physics, while the non-photorealistic model can be anything.

The photorealistic model's goal is to reproduce the same lighting result as in real-life. Light energy from a light source can be reflected, absorbed, or transmitted when it hit any surface. A function defines the behavior, or this interaction is called the bidirectional surface scattering reflection distribution function (BSSRDF).

BSSRDF describes how much light energy is exited from a surface at one point when light enters the surface at some point. BSSRDF includes a subsurface scattering effect, which is a phenomenon of light penetrate a surface from one point and exit the same surface at a different point. This behavior makes the function complicated; thus, we need the bidirectional reflection distribution function (BRDF).

BRDF assumes that incoming and outgoing light is at the same point on the surface. The assumption makes the model simpler to be computed. It ignores subsurface scattering and transmission, which is also important; however, BRDF is very simple and efficient; thus, most reflection model use BRDF instead of BSSRDF:

Diffuse reflection is a reflection that uniformly distributed to any direction. Its real-life phenomenon counterpart is when light enters and surfaces at one point. The light is reflected around inside the surface and gets absorbed. Then the light exit out of the surface at the same point. The reflection inside the surface makes the light possible to travel in any direction when exiting the surface.

Specular reflection is a reflection that non-uniformly distributed to any direction. Unlike diffuse reflection, specular reflection has a particular direction, usually around a reflection direction, with high reflection distribution. A smooth mirror is a perfect example of an object with a perfect specular reflection. Its real-life counterpart is a light that hits a surface, but it doesn't enter the surface. Instead, the surface reflects the light to a direction that depends on an incoming direction and the surface orientation.

Chapter 3

Related Work

3.1 History of Computer Graphics Rendering Research

We can say that an invention of a display device is the starting point of computer graphics. The cathode-ray tube (CRT) display is invented by Braun [43] in 1897. Then the monitor becomes essential equipment for displaying intuitive information from the military tool. With the knowledge from implementing military equipment, the first video game, Tennis for Two, was created by Higinbotham and shown in 1958. It uses an oscilloscope to display an interface. Then right after that, in 1959, TX-2 computer [14] is developed for studying the interactive interface between man and machine. It has a penlight that you can sketch and store the sketch data for later use; it can also create a simple geometry from input data. Other important inventions are the Bezier curve and HMD. Bezier curve is created by Bezier [68] to design and model an automobile in digital. Sutherland experimented with a head-mounted display (HMD) [70], which enables a stereoscopic 3D viewing experience.

Around the same time, in 1968, a critical invention for photorealistic rendering is born; the ray casting algorithm is invented by Appel [3]. We can say that this is the beginning of photorealistic rendering research.

Later in the early 1970s, there is an advancement in metal-oxide-semiconductor large-scale integration (MOS LSI) technology, enabling a memory chip to store a raster image data. A raster-based rendering system will be the fundamental system

for real-time rendering research.

We will explain the progress of photorealistic rendering and real-time rendering separately in the next two subsections.

3.1.1 Photorealistic Rendering Research

Ray casting method showed that we could use it to render a simple geometry with shadow shading, but it still can't be called a photorealistic result. Whitted [79] was the first to develop a ray tracing method that uses ray casting to calculate global illumination lighting. Its product is photorealistic but with a high computational cost. Cook and Torrance [17, 15] introduced a microfacet reflection model, which is a physically based model.

Goral *et al.* [23] took another approach to compute the diffuse global illumination effect by considering all the surfaces as finite elements. Then the method adds the light energy radiation on these finite elements. The technique is called radiosity. Nishita *et al.* [78], and Wallace *et al.* [77] also advanced the method further.

Cook *et al.* [16] improved ray tracing by distributing ray to create a motion blur, depth of field, and glossy reflection effect. Then Kajiya and James [35] introduced one of the most important equations, the rendering equation, which became a basis for all photorealistic rendering. They also invented path tracing, which solves the rendering equation with ray tracing by using Monte Carlo integration. Path tracing enabled a wide range of light physical phenomena to be realized in computer graphics. However, the computational cost is still too high to create a noiseless result until Veach [74] and Lafortune [40] described bidirectional path tracing. Veach also introduced multiple importance sampling and metropolis light transport. These techniques significantly improve the photorealistic rendering's quality and performance, but it isn't interactive yet. Wald *et al.* [76] shows that it is possible by using acceleration structure for ray-triangle intersection testing.

3.1.2 Real-time Rendering Research

Since the MOS LSI chip's improvement, the rasterization-based method is the primary tool in real-time rendering because of its efficiency. But before heading into

that, Catmull [13] invented texture mapping, which is crucial for both offline and real-time rendering.

One of the essential topics for rendering was hidden surface determination. We have to efficiently find which surface is in front of the other surface to render them correctly. There are many methods proposed, but one that became a standard is Z-buffering by Strasser [69]. The process stores each pixel's minimum depth, which will be compared and updated when rendering to ensure that only the closest surfaces are present. Next, there is an improvement in how we shade a surface. The surface can show a gradient of reflection, instead of a flat look with Gouraud shading [24] and Blinn-Phong shading [9]. Bump mapping [10] and normal mapping [50] also increase the detail of the surface by perturbing the direction of each point on the surface.

In 1980, NEC 7220, which is one of the first GPU chips, was released. The chip provides high-performance rendering features, which started to revolution real-time rendering technology. Around the early 1990s, the interactive rendering API OpenGL and DirectX is developed. Still, the cause of the adoption of these two is the first home video card; a GPU called GeForce 256, which Nvidia released in 1999.

With the advancement of real-time rendering API, which creates a programmable shader, general-purpose computing on GPU (GPGPU) is found. It is a way to utilize a GPU to solve a general computational problem other than the rasterization rendering. GPGPU will later help real-time rendering by making ray tracing or path tracing become possible in real-time.

There is much research for each lighting phenomena effect in rasterization based real-time rendering. Shadow calculation can be done by planar [8], curve [48, 52, 58], volume [27], or mapping [80]. Shadow mapping became a fundamental technique of today's real-time shadow rendering. Global illumination is also possible in real-time with an exchange of a loss in quality or an animation. An example of the real-time global illumination method is light mapping [1, 51], screen space ambient occlusion (SSAO) [5], screen space subsurface scattering (SSSS) [33, 32], screen space reflection (SSR) [73], and voxel cone tracing [18].

In 2013, Karis [38] introduces a method to approximate the physically based shading in real-time. The technique enables a consistent, realistic, and intuitive

parameter material that can be rendered in real-time.

In 2018, Nvidia released GeForce RTX, a GPU series that included a ray tracing specialized circuit. The GPU reinvigorates real-time ray tracing research, which hopes to replace the current rasterization based rendering.

3.1.3 Efficient Path Tracing

In recent date, path tracing's performance has been significantly improving due to parallel implementation and other sampling techniques. It is getting close to real-time rendering, thus we should also investigate the research in this are as well.

Parallel Path Tracing

The Russian roulette technique is usually used in path tracing [36][40] to probabilistically terminate the path tracing process. In path tracing, paths between eyes and light sources are searched by casting rays from either eyes or light sources and checking their intersections to surfaces. If the casted rays hit any surfaces, reflected rays are cast and tested for further intersections. This whole process continues until they connect the eyes to any light sources and the eyes-light paths are used to calculate energies from light sources to the eyes, or pixels in the result. The problem is an unpredictability of the path tracing process since the number of reflections cannot be predetermined before a ray hits a light source. Some paths might start on a surface that is obscured from any light sources thus these paths would require more iterations than paths that start in a surface in front directly of a light source. The Russian roulette technique was introduced to terminate paths that are too difficult to be calculated without affecting the correctness of the process by randomizing path termination trigger, which becomes easier to be triggered as the path bounded from any surface. However, if we use this algorithm on multiple threads, it causes an unbalanced workload due to the unpredictable termination of the algorithm on each thread. The terminated threads become idle threads and do nothing, which reduces the performance of the whole process. Novák *et al.* [55] proposed a path regeneration method to solve this problem. This method regenerates paths from their originating pixel for each terminated path. In another word, any threads that are terminated by the Russian roulette before the whole process ends instead of be-

coming idle threads, the algorithm assigns new jobs to them. These new jobs are to trace the same pixels that they were tracing before. Because of this, the terminated threads can continue to work and be able to produce more than one sample for each thread in each iteration. Unfortunately, due to the architecture of a GPU, if there are any threads that are processing on a different procedure, such as the regeneration procedure, the other threads must wait for the branched threads to finish their procedure. This waiting becomes an obstacle to archive a full performance parallel path tracing. Since then, Wald [75] presented an active thread compaction method to speed up the path tracing process on a GPU. The method creates a list of paths that is tracing or to-be-traced. Then, each thread of a GPU traces just one bound of a path in the list or terminates it. After that, stream compaction [7], which is an algorithm that can remove elements from a list in parallel, is used to remove terminated paths from the path list and let the threads efficiently process these paths. Antwerpen [2] proposed an even more efficient method by combining active thread compaction and path regeneration techniques, which improves the SIMD efficiency of path tracing.

Adaptive Sampling and Reconstruction

Monte Carlo path tracing suffers from noise due to variance. One of the solutions to reduce the noise is denoising by filtering, which is an active research field [88]. Rousselle *et al.* presented a denoising method that applies multiple filters to a noisy result then chooses one, which minimizes the mean square error, of the filters for each pixel [62] (Figure 3.1a). They made further improvements by using non-local mean filtering [61], which is a filter that processes on a patch of an image, not a pixel (Figure 3.1b), and using feature information, such as normal direction, texture, and depth, to filter the input to produce multiple candidates [63]. The method then uses SURE error estimation to compute the weight of each candidate and combine them together to create the result (Figure 3.1c). Gastal *et al.* [21] presented a real-time high dimensional filtering technique by computing multiple manifolds, which is able to adapt to the high and low frequency of any input, of the result and denoising the input with the convolution of these manifolds (Figure 3.1d). Mehta *et al.* [49] also proposed an image-space filtering method for interreflections interactively by

analyzing signals in the Fourier domain and using an axis-aligned filter for denoising. While our method builds on these methods, we focus on the implementation of an efficient adaptive sampling technique on a GPU.

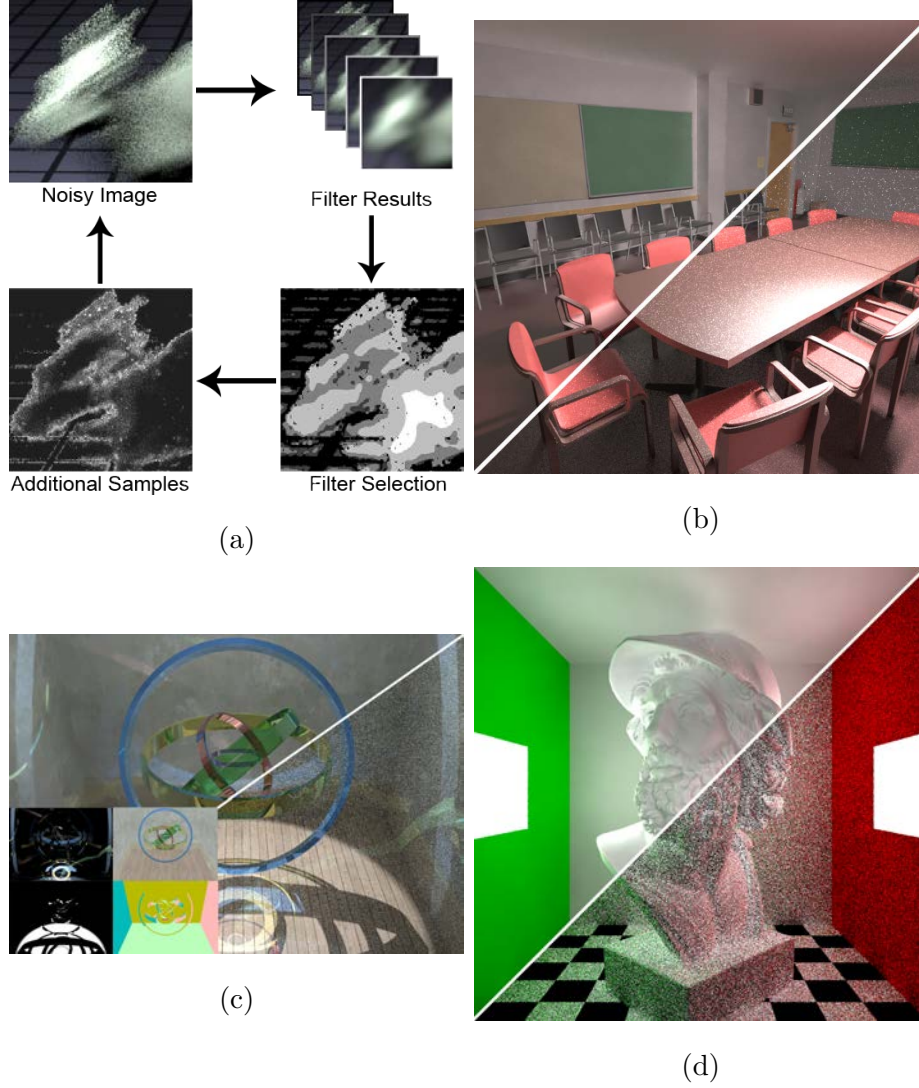


Figure 3.1: Progression of adaptive sampling and reconstruction research.

3.2 Relevant Research on Complex Surface Reflection

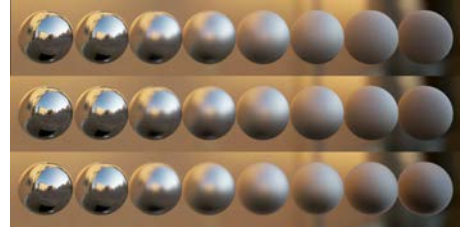
There are three main works related to our research: reflection rendering, glint and caustic rendering, screen space anti-aliasing, and geometric analytic anti-aliasing.

3.2.1 Reflection Rendering

One of the most popular methods for rendering reflections in real-time is image-based rendering. A method proposed in [12] renders realistic reflection images at an interactive speed by precomputing radiance maps that store the convolution of an environment map and a BRDF to efficiently compute the outgoing intensity of light for an arbitrary surface orientation and viewing direction (Figure 3.2a). Since this method cannot change either surface materials or lighting conditions at run time, Karis [38] introduces a split-sum approach that approximately computes the product of the environment map and the BRDF efficiently (Figure 3.2b). More recent work achieves real-time shading due to a polygonal light source [28] (Figure 3.2c) or a spherical light source [19]. However, none of these methods pay any attention to temporal aliasing.



(a)



(b)



(c)

Figure 3.2: A part of reflection rendering research progression.

3.2.2 Glint and Caustic Rendering

Glint and caustic is one kind of reflections but it is closely related to our work thus we separated it from the typical reflection rendering. We categorized it into two groups: procedural generation and multi-sampling.



Figure 3.3: A part of glint and caustic rendering research progression.

Procedural generation creates a surface model, which can simplify the computation. Jakob *et al.* [31] proposed a stochastic model of randomized metallic flakes, which was used to evaluate a surface in each pixel to render glints (Figure 3.3a). However, the evaluation is still too expensive to be done in real-time. Zirr and Kaplanyan [87] proposed a procedural model of micro-detail that produced a glittery effect. The procedural model is consistent due to stable multi-scale noise generation, and runs in real-time. However, both methods support only limited types of surface models.

Multi-sampling uses multiple samples to obtain the accurate result efficiently. Some methods for temporal anti-aliasing [85] reuse the previous frame's samples to increase the efficiency. It can sample difficult reflection to reproduce glints and caustics but the information of a sample in the current frame might not be in the

predicted location or not available at all in the previous frame. This causes the method to fall back to fewer samples, increasing the aliasing. Yan *et al.* [84, 83] introduce a technique that accurately renders glints by sampling a whole normal map within a pixel footprint with a computationally efficient strategy (Figure 3.3b). Zeltner *et al.* [86] sample light paths related to caustics and glints efficiently by using manifold walking, which helps light paths to converge to important light paths (Figure 3.3c). Loubet *et al.* [45] importance-sample caustic and glint light paths by approximating normal variation as a linear interpolation between three normal vectors at vertices of a triangle (Figure 3.3d). This work uses an assumption similar to our work, which will be explained later. All the methods in this group except for the temporal anti-aliasing are computationally too expensive to be computed in real-time.

3.2.3 Screen Space Anti-aliasing

Fast approximate anti-aliasing (FXAA) [44] is one of the most popular screen space anti-aliasing methods. It simply detects jagged edges in an image and smoothes them (Figure 3.4a). It is fast but it blurs some details in the final image. Screen space temporal anti-aliasing has also been studied for a long time. One of the first methods is by Korein and Badler [39]. It determines the area of pixels that a moving object covers and then performs filtering on that area. This method works for a simple moving object. A method by Shinya [66] uses a velocity field to track surfaces moving on a screen. Then the method reconstructs an anti-aliased result by blending pixels with the same surface position in multiple-frames. This method can be called a temporal reprojection method. The temporal reprojection method has a wide variety of applications [85, 53, 64], but there are three main problems. First, it cannot deal with any surface that is hidden in previous frames. Second, it has parameters that needed to be adjusted carefully. Third, it cannot track the movement of the reflection direction, which is our main objective. Another useful method is screen space NDF filtering [37, 72]. The method approximates surfaces' NDF as a Gaussian distribution function by using neighbor pixels' normal vectors and uses the NDF to compute the light contribution (Figure 3.4b). However, the approximation's accuracy is limited. The results are blurry reflections from using

the over-filtered smooth function NDF.

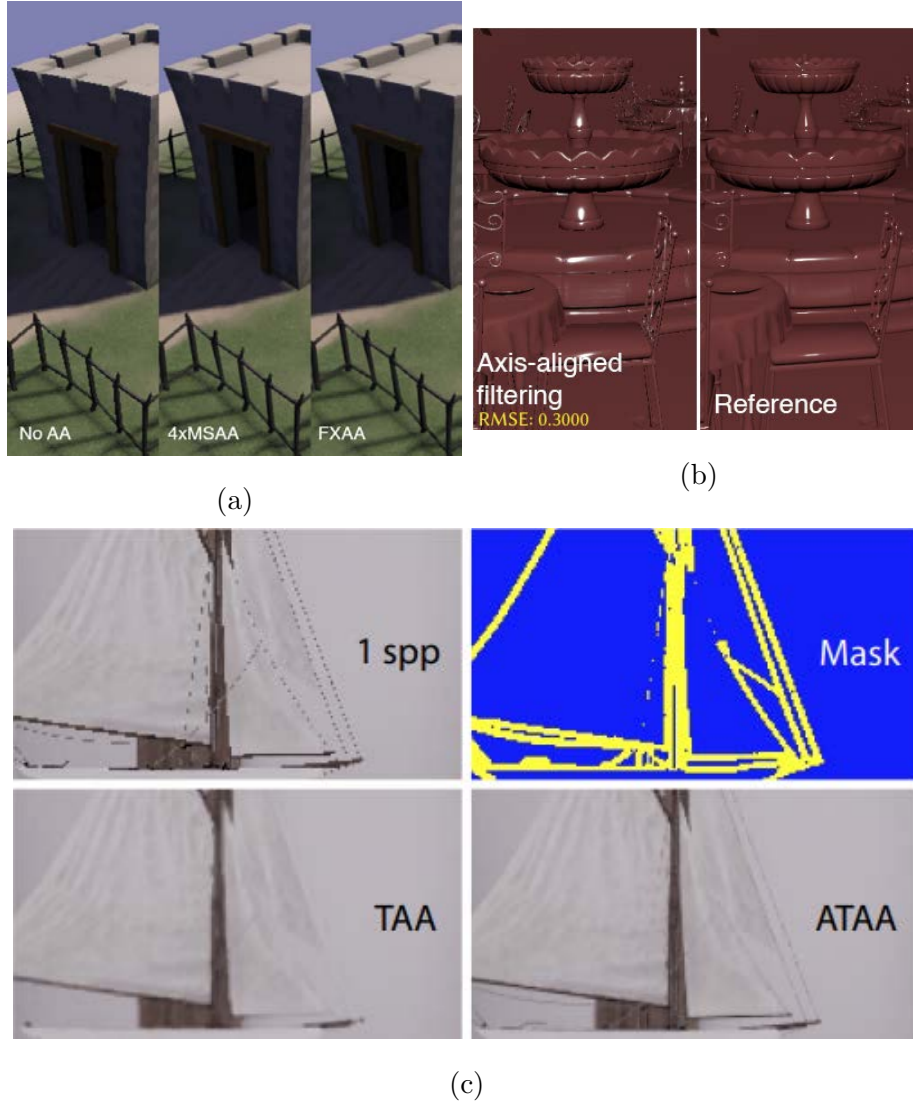


Figure 3.4: A part of screen space anti-aliasing research progression.

We can also use adaptive method to increase an efficiency. Adaptive anti-aliasing or supersampling is mainly about multi-sampling each pixel according to some evaluation. Lau and Rynson [41] introduce adaptive supersampling to remove hidden surfaces at the edge of geometry. Jin *et al.* [34] evaluate pixels in both image and object space then multi-sample them with ray tracing accordingly. Barringer *et al.* [4] render a scene first on a GPU and then evaluate and sample each pixel if needed with a CPU. Hollander *et al.* [29] apply adaptive supersampling in deferred rendering by using a supersampling g-buffer to help evaluate and shade each pixel appropriately. Lee *et al.* [42] introduce tile-based adaptive supersampling and undersampling ray tracing. Marrs *et al.* [46] remove artifacts from temporal

anti-aliasing [66, 71] by evaluating artifact pixels and multi-sample them with ray tracing (Figure 3.4c). These methods usually need a small number of additional samples (e.g. less than 16) per pixel. However, for rendering water surfaces, especially surfaces in a distance, the pixels can contain highly detailed geometries and more samples are usually required.

3.2.4 Analytic Anti-aliasing

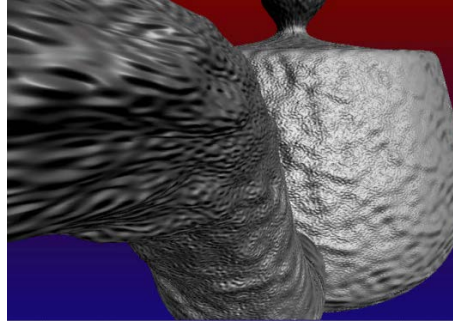
Our method is categorized in this group. The methods in this group analyze the cause of the aliasing and find a mathematical solution for each target aliasing situation. A clamping method [54] removes aliasing on a textured surface by using the Nyquist theorem [65], which we will use in our method. The method proposed in [25] analyzes the movement of a polygon in image space and generates a space-time representation of the object for spatial and temporal anti-aliasing. EWA volume splatting [89] is an anti-aliasing method for volume rendering. It uses the elliptical Gaussian function as a reconstruction kernel for the volume to avoid aliasing. We are inspired by this work and borrow the idea of its Gaussian kernel’s affine mapping. Our method also builds on LEAN mapping [56] that approximates the distribution of surface normal vectors with a Gaussian function to eliminate aliasing that comes from using a bump map (Figure 3.5a). LEADR mapping [20] is another method that extends LEAN mapping to cover physically based rendering. Becker and Max [6] combine three rendering algorithms, which are BRDF, bump mapping, and displacement mapping, and transition between them in an aliasing-optimal way (Figure 3.5b). Bruneton *et al.* [11] also use these rendering algorithms as three levels of details to eliminate aliasing on a dynamic water surface in real-time (Figure 3.5c). Han *et al.* [26] uses spherical harmonics for low-frequency data and a spherical von MisesFisher distributions for high-frequency data (Figure 3.5d). Wu *et al.* [81] accurately render anti-aliased materials across multiple levels of details by generating an SVBRDF from a displacement map and a BRDF. The method considers occlusion, shadow-masking, and interreflection effects of the displacement map as well. However, these methods do not take into account the temporal aliasing, which we address in this paper. Feline mapping [47] also helps us improve the efficiency and quality of our work. It is a texture filtering method that approximates

an anisotropic filter by the sum of multiple isotropic filters.

Less related to our work is Monte Carlo distributed ray tracing [16], which combines spatial and temporal aliasing by tracing many rays from random points in the finite pixel area, and at random times in the frame interval. It produces noisy images unless very many rays are used, but modern noise removal algorithms can decrease this noise fairly effectively.



(a)



(b)



(c)



(d)

Figure 3.5: A part of analytic anti-aliasing research progression.

Chapter 4

Real-time Glittery Surface Rendering

In addition to our main methods, we also experiment with glittery surface rendering method. This part of the research is still in the progress but our result shows a potential and should be useful for the reader.

Glittery surfaces have attractive visual appearances; we see many small bright spots on the surfaces but the bright spots suddenly disappear when the view and/or light directions change. This phenomenon happens because the surfaces have a complex microfacet structure. Many tiny facets are aligned in different directions at each spot on the surface, and each of these facets can reflect a specular light individually. If the microfacets reflectance is high enough, each of the facets could reflect light and create a strong glint by itself. There are many glittery surfaces in the real-world such as snow, metallic paint, and glitter for cosmetics. If we could reproduce this appearance with computer graphics, we can raise the quality and the fidelity of synthetic images and fill the gap between computer graphics and reality. However, it is difficult to reproduce the glitter effect with the traditional rendering method.

Recently, several methods have been proposed for rendering the glitter effect [87, 31, 84, 83, 86]. Although these methods succeeded in rendering the glitter effect realistically, one problem with these methods is the expensive computational cost. None of these are efficient enough to be executed in real-time. Our work aims to address this problem by sacrificing the least amount of quality while capturing the

effect of glittering in real-time. Our method can reproduce a sparkling effect from glittery surfaces in interactive applications.

4.1 Proposed Method

To render the glitter effect, we need to efficiently represent the structure of glittery surfaces that consist of many microfacets. A straightforward solution is to use a very high-resolution normal map to store each microfacet's normal vectors individually. However, this requires expensive storage costs. Instead, we propose a new data structure called a *flake map* that can store multiple normal vectors for each texel.

4.1.1 Flake Maps

Let us consider a binary bit string with length N that is stored in each texel of the flake map (typically, $N = 32$). We then subdivide a hemisphere uniformly into N bins as shown in Fig. 4.1 and each bit in the bit string corresponds to one of the bins of the subdivided hemisphere. For each bin, a vector pointing toward the center of the bin is assigned. When the bit is one, we assume a microfacet with the normal vector of the corresponding bin. We can therefore store N normal vectors at the maximum for each texel.

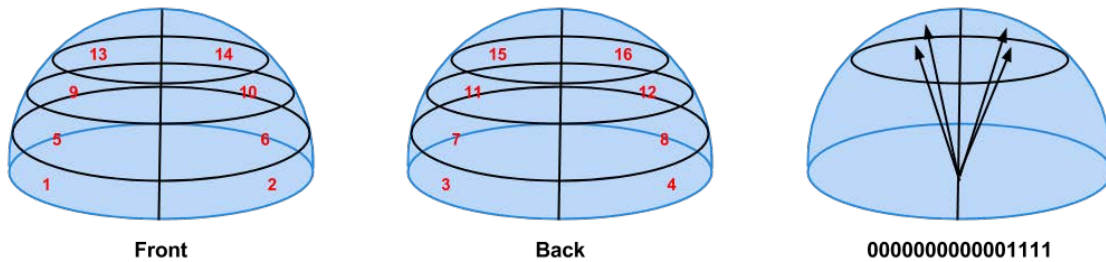


Figure 4.1: Left and center figures show front and back of the hemisphere, which is divided into evenly 16 spaces. Right figure is an example of 16 bit field that represent 4 vectors, which are pointed to 13, 14, 15 and 16 labeled spaces, we store these vector bit field as 0000000000001111.

4.1.2 Construction of Flake Maps

A flake map is constructed from a normal map that stores a single normal vector for each texel. Similar to the mipmap data structure, we generate multiple flake maps at different levels of detail. The finest level of the flake map is generated by quantizing the original normal map; that is, the normal vector in the normal map is quantized to the closest vector among N vectors of the bins described in the previous section. The corresponding bit is set to one. Higher levels of the flake maps are constructed by combining four pixels of lower levels by simply applying bitwise OR operation. Figure 4.2 shows the flake maps generation process. With this method, we can generate a flakes map with the mipmap structure that could describe the surfaces' structure at different levels of detail. Then we map this flakes map onto our polygonal model in the same way as texturing the model.

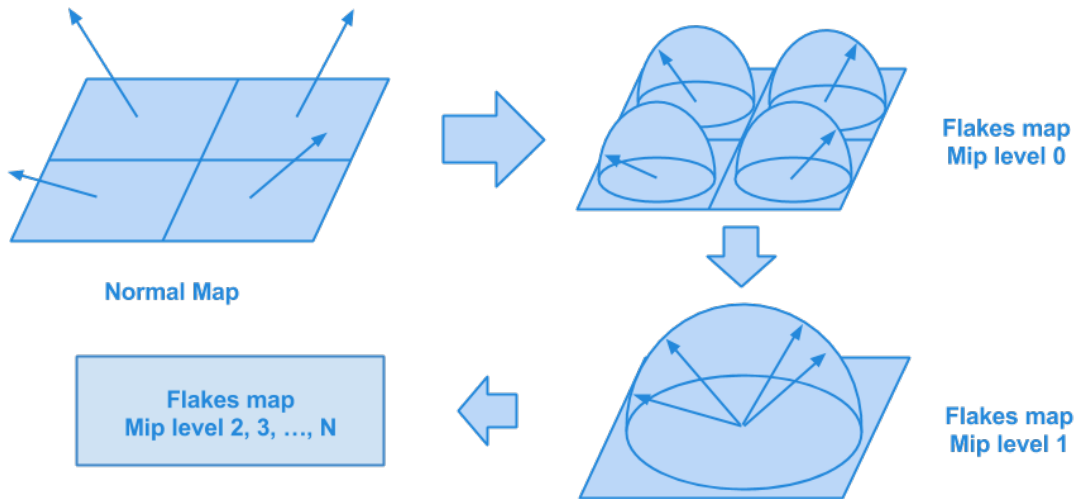


Figure 4.2: Flakes map generated by using a normal map as a reference. Flakes map mip level 0 is generated directly from normal map data then generate higher mip level data from the current level.

4.1.3 Rendering

For the rendering part, our current implementation supports directional and environment map light sources. Due to the efficiency of our flakes map, we can naively do the rendering as follows. We calculate lighting from a directional light or an

environment map light with every normal vector in the pixel and then average them for each pixel. However, since we uniformly divided hemisphere space into a small number of spaces, each labeled space from the flake map covers a huge area. If we use just one normal vector to represent each divided space, it will make the lighting look unnatural because many flakes will share the same normal vector, and lighting will focus on that point instead. We solved this problem by randomly jittering a normal vector, which we got from the flakes map in each pixel. We use a UV value on the surface as a seed number to randomize an offset vector to the labeled space-centered normal vectors, making the normal vector jittered inside its own labeled hemisphere space. This jittering makes the lighting spread around the designated space instead of focusing on the center of each labeled space, as shown in Fig. 4.3.

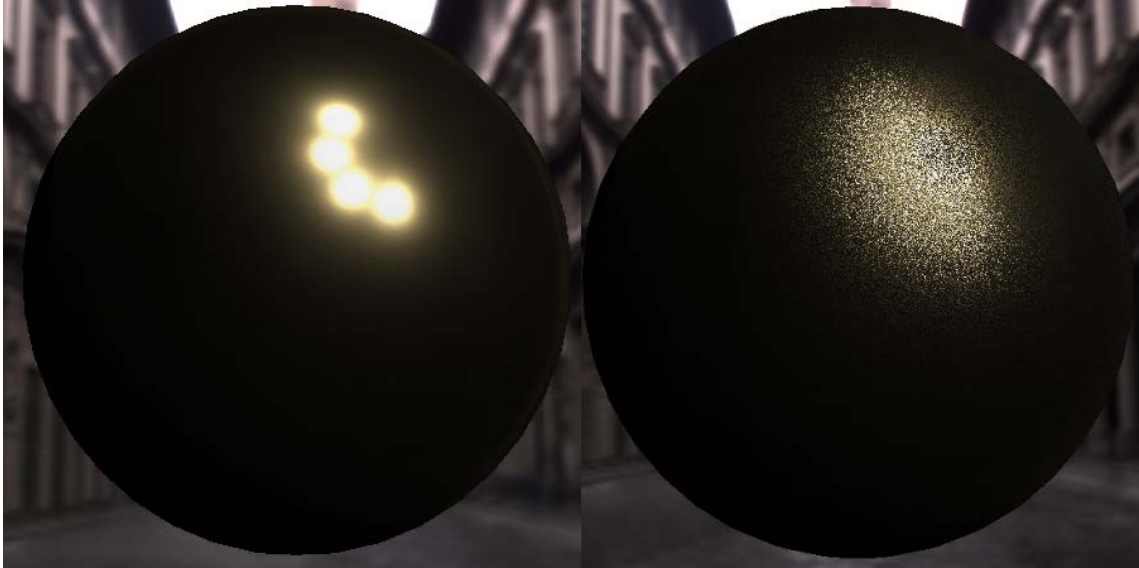


Figure 4.3: These two scenes have the same directional light and the same four flakes in each pixel. The left image is without any jittering; the left ball focuses all of its four normals onto the center of 4 labeled spaces, which result in 4 strong specular lights. The right image is rendered with a random offset; it distributed four normal vectors in each labeled space, create the glittery and uniformly lighting.

4.2 Results

We use $N = 32$ bits for the flake map. Then we generate flakes map by three methods:

- Choose four normal vectors on the top of the hemisphere, which give the flakes map a natural uniformed glittery look.
- Choose four normal vectors on the top of the hemisphere and randomly add another one of the normal vectors to inject some sparkling.
- Choose a normal vector for each pixel from a reference normal map.

With these three types of flakes map, we can acquire many distinctive results and make it close to our ideal result. Then we render these flakes map on a high-heel model, which we used its flakes map UV values as our seed value for randomizing jittering offsets.

We experiment with our method by generating flake maps from normal maps and randomly generate normal vectors. The result is rendered under a directional light and an environment light with above 100 FPS under our standard spec PC (Intel Core i5 CPU 760 @ 2.80Ghz, NVIDIA GeForce GTX 460). Figure 4.4 and 4.5 show the results from different flakes maps.



Figure 4.4: The result image of our glittery rendering method. The high heel is rendered in real-time with randomly generated flakes map. The result show glittery lighting on its body as it is being shined by a directional light and a light from environment map.

4.3 Discussion

Our method can generate a look of glittery surface in real-time. Our approach could also change the appearance of the glitter according to the input normal map. However, we don't have a reference result to compare to yet; thus, we cannot measure our method's effectiveness.

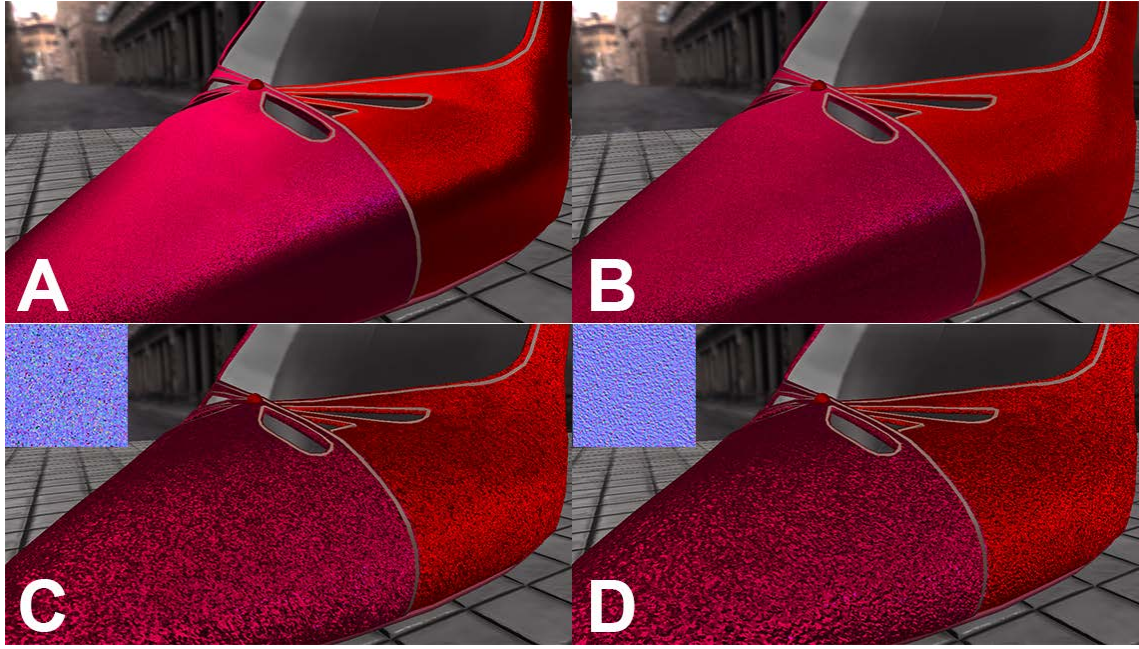


Figure 4.5: Rendered images from different flakes maps: A use 4 vectors on the top of hemisphere flakes map, B use A flakes map with one more random vector, C is generated from a metallic flakes normal map, D is generated from an isotropic noise normal map.

Another problem is that our vector bit field cannot store a sharp structure due to its limited number of labeled vectors, 32 in our case; however, it is enough to capture the sparkling effect from normal objects as we wanted.

This work is still in progress. Next, we want to create a reference image by rendering a high-resolution normal map to evaluate our result and find a more efficient method for rendering the effect rather than our naive approach, which calculates lighting on every flake.

Chapter 5

Real-time Spatio-temporal Reflection Anti-aliasing

Ocean simulation and rendering have been studied in the computer graphics field for 40 years. That research can simulate an ocean under various weather conditions [22], and interacting with solid objects in three dimensions in real-time [30]. However, many problems in real-time ocean rendering remain. One of the issues is aliasing and flickering in rendering reflections on the water surface.

The ocean surface is highly dynamic. It moves rapidly, and thus, its shading changes rapidly as well. Usually, this does not pose any problems if the shading is smooth. However, for a surface that has a strong highlight or bright reflection moving rapidly, it causes inaccurate and unnatural flickering. In traditional rendering algorithms, each frame is rendered independently at discrete time steps, resulting in serious temporal aliasing artifacts. Please see Fig. 5.1 for an illustration of the problem. This paper focuses on this problem, assuming the light source is a sphere such as the full moon or the sun.

Removing aliasing in real-time is an active research area, and many methods have been proposed [44, 39, 54, 20, 11]. They can improve the fidelity and efficiency of the rendering method. However, their focus is on spatial anti-aliasing, and most of them do not address the temporal aliasing problem, particularly the one observed in rendering a reflected image of a light source on a wavy water surface.

In this paper, we present a method that can remove the spatial and temporal aliasing simultaneously, assuming that the water wave is represented as a sum of sine

waves and the light source is a sphere. The basic idea is to compute the intersection of the light source with a plane formed by two reflection vectors for neighboring frames. This provides us with a fraction of time when the light source is visible on the water surface. We combine this idea with a traditional spatial anti-aliasing method.

Our main contributions are:

- We propose a method that analytically integrates a light reflection on a summed sine wave surface over a time period.
- We propose a method that approximates a reflection direction distribution and then analytically integrates a light reflection on a summed sine wave surface over the reflection distribution and a time period.
- We give an accuracy improvement for the Feline method of [47].

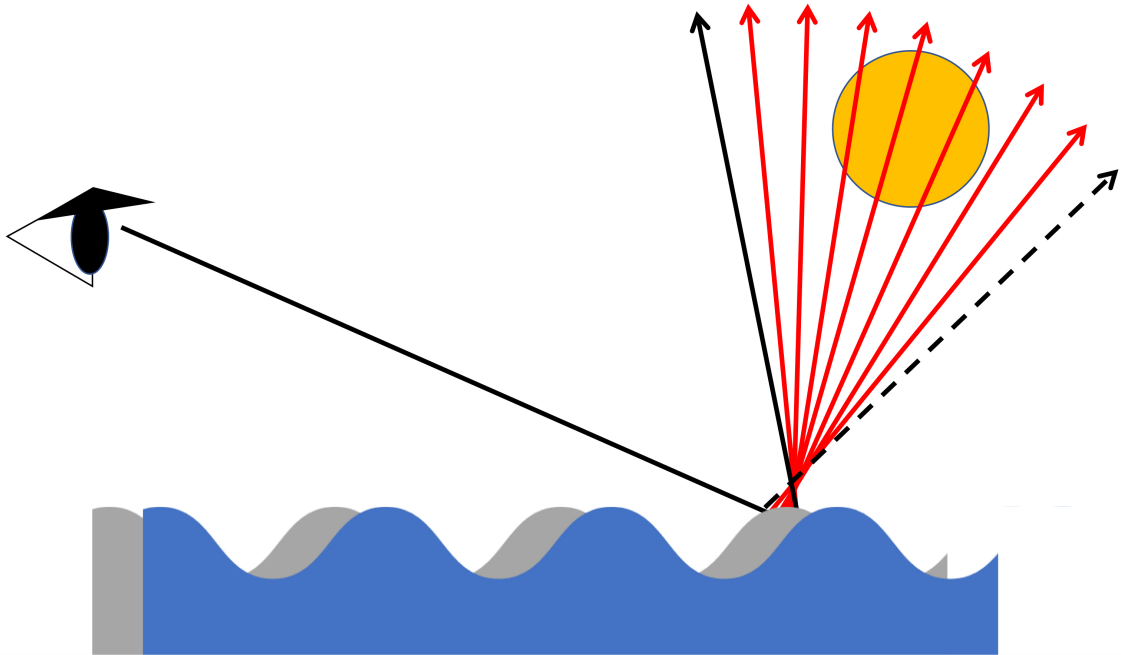


Figure 5.1: The problem in rendering a reflection on dynamic surfaces. Typically a single reflection ray (solid black) is generated for each frame, but it often misses the light source, which causes temporal aliasing. We address this problem by using the ray (dashed black) shot in the previous frame. The light source is sampled by using multiple interpolated rays (red) between them to reduce the temporal aliasing.

5.1 Proposed Method

Our goal is to compute the contribution of a spherical light source over the period of time between frames. As we mentioned earlier, we assume that the water wave is represented as a sum of sine waves with different frequencies and directions. For each pixel, our method first decomposes the water wave into two spatial frequency bands: low and high-frequency bands, namely temporal-aliasing waves and spatio-temporal-aliasing waves, respectively. The high-spatial-frequency component causes spatial aliasing due to under-sampling and may also cause temporal aliasing due to its movement. We develop two methods, temporal-aliasing wave rendering and spatio-temporal-aliasing wave rendering, for these two frequency bands, respectively, and make a smooth transition between them. The fundamental idea of both methods is the same for each sine wave, but in the second method, the spherical light source is blurred to account for the effects of the BRDF caused by the aliasing waves. We are also using a height field of the wave as a displacement map of our water surface geometry. Figure 5.2 shows the outline of our method.

5.1.1 Spatial-aliasing detection

We use the clamping anti-aliasing method [54] to decompose the water wave into the spatio-temporal-aliasing and temporal-aliasing waves. According to the Nyquist theorem in sampling theory, to avoid aliasing, the spatial sampling frequency must be higher than twice the highest frequency of the water wave. In our case, this means that the pixel resolution should be twice finer than the wavelength of the water wave projected onto the screen. That is, when a wave is projected onto the screen, the wave must cover at least two pixels within one wavelength. This maximum frequency of the projected wave is called the Nyquist limit. If a wave does not meet this criterion, then it will cause aliasing. The maximum frequency, or the minimum wavelength, happens when the wave is aligned horizontally or vertically to our screen space, as can be seen in Fig. 5.3. Thus, our Nyquist limit is $\frac{1}{2}$, which is the spatial frequency of a wave with a wavelength equal to the size of two pixels.

For each pixel, we calculate the projected wavelength of each sine wave on the screen. Each of the sine waves is then classified per pixel as either temporal-

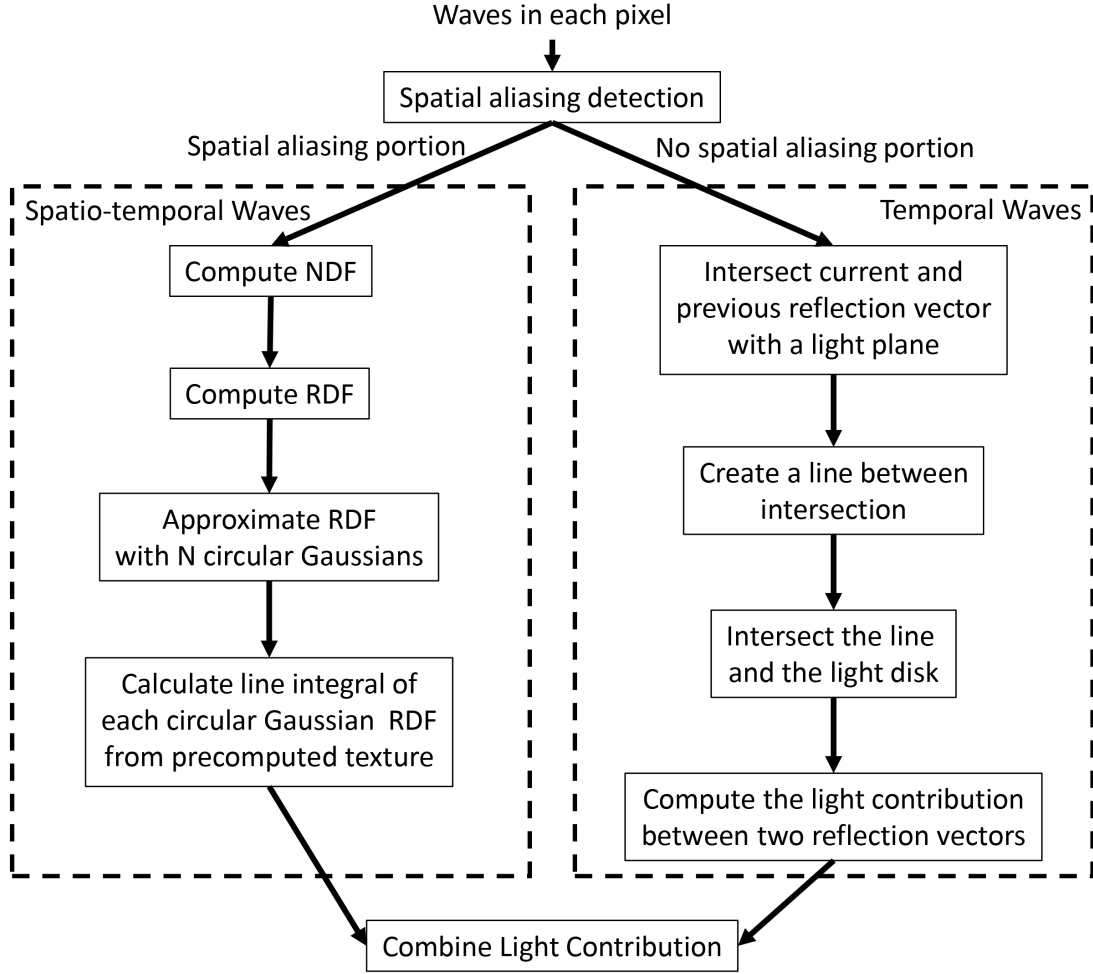


Figure 5.2: The outline of our methodology.

aliasing (called simply “non-aliasing” below) or spatio-temporal-aliasing (called simply “aliasing” below) according to the Nyquist frequency, which detects spatial aliasing. A pixel that only contains non-aliasing waves and a pixel that contains aliasing waves will be then processed by two different rendering methods, which produce slightly different results.

A sudden transition between the two types of waves will create arcs of sudden appearance changes on the boundary between pixels with and without aliasing waves. To prevent this, we use soft classification to blend the results between both methods smoothly. Section 5.1.6 will explain the method in detail.

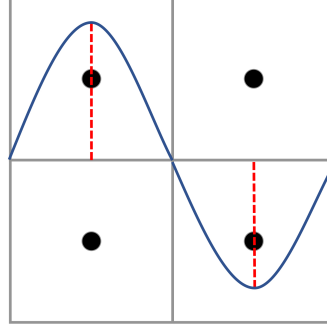


Figure 5.3: The projected wave is aligned to a 2x2 screen. If any wave in any projected direction has higher spatial frequency or lower wavelength than this wave then it will cause aliasing.

5.1.2 Temporal-aliasing wave rendering

For a pixel that only contains non-aliasing waves, we sample a single point on the water surface corresponding to the pixel center and compute the contribution of the reflected light over the time interval between the current and the previous frames. We assume that the spherical light source is distant from the sample point so that it subtends the same solid angle for every sample point and is represented by a disk facing toward the sample point. This disk is the projection of the light sphere onto a light plane perpendicular to the line from the sample point to the center of the light sphere. Two reflection vectors are computed by using the normal vectors at the previous and the current frames. We then calculate the intersection points between the light disk and a plane formed by the two reflection vectors, as shown in Fig. 5.4. The light disk contribution between two reflection vectors can be calculated as follows. First, we calculate two intersection points p_c and p_p between the two reflection vectors and the plane. Next, we calculate the overlapping line segment between p_cp_p and the light disk. Then the arc is subtended by p_cp_p (the whole arc), and that subtended by the overlapping line segment (the light arc) is calculated. The light source's fractional contribution between the successive frames is obtained by the ratio of the angle of the light arc to that of the whole arc, as shown in Fig. 5.4.

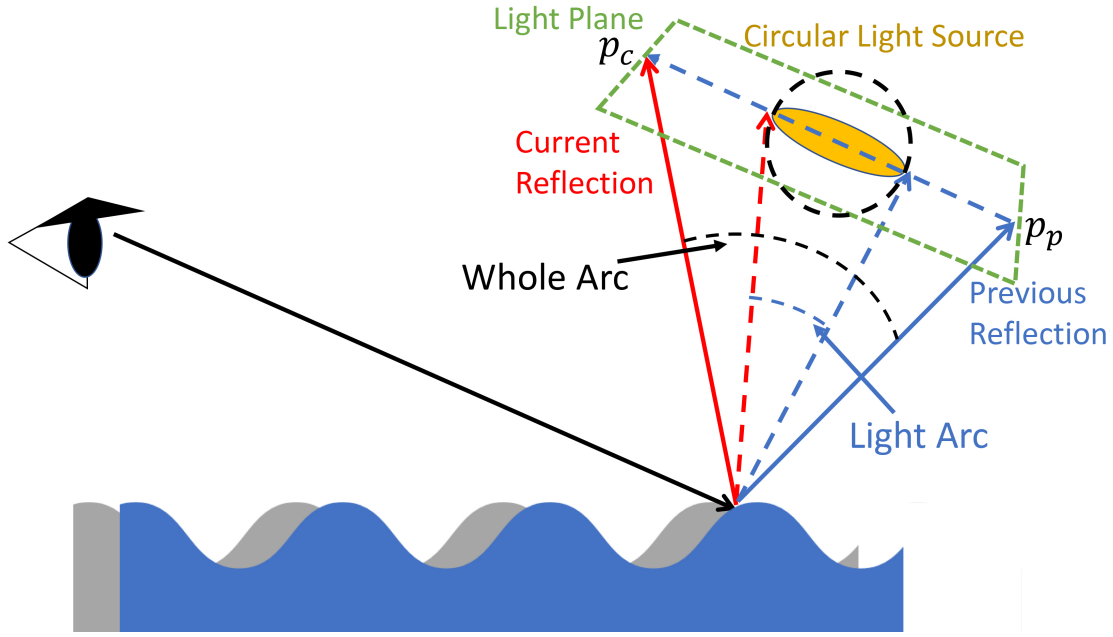


Figure 5.4: Our method uses a simple geometric analysis to find the contribution of light between the current reflection and the previous reflection.

5.1.3 Spatio-temporal-aliasing wave rendering

For a pixel that contains any aliasing waves, a single sample point per pixel is not sufficient. We have to compute the average intensity of the reflected light over the pixel area, taking into account the distribution of the surface normals. A straightforward solution is to generate multiple rays for each pixel as in [16], which significantly increases the computation time. Instead, we borrow the idea of the LEAN mapping technique [56] for efficient computation, which approximates the distribution as a Gaussian distribution. However, this approximation removes details from the surface and blurs the reflection result, which makes its temporal reflection changes less noticeable. Thus, our method assumes that aliasing waves are unchanging in time, and only non-aliasing waves contribute temporal changes to its pixel's reflection result. In a pixel that contains both types of waves, we need to solve spatial aliasing from aliasing waves and temporal aliasing from non-aliasing waves. To eliminate spatial aliasing, our method approximates an aliasing wave as a distribution function of its surface normal. This distribution itself doesn't change in time, but it is being moved by non-aliasing waves, which is according to our assumption. Our method then removes temporal aliasing by computing the time integration reflec-

tion of the distribution and the light source. Figure 5.5 shows the outline of our spatio-temporal-aliasing wave rendering method.

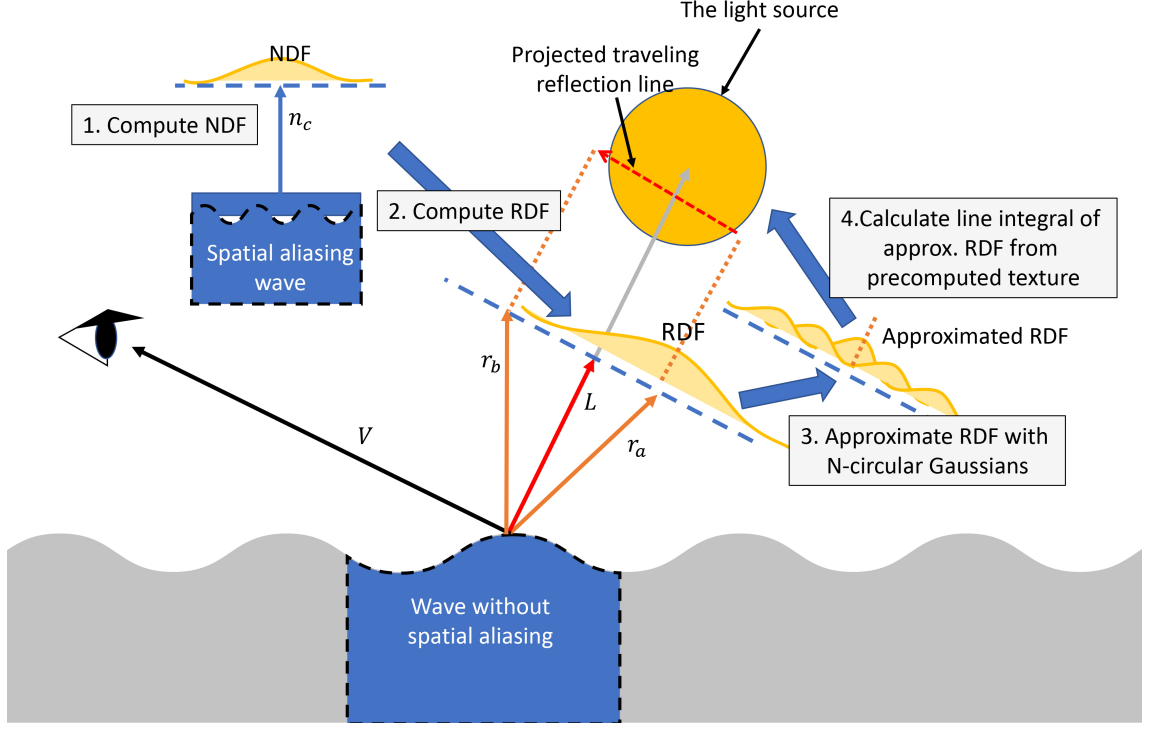


Figure 5.5: The outline of our spatio temporal waves rendering method.

We first calculate the normal distribution function (NDF) of the aliasing waves. The NDF is defined as a distribution of normal vector on a horizontal xz plane, assuming the y axis points in the upward direction (see Fig. 5.6). We call this the NDF plane. The NDF for the sum of the sine waves is obtained by accumulating the NDF for each sine wave. So, let us consider the NDF for the following single sine wave:

$$y(x, z, t) = A \sin(ax + bz + ct), \quad (5.1)$$

where A is an amplitude and t is time. The wave vector (a, b) determines the wavelength and direction of the wave, and c together with the wave vector determines the velocity. We approximate the NDF by a two-dimensional elliptical Gaussian function of a point $\mathbf{n}$ on the NDF plane:

$$\mathcal{G}_{\Sigma, \mathbf{n}_c}(\mathbf{n}) = \frac{1}{2\pi|\Sigma|^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{n}-\mathbf{n}_c)^t \Sigma^{-1}(\mathbf{n}-\mathbf{n}_c)}, \quad (5.2)$$

where $\mathbf{n}_c$ is the point corresponding to the normal vector without aliasing waves (*i.e.*, the low frequency components) and is the center of the Gaussian on the NDF plane.

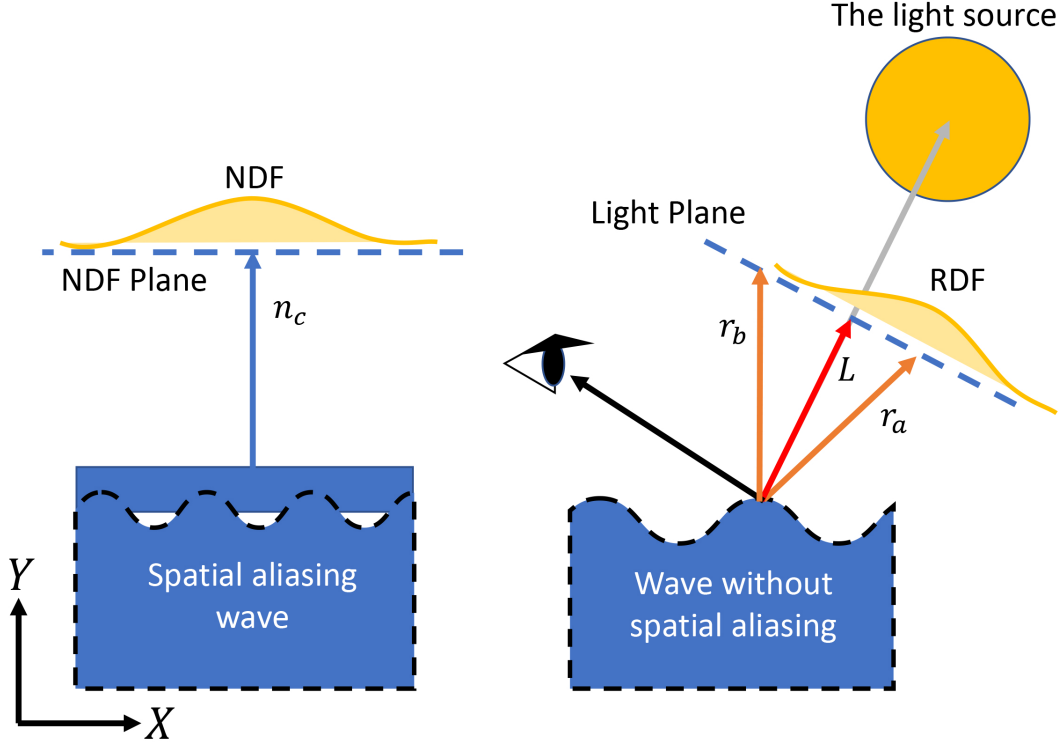


Figure 5.6: Our NDF is a normal distribution function of spatial aliasing waves. Our RDF is a reflection distribution function of the NDF. It is defined on a light plane perpendicular to the direction to the center of the light source.

Σ is the covariance matrix of the elliptical Gaussian and can be expressed differently by our two approximation methods: uniform method and per-pixel method.

Our uniform method assumes that each aliasing wave has its whole wave cycle inside each pixel, thus its Σ can be expressed by:

$$\begin{aligned} \Sigma &= \lim_{T \rightarrow \infty} \frac{1}{T} \begin{bmatrix} \int_0^T \left(\frac{\partial y}{\partial x}\right)^2 dt & \int_0^T \frac{\partial y}{\partial x} \frac{\partial y}{\partial z} dt \\ \int_0^T \frac{\partial y}{\partial x} \frac{\partial y}{\partial z} dt & \int_0^T \left(\frac{\partial y}{\partial z}\right)^2 dt \end{bmatrix} \\ &= \begin{bmatrix} \frac{A^2 a^2}{2} & \frac{A^2 ab}{2} \\ \frac{A^2 ab}{2} & \frac{A^2 b^2}{2} \end{bmatrix} = \begin{bmatrix} \sigma_x^2 & \sigma_x \sigma_z \\ \sigma_x \sigma_z & \sigma_z^2 \end{bmatrix}, \end{aligned} \quad (5.3)$$

where $\sigma_x = Aa/\sqrt{2}$ and $\sigma_z = Ab/\sqrt{2}$, and the temporal average over t is the same as the spatial average over x and z because of the form of Eq. (5.1).

Our per-pixel method finds the beginning and ending of the sine wave within the pixel, where Σ can be defined more accurately at the cost of increased computation.

The covariance matrix Σ can be instead expressed by:

$$\Sigma = \begin{bmatrix} M.xx - B.x * B.x & M.xz - B.x * B.z \\ M.xz - B.x * B.z & M.zz - B.z * B.z \end{bmatrix}, \quad (5.4)$$

$$B = (B.x, B.z) = \frac{1}{p_2 - p_1} \left(\int_{p_1}^{p_2} \frac{\partial y}{\partial x} dp, \int_{p_1}^{p_2} \frac{\partial y}{\partial z} dp \right), \quad (5.5)$$

$$\begin{aligned} M &= (M.xx, M.xz, M.zz) \\ &= \frac{1}{p_2 - p_1} \left(\int_{p_1}^{p_2} \left(\frac{\partial y}{\partial x} \right)^2 dp, \int_{p_1}^{p_2} \frac{\partial y}{\partial x} \frac{\partial y}{\partial z} dp, \int_{p_1}^{p_2} \left(\frac{\partial y}{\partial z} \right)^2 dp \right), \end{aligned} \quad (5.6)$$

$$p(x, z) = ax + bz, \quad (5.7)$$

where p_1 and p_2 are the beginning and the ending of the wave within the pixel respectively. However, p_1 and p_2 of each wave within the pixel are varying depend on a position. To simplify the problem, we only consider the position at the pixel center. Eq. (5.5) and (5.6) can be solved analytically, however we omit the detail here due to its length. Appendix A.1 shows the solution to the equations and explains how we find p_1 and p_2 in more detail. The NDF for the sum of the waves is approximated as the sum of the NDFs for each of them.

The NDF computed above is then transformed into the reflection space to obtain the reflection distribution function (RDF), which represents the distribution of the reflection directions. The RDF is defined on a light plane perpendicular to the unit direction vector to the center of the light source and is located at the endpoint of this unit vector, as shown in Fig. 5.6. We explain the derivation of our transformation function below (see also Fig. 5.7).

Let us consider a function Φ that maps point $\mathbf{n}$ on the NDF plane to point $\mathbf{r}$ on the light plane, *i.e.*, $\mathbf{r} = \Phi(\mathbf{n})$. $\Phi(\mathbf{n})$ can be derived from geometrical analysis of Fig. 5.7, as follows:

$$\Phi(\mathbf{n}) = \frac{r(\mathbf{n})}{r(\mathbf{n}) \cdot \mathbf{L}}, \quad (5.8)$$

$$r(\mathbf{n}) = 2(n(\mathbf{n}) \cdot \mathbf{v})n(\mathbf{n}) - \mathbf{v}, \quad (5.9)$$

$$n(\mathbf{n}) = \frac{\mathbf{n}}{|\mathbf{n}|}, \quad (5.10)$$

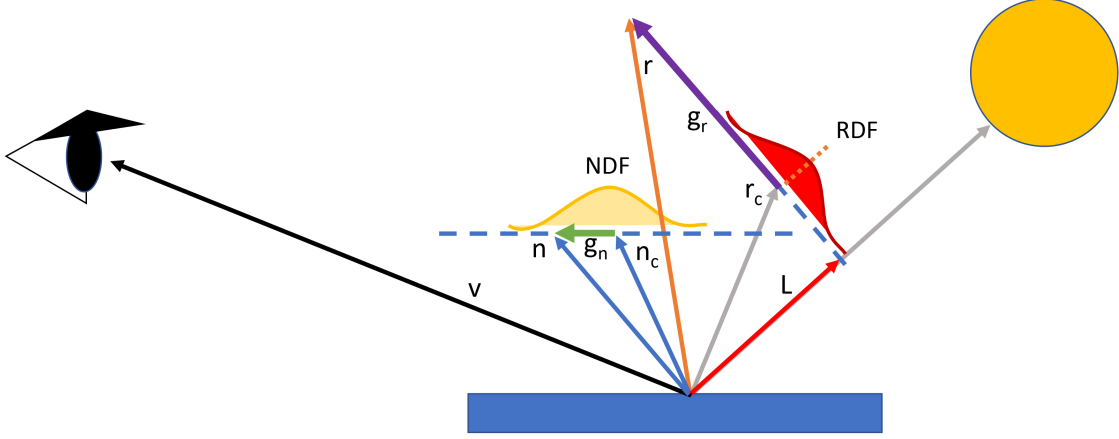


Figure 5.7: Transformation from NDF to RDF.

where $\mathbf{v}$ is the eye direction and $\mathbf{L}$ is the light direction. As you can see in Eq. (5.8) through Eq. (5.10), Φ is a complicated nonlinear function, so we employ a first order approximation. Let $\mathbf{n}_c$ be the center of the NDF, $\mathbf{r}_c = r(\mathbf{n}_c)$ be the center of the RDF, and $\mathbf{g}_n = \mathbf{n} - \mathbf{n}_c$, so that $\mathbf{n} = \mathbf{n}_c + \mathbf{g}_n$. Then the approximation is:

$$\Phi(\mathbf{n}) \approx \Phi(\mathbf{n}_c) + J_c \mathbf{g}_n = \mathbf{r}_c + J_c \mathbf{g}_n, \quad (5.11)$$

$$J_c = \frac{\partial \varphi}{\partial \mathbf{g}_n}(0), \quad (5.12)$$

$$\varphi(\mathbf{g}_n) = \mathbf{g}_r = \mathbf{r} - \mathbf{r}_c = r(\mathbf{g}_n + \mathbf{n}_c) - (2(\mathbf{n}_c \cdot \mathbf{v})\mathbf{n}_c - \mathbf{v}), \quad (5.13)$$

where J_c is a Jacobian matrix of $\varphi(\mathbf{g}_n)$ computed at $\mathbf{g}_n = 0$. $\varphi(\mathbf{g}_n)$ is a function that maps point $\mathbf{g}_n$ on the NDF plane to point $\mathbf{g}_r = \mathbf{r} - \mathbf{r}_c$ on the light plane. Then, Φ^{-1} is approximately expressed by:

$$\Phi^{-1}(\mathbf{r}) \approx \mathbf{n}_c + J_c^{-1} \mathbf{g}_r. \quad (5.14)$$

With this linear approximation, the RDF can also be expressed by another elliptical Gaussian function with covariance matrix Σ_r . That is, the RDF is expressed by:

$$\begin{aligned} RDF(\mathbf{r}) &= NDF(\Phi^{-1}(\mathbf{r})) \\ &\approx \frac{1}{|J_c^{-1}|} \mathcal{G}_{\Sigma_r, \mathbf{r}_c}(\mathbf{r}) \end{aligned} \quad (5.15)$$

$$\Sigma_r = J_c \Sigma J_c^t \quad (5.16)$$

Note that $\mathbf{r}_c$ is the center of the above Gaussian function on the light plane. The accuracy of our RDF approximation is shown later in Appendix A.2.

We can now compute the intensity of the reflected light using the convolution between the RDF and the light disk. The light disk here is the projection of the spherical light source onto the light plane. However, to remove the temporal aliasing, we need to compute the average intensity over the time interval between the frames. That is, the intensity I is expressed by:

$$I = \frac{L_l}{T} \int_{t-T}^t \int_{\mathbf{r} \in D} RDF(\mathbf{r}, t^*) d\mathbf{r} dt^* \quad (5.17)$$

where L_l is the radiance of the light source and T is the time interval between the frames. D represents the circular region within the light disk. Note that we include the time parameter t^* for RDF to explicitly indicate that it is time-varying.

The above equation is too expensive to compute in real-time. We, therefore, precompute the integral under some simplifying assumptions. We assume that the path of the reflection direction on the light plane during the frame is a straight line segment. The accuracy of this approximation can be improved by dividing the frame time into subintervals, each with its straight segment. Also we approximate the elliptical Gaussian RDF by a circular Gaussian, with variance $\sigma_r^2 = \sigma_s \sigma_t$, where σ_s and σ_t are the eigenvalues of the covariance matrix Σ_r in Eq. (5.15). Later we will approximate the elliptical Gaussian more accurately as a weighted sum of circular Gaussians. Finally, we assume that the temporal changes in the Jacobian J_c and covariance matrix Σ_r are small. So J_c and Σ_r are also assumed to be constant between the neighboring frames. We only consider the change in the primary normal direction $\mathbf{n}_c$ (and therefore the primary reflection direction $\mathbf{r}_c$, *i.e.*, the center of the Gaussian RDF).

5.1.4 Circular Gaussian approximation

For the circular Gaussian, we need to precompute Eq. (5.17), which is an integral, over the reflection vector segment for the frame time of the RDF convolved with the light disk. Because this convolution is circularly symmetric, the integral can be precomputed using three parameters: the closest distance of the line containing the segment to the center of the convolution, the start position of the segment on that line, and the end position. (See Fig. 5.8.) Considering also the variance σ_r^2 for the RDF, this gives four parameters.

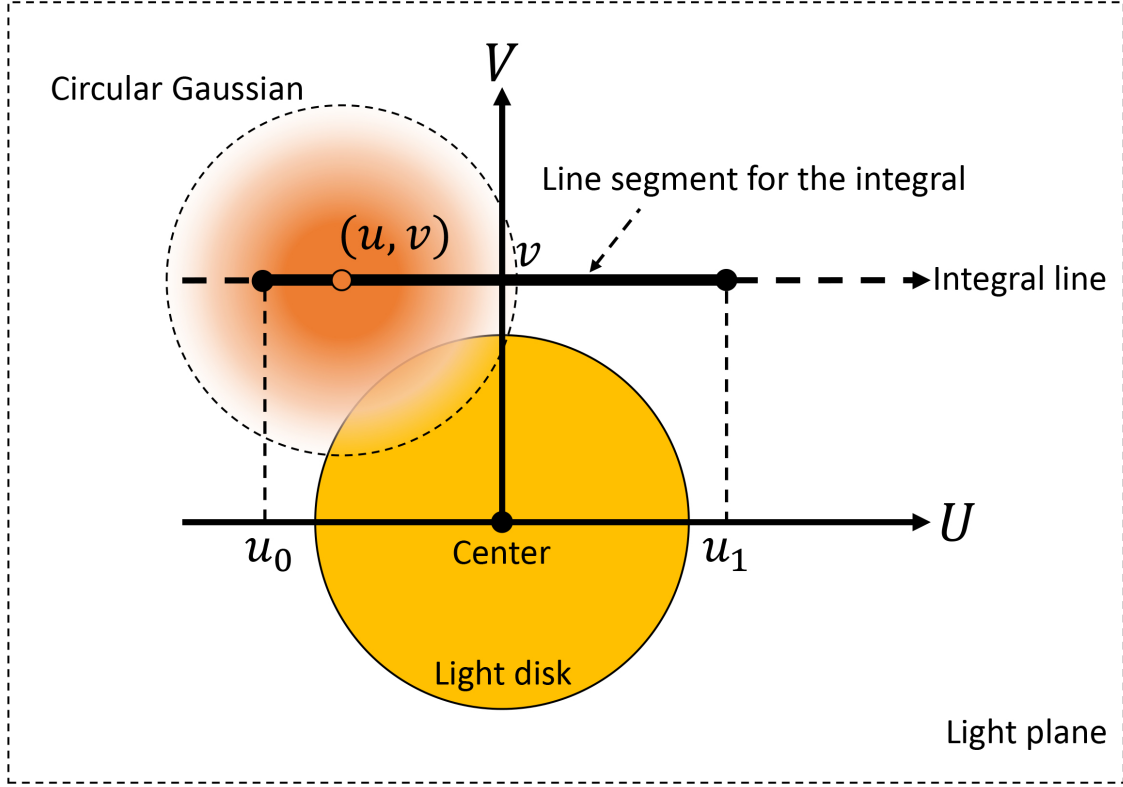


Figure 5.8: Precomputation for the line integral, which is an integral of a convolution of the light disk and the circular Gaussian that is moving on the line segment.

Fortunately, we can reduce these to three parameters. The straight segment integration can be separated into four cases: a segment that starts and ends at the outside but cuts through the convolution, a segment that starts at the outside but ends inside the convolution, which, by reflection symmetry across the V axis in Fig. 5.8, also included the segment which starts inside and ends outside, a segment that starts and ends inside the convolution, and a segment that starts and ends outside the convolution without cutting through it, which has integral zero. The first case has one parameter, which is the closest distance of the segment to the center of the convolution. The second case has two parameters, which are the closest distance of the segment to the center of the convolution and the end position of the segment. The third case has the three parameters we mentioned earlier. However, we can calculate the third case integral by first finding the integral over a segment that starts from an infinite distance and goes to the end position while passing through the start position of the integral segment. Then, we subtract another integral on the same line that starts from an infinite distance and goes to the start position of

the segment. In other words, an integral of the third case can be calculated from the difference of two second case integrals. Thus, we only need to precompute for the first and the second case, and the first one is a special case of the second case. The second case needs two parameters, which are the closest distance and the end position of the segment. Thus, including the variance too, we can store the line integral into a 3D texture, as described below.

Let us consider local coordinates UV on the light plane with its origin at the center of the light disk (see Fig. 5.8). Because of circular symmetry, we can assume that the line segment for the line integral is parallel to the U axis and intersects the V axis at $V = v$, shown as the thick black line segment in the figure. The center of the circular Gaussian RDF moves along the integral line. Our method precomputes the following table for various variances.

$$S(u, v, \sigma_r) = \int_{-R}^u F(u', v, \sigma_r) du', (-R < u, v < R), \quad (5.18)$$

where $F(u, v, \sigma_r)$ corresponds to the inner integral of Eq. (5.17), *i.e.*, the convolution of the light disk with the circular Gaussian whose center is at (u, v) . R is a large value chosen such that $F(R, 0, \sigma_r)$ is sufficiently small. With this table, the intensity expressed by Eq. (5.17) is obtained by:

$$I = S(u_1, v, \sigma_r) - S(u_0, v, \sigma_r), \quad (5.19)$$

where (u_0, v) and (u_1, v) correspond to the two primary reflection directions at the previous and the current frames, exchanged if necessary to make $u_1 \geq u_0$. Our method stores S in a 3D texture on the GPU. The precomputation only considers a constant light disk radius, but we can linearly transform the light disk radius and the RDF variance together. Thus, we can transform various input parameters to fit our precomputed data before using it. Our method supports changing both light and wave parameters in real-time.

5.1.5 Elliptical Gaussian approximation

We needed to approximate an elliptical Gaussian RDF as a circular Gaussian to keep the precomputed table to 3 dimensions, which has hardware support on a GPU. However, this approximation loses its accuracy when an RDF has a very elongated

distribution in one of its axes, which happens when the viewing direction is close to the horizon. So we approximate an elliptical Gaussian RDF more closely with a method similar to Feline mapping [47], which uses multiple circular Gaussian kernels to approximate a single elliptical Gaussian kernel. However, we improve on [47] by non-linearly optimizing the circular Gaussian kernel weights, center positions, and variances so that their combination matches as close as possible to the RDF, which is an elliptical Gaussian kernel. Figure 5.9 illustrates the idea. The details of this fit are given in the appendix A.3, and produce curve fits for the circular Gaussian parameters.

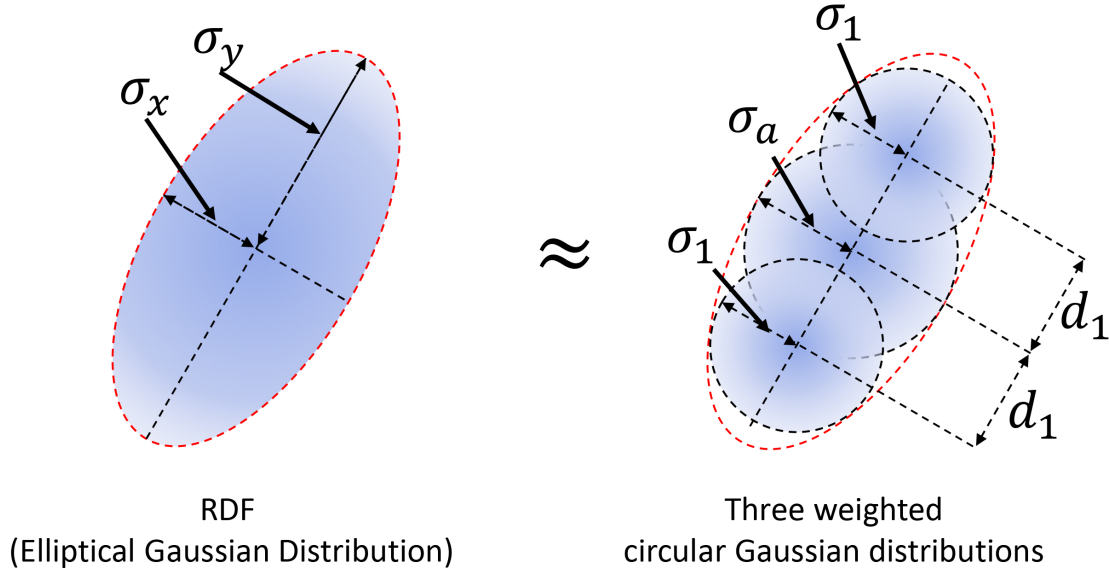


Figure 5.9: We further approximate our RDF, which is defined as an elliptical Gaussian distribution, as a combination of multiple weighted circular Gaussian distributions. In this example, we approximate RDF as 3-weighted circular Gaussian distributions.

When we are rendering the scene, after we obtain the RDF, we scale the whole RDF so that its minor axis variance is equal to 1. Then we find the optimal circular Gaussian kernel parameters by inserting the scaled major axis variance of the RDF into each curve fitted equation of each parameter as a function of the major axis variance. Next, we scale the optimal parameters back to the original scale at the beginning. Then we find the line segment of each circular Gaussian kernel by offsetting the segment from the original segment with a vector from the center of the RDF

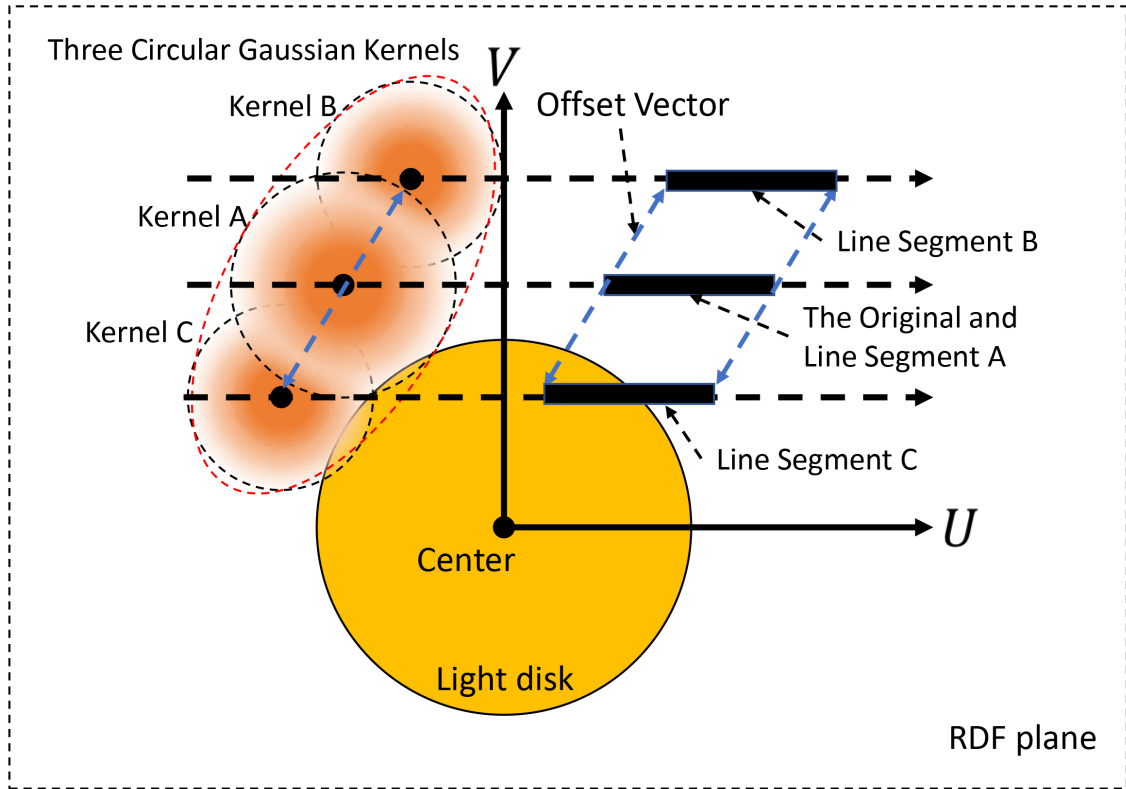


Figure 5.10: A, B, and C circular Gaussian kernels are used to approximate the RDF. Each circular Gaussian kernel has its own integral segment, which can be calculated by offsetting the original integral segment with the vector from the center of the RDF to the circular Gaussian kernel.

to the center of each circular Gaussian kernel. Figure 5.10 shows an example of the line segment offsetting for the three circular Gaussian kernels case. Finally, we use our circular approximation method to find the result with the adjusted segment for each circular Gaussian kernel and combine them according to the unscaled optimal weight parameters.

However, there is a limitation to this method. An elliptical Gaussian kernel with a more elongated shape will require more circular Gaussian kernels, which increases computational time. To have stable performance, we decide to use a fixed number of circular Gaussian kernels. (Nine in our implementation.) Then if the elongation is more than a threshold, we use our fallback method instead. The elongation is a ratio between a major and a minor axis standard deviation of an elliptical Gaussian kernel. The threshold we used is a minor axis standard deviation, which is not yet scaled, multiplied by a fixed number of circular Gaussian kernels.

Our fallback method is computing a convolution of the RDF and the light source with the Riemann sum method. We ignore the temporal integration in our fallback method because the reflection from a high variance RDF is blurry, which is less likely to be improved by the temporal integration.

5.1.6 Aliasing wave transition

Our method treats non-aliasing waves and aliasing waves differently. The non-aliasing waves are used to determine temporal changes of the surface, while the aliasing waves are approximated by a Gaussian RDF. However, this approximation, which is in our spatio-temporal-aliasing wave rendering method, introduces a visual artifact to our result. This classification causes a different rendering result between two types of waves when we use the Nyquist limit to classify the type of each wave within each pixel. This classification switch is too sudden. Any wave with a projected frequency lower than the Nyquist limit is considered a non-aliasing wave, while the same wave in a neighbor pixel might have projected frequency equal or slightly above the Nyquist limit and be suddenly considered an aliasing wave. The difference between the two pixels will be apparent as seen in Fig. 5.11b.

To prevent this, instead of using the Nyquist limit, we use soft classification to smoothly transform non-aliasing waves to aliasing waves. We create a transition region, which determines a fractional aliasing value α between 0 to 1. The transition region is in frequency space and stationed before reaching the Nyquist limit. Any wave within the region is considered to contain both aliased and non-aliased waves with weights of α and $1 - \alpha$ respectively. If the transition region has a width of δ in frequency space and the Nyquist limit is f_n , the transition region will start at $f_n - \delta$ and end at f_n . A wave with a frequency of f within the region will have an aliasing value α given by

$$\alpha = \begin{cases} 0 & f < f_n - \delta \\ (f - (f_n - \delta))/\delta & f_n - \delta \leq f \leq f_n \\ 1 & f_n < f \end{cases} \quad (5.20)$$

For α between 0 and 1, we transform the sine wave into two waves, an aliased wave with amplitude weighted by α , and a non-aliased wave weighted by $1 - \alpha$.

We do this for every sine wave and combine the waves of the same type. If there aren't any aliased waves, the combined non-aliased waves will be rendered with our temporal-aliasing wave rendering method. However, if there are aliased waves, the combined aliased waves will be represented as an NDF, and the combined non-aliased waves will determine a reflection path in our spatio-temporal-aliasing wave rendering method. Figure 5.11 shows the result of our transition.

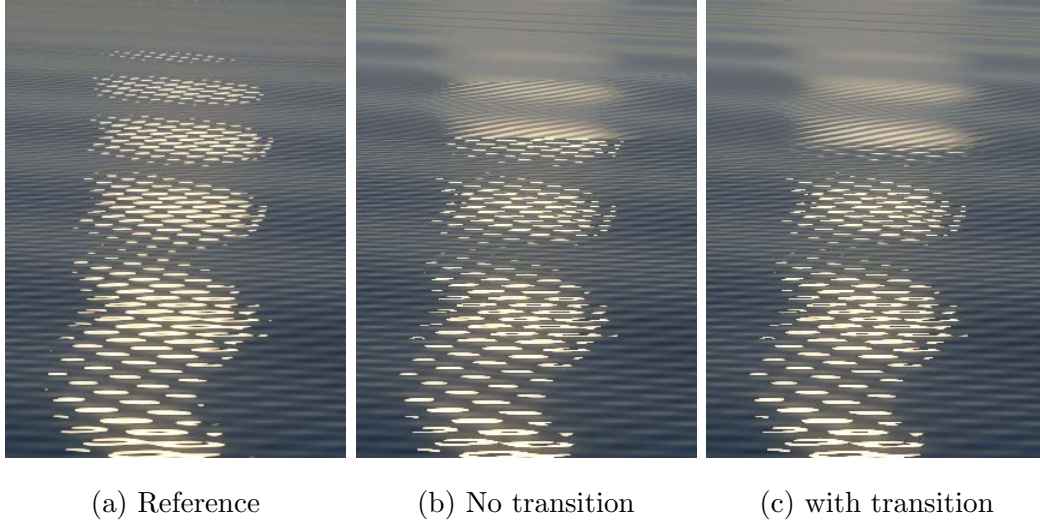


Figure 5.11: Comparison with/without transition regions.

5.2 Implementation

We implemented our method with OpenGL. The size of the transition region between the aliasing and non-aliasing waves can be adjusted to achieve a smooth transition between them. Our temporal-aliasing rendering method is faster than our spatio-temporal-aliasing wave rendering method. So we need to adjust the transition region to use the temporal-aliasing wave rendering method as much as possible. We experimentally determine the transition region to start at 0.2 cycles per pixel and finish at 0.5 cycles per pixel.

In our implementation of spatio-temporal-antialiasing, the size of the precomputed 3D texture for the circular Gaussian approximation is 512x512x64; the dimensions correspond to the closest distance, line segment endpoint, and variance of the circular Gaussian kernel, respectively. For the elliptical Gaussian approximation, we use 1, 3, 5, or 9 circular Gaussian kernels. However, we cannot use this 3D tex-

ture when there are no non-aliased waves that vary in time because we cannot find the point integral of a still reflection vector with our method (there is no reflection path). So we need to precompute another 2D texture that just stores a convolution result of a circular Gaussian kernel and a light disk with a resolution of 512x64; each dimension corresponds to the distance between the center of the light disk and the kernel, and variance of the circular Gaussian kernel, respectively.

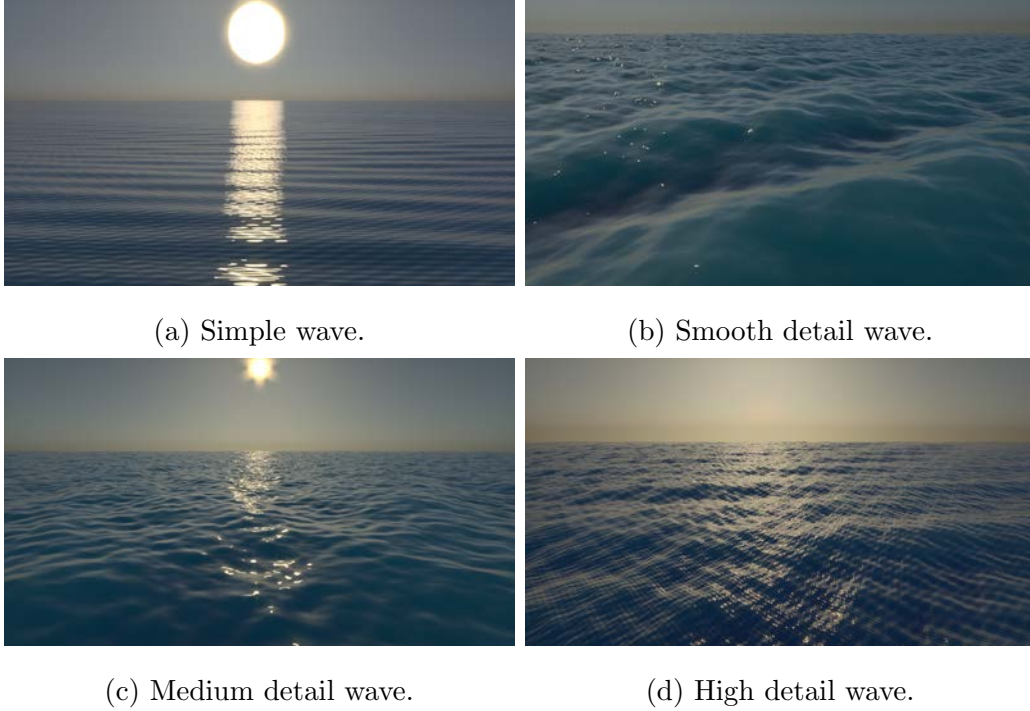


Figure 5.12: The preview of our four experiment scenes.

5.3 Results

We investigated the abilities of our method by using four scenes with a simple wave, a smooth detail wave, a medium detail wave, and a high detail wave (see Fig. 5.12). For each of the scenes, we created multiple images with six different rendering methods: the method without any anti-aliasing techniques, a spatial-only 32 multisample anti-aliasing (32x MSAA) method, LEAN mapping method, our method with uniform NDF, our method with per-pixel NDF, and a reference image. We generate a 4K normal map from sine wave data for LEAN mapping for each frame. Our method uses nine circular Gaussian kernels for the temporal-aliasing-wave rendering method. The reference image was created offline by simply

generating a supersampled image with 2x2, 4x4, or 8x8 times higher resolution and then downsampling it. We also included temporal integration in the offline rendering by generating 4 or 8 supersampled images between the neighboring frames. Glare and tone mapping post-processing were applied to all these images to show the sun's high brightness reflection. The rendering times are shown in the captions of the figures, while our precomputation time is 4.24 ms. The rendering times are measured on a PC with Intel Core i9-9900KF @3.6Ghz, Memory 16 GB, and NVIDIA TITAN RTX, 24 GB.

Figure 5.13 shows the comparison using the simple wave consisting of five sine waves with similar directions. This scene shows a simple and predictable reflection movement. Our method produces almost the same result as the reference image for the regions near the camera. In the far region, our elliptical Gaussian approximation removes the spatial aliasing. However, the other two methods produce severer spatial aliasing in both the near and the far regions.

Figures 5.14 and 5.15 use the smooth and the medium detail waves, respectively. These waves consist of 64 and 48 sine waves, respectively. These waves exhibit sparse reflections that are difficult for the previous methods to capture. Our method successfully renders an accurate image.

Figure 5.16 shows images rendered with the high detail wave consisting of 32 sine waves with a shorter wavelength. Again, our method can render the accurate reflection image near the camera, and the spatial aliasing is removed in the far region. The other two methods suffer from severe aliasing.

Table 5.1 and Fig. 5.18 show that our method produces results that are closer to the references than the others.

In addition to the RMSE error data, we also conduct a small survey with 22 respondents to perceptually evaluate the results. First, we show the result of MSAA method, LEAN mapping method, and our method with per-pixel NDF of each scene without showing their method name to the respondents. Second, we ask two questions: 'Which result has the most natural reflection?' and 'Which result has the highest resolution reflection?'. Third, we show the result of the reference method, MSAA method, LEAN mapping method, and our method with per-pixel NDF of each scene, but this time we tell the respondents which one is the reference

result. Last, we ask ‘Which result is the most similar to the reference result’. We didn’t include the method without any anti-aliasing techniques and our method with uniform NDF. Both have a similar result but respectively worse than MSAA method and our method with per-pixel NDF.

Fig. 5.19 shows the result of our survey.

	Scenes			
Method	Simple	Smooth	Medium	High
No AA	0.0797	0.0299	0.0315	0.0778
32x MSAA	0.0795	0.0287	0.0282	0.0676
LEAN	0.0675	0.0329	0.0298	0.0633
Our (uniform)	0.0384	0.0200	0.0278	0.0611
Our (per-pixel)	0.0384	0.0192	0.0229	0.0610

Table 5.1: RMSE of the result of each method when compared to the references.

5.4 Discussion

Our method particularly improves the rendering of the reflection near the camera, as shown in the examples in the previous section. However, the images of the reflection in the regions far from the camera are blurred due to our elliptical Gaussian approximation. This problem is reduced by using per-pixel NDF at the cost of more computational time. Per-pixel NDF doesn’t reduce the error as much as we expected, but it produces a sharp reflection result, which is visually similar to the reference.

Our elliptical Gaussian approximation increases the accuracy of the elongated reflection shape as we increase the number of circular Gaussian kernels. According to our experiments, three circular Gaussian kernels are enough, as shown in Fig. 5.17 but increasing the number of kernels can improve the quality of reflection at the horizon with little increased computational efforts. In Fig. 5.17, we also include an accurate convolution result of the Gaussian-approximated RDF with a light source, which is computed by using the Riemann sum method, for easier comparison.

This paper does not consider the effects of occlusion (from the viewpoint) or shadowing (from the light source) of one wave by another. Smith [67] developed

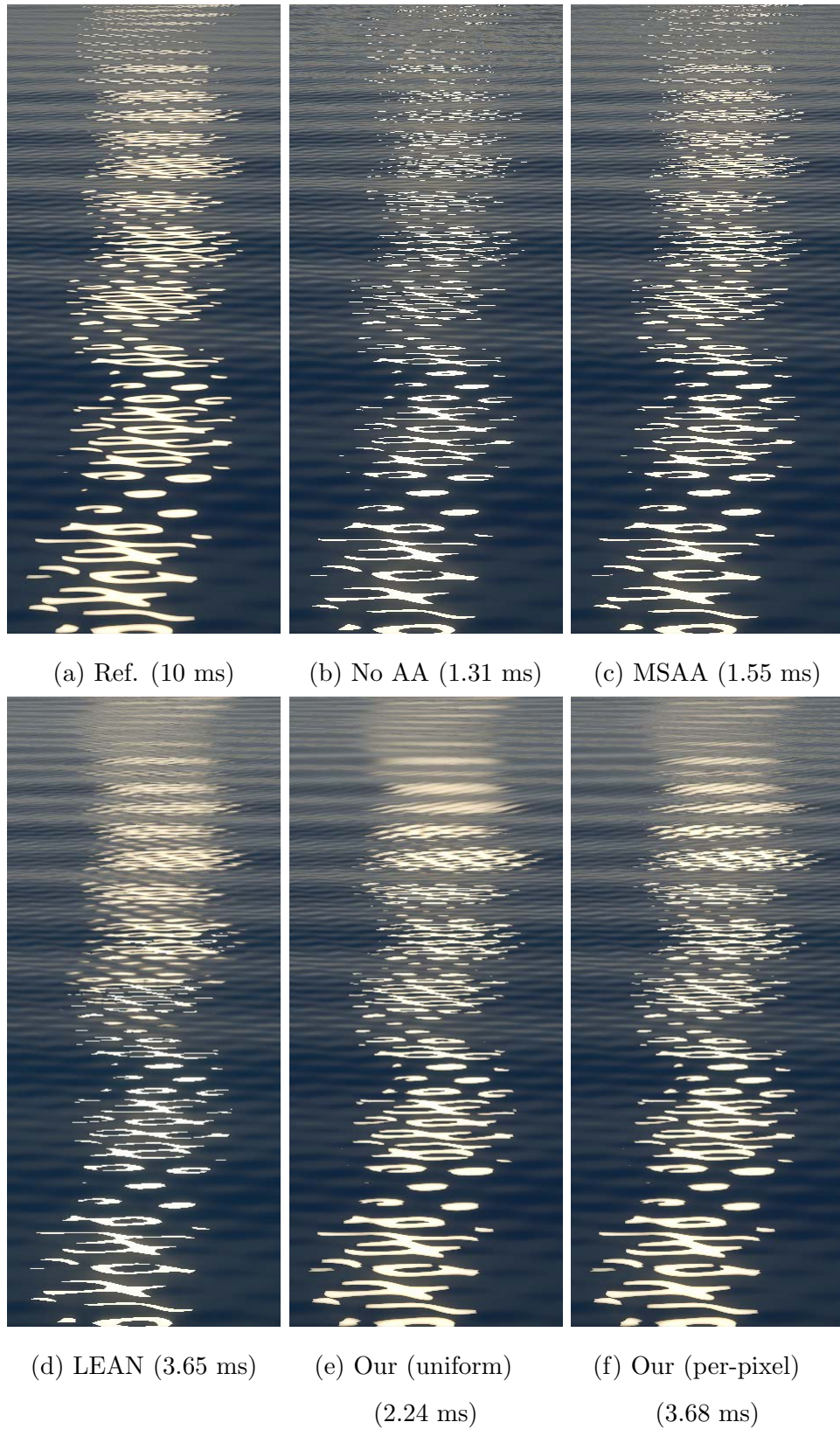


Figure 5.13: Here is the reflection result from the simple wave scene. The reference uses 8 spatial samples and 4 temporal samples.

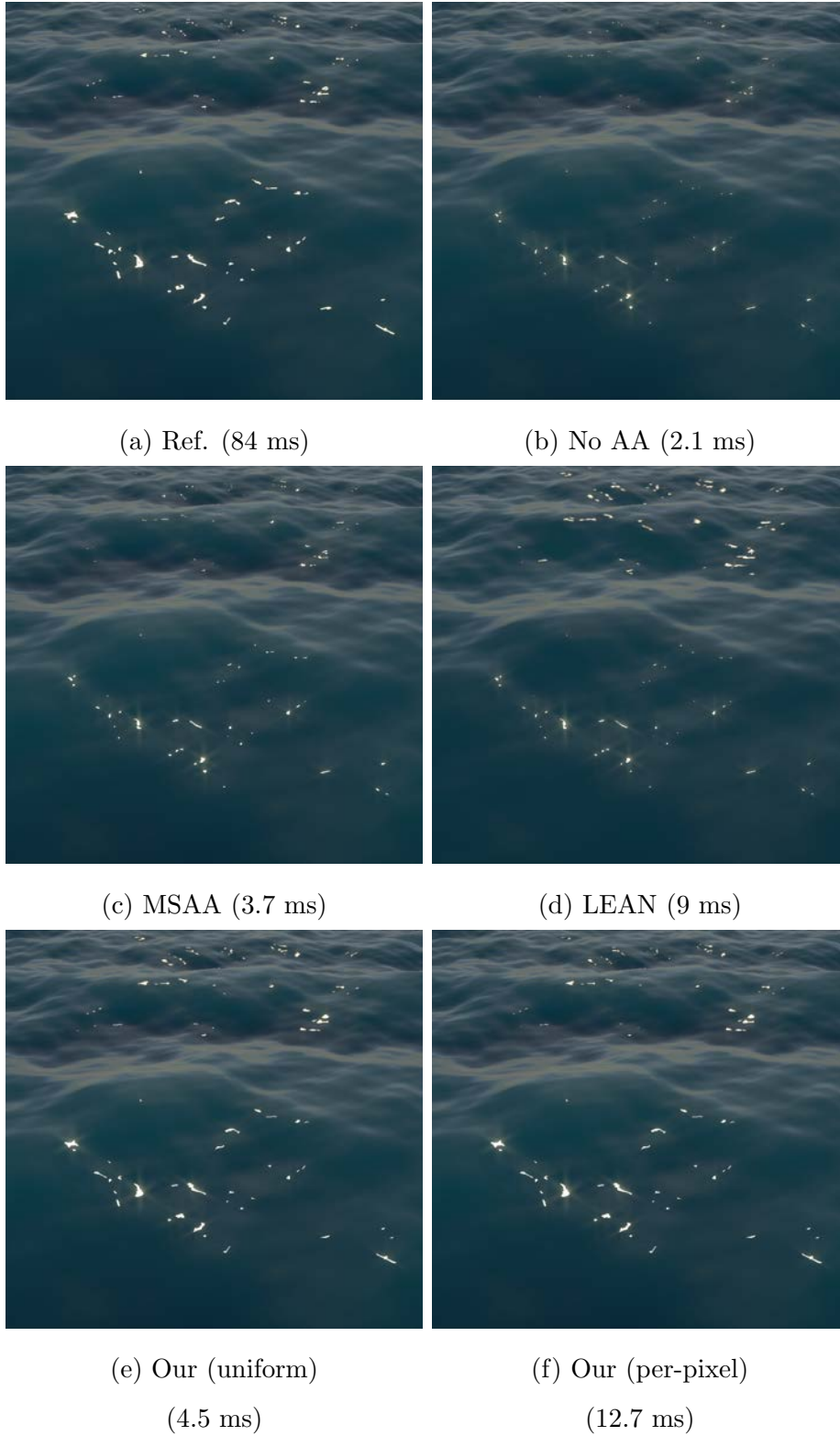


Figure 5.14: Here is the reflection result from the smooth detail wave scene. The reference uses 4 spatial samples and 8 temporal samples.



(a) Ref. (124 ms) (b) No AA (1.9 ms) (c) MSAA (3.4 ms)



(d) LEAN (7.9 ms) (e) Our (uniform) (4.5 ms) (f) Our (per-pixel) (10 ms)

Figure 5.15: Here is the reflection result from the medium detail wave scene. The reference uses 8 spatial samples and 8 temporal samples.

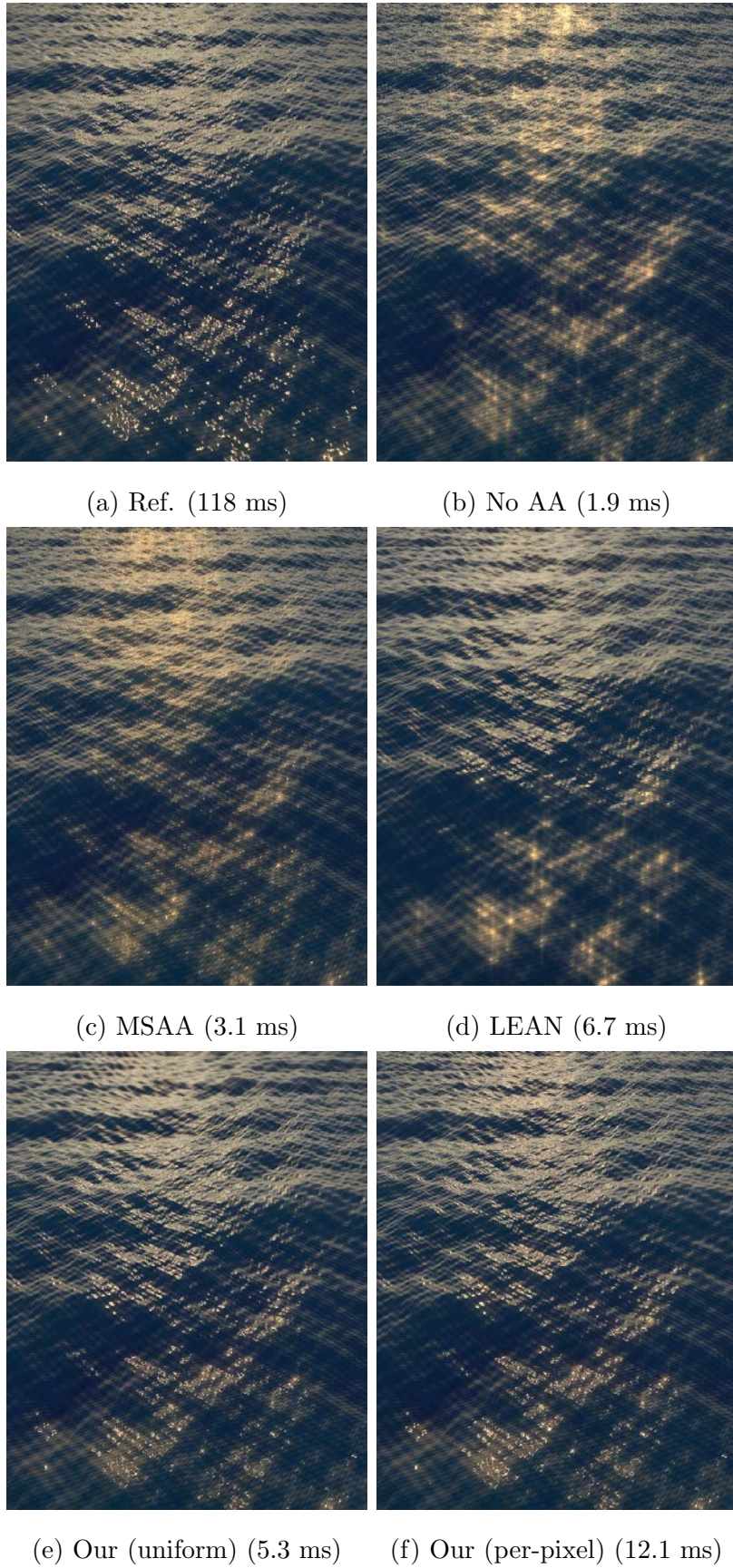


Figure 5.16: Here is the reflection result from the high detail wave scene. The reference uses 8 spatial samples and 8 temporal samples.

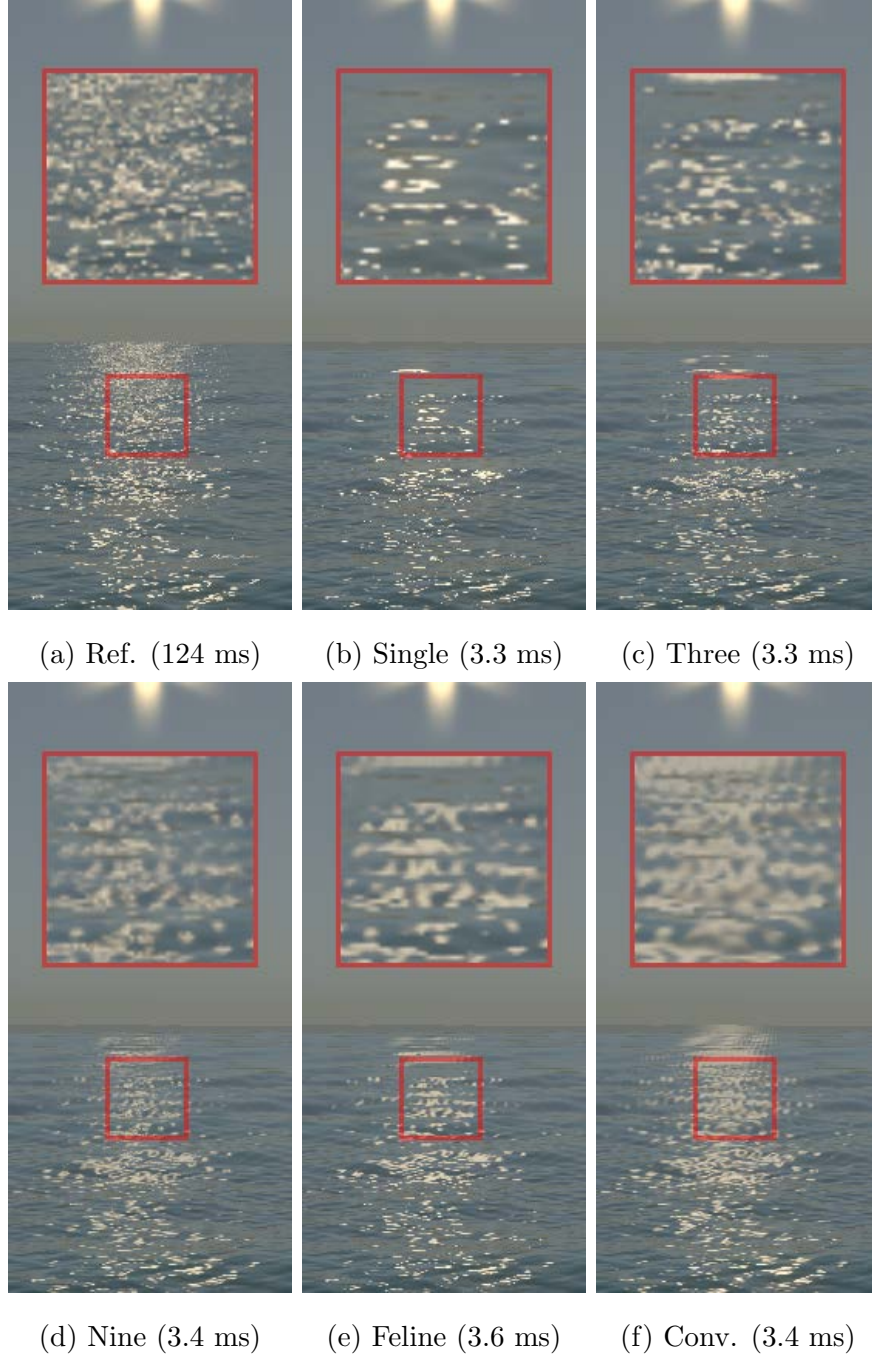


Figure 5.17: Here is the comparison of our approximation with different numbers of circular Gaussian kernels (without a fallback method), the Feline method, and the convolution. The reference uses 8 spatial samples. The RMSE of (b) to (f) are 0.0209, 0.0195, 0.0181, 0.0191, and 0.0183 respectively. (d) has the best performance for capturing and ability to compute time integral but (f) is better at capturing the reflection near the horizon. Thus, our method uses both (d) and (f).

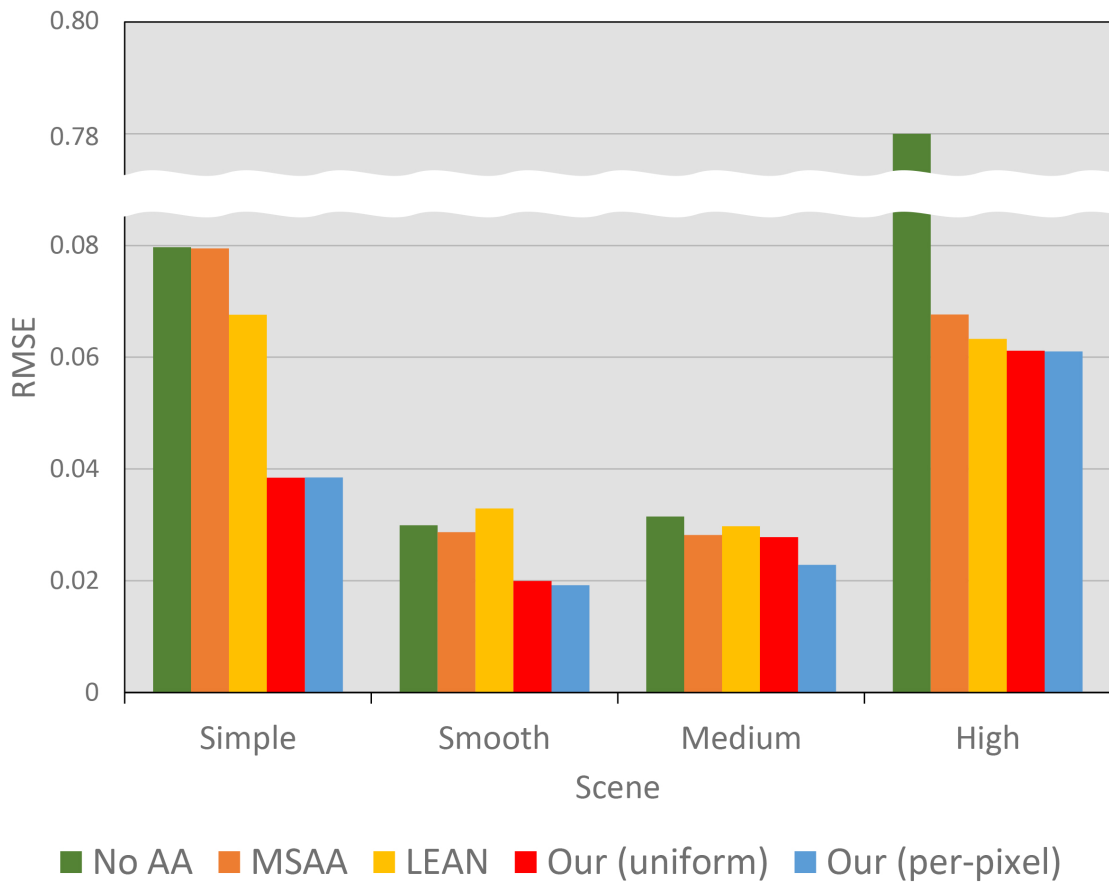


Figure 5.18: A bar chart of RMSE comparison data.

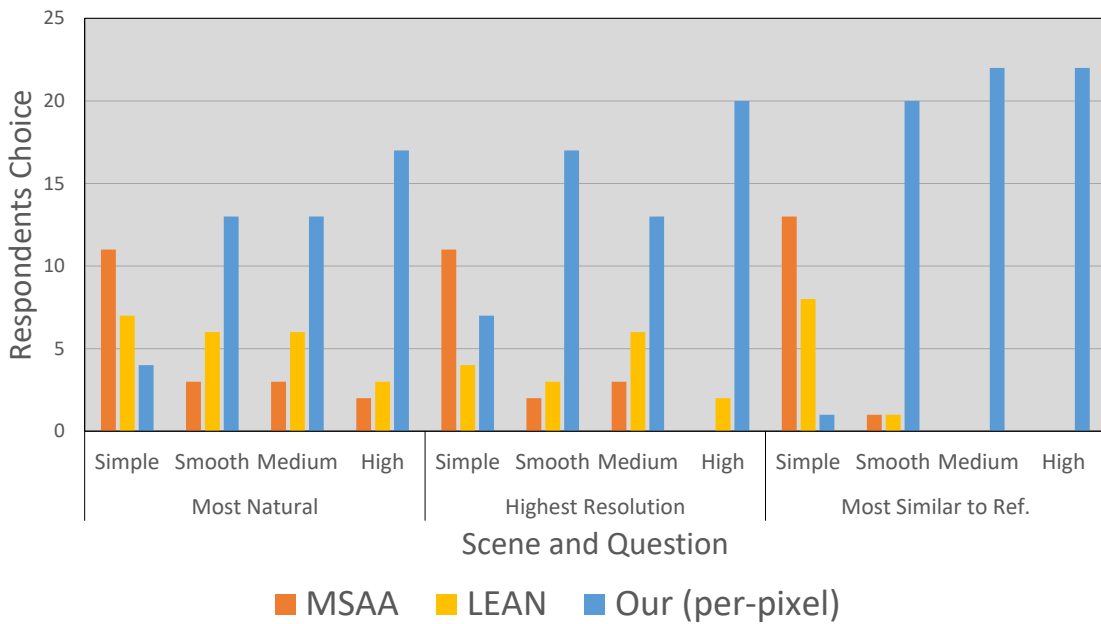


Figure 5.19: A bar chart of our survey data.

equations for these, which were correctly normalized by Ross [59], under the assumption that the waves were random and that the normals were uncorrelated with the wave height. These ideas were applied to rendering sea waves by Bruneton and Neyret [11], and to general surfaces by Dupuy *et al.* [20], enhancing LEAN mapping to “LEADR” mapping. They could be combined with our temporal anti-aliasing. However, when summing only a few sine waves, the heights and normals are, in fact, correlated. Becker and Max [6] handled visibility in the mesoscale where normal perturbation replaces displacement mapping by precomputing a redistributed normal map which accounted for occlusion but not shadowing, and this method can apply to any height field. However, it requires the precomputation of the visibility from any viewing angle, which can not currently be done in real-time for a time-varying height field.

Our survey also shows that our method has the most natural, the highest resolution, and the most similar result to the reference in three scenes except for the simple scene. Turn out the result of our method in this scene is the worst to most respondents. Our approach does well in most areas except for the reflection near the horizon, which attracts respondents’ attention due to its high detail. In this area, our method clamps some directional waves separately and essentially changes their overall directional because the simple scene only contains a small number of directional waves. Unlike LEAN mapping method, which over blurring the reflection but equally clamps all the waves together. MSAA method does well in the survey of this scene because its reflection with aliasing looks very similar to the detailed reflection of the reference.

5.5 Real-time Adaptive Supersampling

Our real-time spatio-temporal reflection anti-aliasing removes aliasing from water surfaces by using geometric simplification. But it also erases a detail from the surface, which causes over-blurring reflections in a result. Thus, we extended our method to solve this problem of our previous method.

We propose a method that utilizes both supersampling and geometric simplification to have a spatio-temporal anti-aliasing solution for rendering water surfaces comprised of sine waves. Our method is cost-effective yet provides detail and an accurate result.

Our main contributions are:

- A sine waves clamping method that removes waves causing aliasing and approximates their reflection contribution.
- A method that evaluates the reflection detail of each pixel.
- A method that computes parallel super sampling of each pixel proportionally to its evaluated detail.
- A demonstration that by combining our methods, we can render detailed anti-aliased reflections on water surfaces in real-time.

5.5.1 Proposed Method

Our main objective is to render the reflection of disk light sources on a water surface comprised of sine waves, without aliasing and without losing any details. An overview of our method will be explained first then we will go into the detail in each section.

Our method can be separated into two parts: an approximate reflection rendering and a detail reflection rendering. As shown in Fig. 5.20, our method starts with the approximated reflection rendering, which shares a lot of similarities to our previous work [57], to create a reflection result without aliasing and detail. Then, the detail reflection rendering evaluates the detail potential of each pixel of the approximated result. After that, it computes the multi-sampling of each pixel according to its

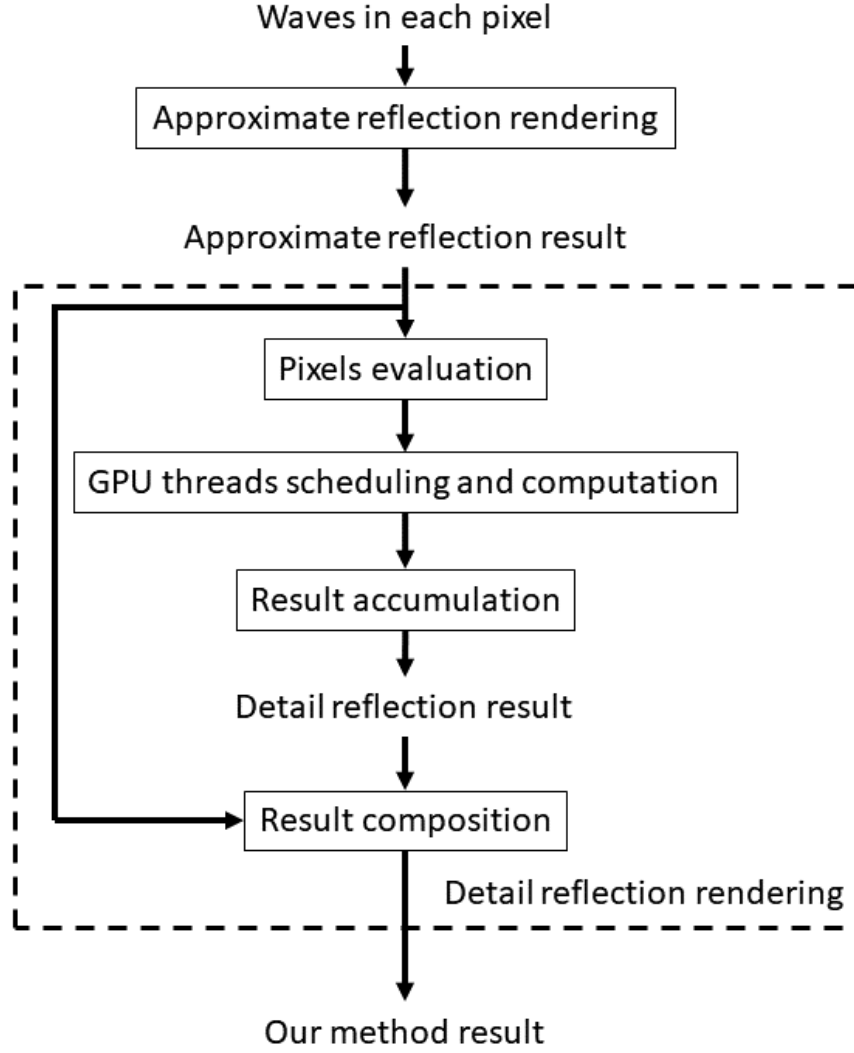


Figure 5.20: The overview of our method.

evaluation. In the end, the multi-sampling result is combined and composed with the approximated result to create a final result.

Approximate reflection rendering

In our method, the water surface is represented by a combination of sine waves with different frequencies and amplitudes. We want to remove spatial-aliasing by removing every sine wave that causes spatial-aliasing from the water surface geometry, and then compensating for its removal by adding its contribution to the LEAN mapping reflection rendering.

First, we utilize the clamping method [54] to identify which sine waves will cause spatial-aliasing on each pixel. To explain further, each sine wave in a pixel will

be projected to the screen space, then any sine wave whose projected frequency is larger than the Nyquist frequency is considered to be a spatial-aliasing wave. Every spatial-aliasing wave will be removed from the geometric calculation of the water surface.

Second, the clamped geometry will be rendered by our temporal ray tracing to remove any temporal-aliasing that it might have. Temporal ray tracing is done by calculating the time-integrated light contribution between the current frame and the previous frame. This can be done easily in our disk light source scenario. Our method calculates the intersection points between the plane of the disk light source and the reflected rays for both the current and the previous frames. Then we calculate the overlapped length between the disk light source and the line segment between the two intersection points. The time-integrated light contribution is calculated by the ratio of the length of the overlapped line segment to that of the line segment between the two intersection points.

Third, we have to compensate for the removed sine waves for any pixels where the sine waves causing spatial-aliasing. We combine the removed sine waves and approximate the normal distribution function (NDF) for those waves with a Gaussian distribution using the LEAN mapping method [56]. Then we transform the NDF to the reflection distribution function (RDF) by using the Gaussian kernel transformation technique in EWA volume splatting [89]. Then the light contribution can be calculated from a convolution of the RDF and the disk light source. We calculate this convolution by using numerical integration on a GPU.

Lastly, we can combine the result of temporal ray tracing and RDF-disk-light convolution to have a final approximated result. However, the result of the two methods is too different. Combining them directly will lead to a non-smooth result. Thus, we transition the wave clamping by gradually reducing the amplitude of each wave when its projected frequency gets close to the Nyquist frequency. With this method, we can blend the result between the two methods very smoothly.

Detail reflection rendering

The approximated result that we created earlier doesn't contain enough details in the reflection because the spatial-aliasing waves are approximated with a Gaussian

distribution NDF, which is not accurate enough. An accurate result can be obtained by supersampling each pixel. However, water surfaces are very complicated, and some pixels might need even more than 1000 samples, which is impossible in real time for all pixels with a traditional method. Our method tackles this problem by efficiently scheduling a million GPU threads to help compute a few pixels that make a difference in the detail of the overall reflection result.

First, our method evaluates the importance value of each pixel by using its pixel value from the approximated result. In this current stage, we simply put the approximated result into a Gaussian function, so that a pixel on the edge of the light source reflection area is considered more important, and any pixel that is not in that reflection at all is ignored. This can be described as:

$$v_i = g_\rho(R_a(i) - c), \quad (5.21)$$

where v_i is the importance value of i -th pixel, g_ρ is a 1D Gaussian function with variance of ρ^2 , $R_a(i)$ is the approximated result of pixel i , and c is a center of the Gaussian function. We adjust both ρ and c manually for each scene. Any pixel that has an importance value above a threshold will generate a sampling task, which contains its pixel position, sampling position, and the importance value. Every sampling task will be added to a sampling task list.

Second, our method calculates a prefix sum of the importance value of all tasks in the sampling task list. This can be done in parallel on a GPU as well.

Third, our method dispatches a fixed number of GPU threads to work on sampling tasks. Our objective is to assign a task to a group of threads whose size is as proportional as possible to the task's importance value. This can be done by associating each GPU thread id to each task's prefix sum value of importance value. Each thread first calculates the following searching value:

$$s_k = k \sum_{j=0}^N v_j, \quad (5.22)$$

where k is a GPU thread id, N is the size of the sampling tasks, v_j is the importance value of j -th task. Then we use a binary search to find a task that has the maximum prefix sum value that is still lower or equal to its s_k value. Now each GPU thread knows which task to execute.

Fourth, each GPU thread computes its task by randomly sampling the position within the task's pixel then using our temporal ray tracing to shade that ray.

Fifth, the result from each GPU thread is atomically collected into the detail result.

Last, the detail result is blended with the approximated result by using a weighting value for the detail result. We can calculate the weight a of i -th pixel of the detail result from the ratio of a size of threads that working on this pixel and the sample size needed to satisfy the Nyquist limit, which is:

$$a_i = \text{clamp}(M * \frac{v_i}{\sum_{j=0}^N v_j} * \frac{f_{nyquist}}{\max(f_k)}, 0, 1), \quad (5.23)$$

where M is a total thread size, v_i is the importance value of i -th pixel, f_k is a projected frequency of k -th sine wave, and $f_{nyquist}$ is the Nyquist frequency. The blended result R of i -th pixel will be:

$$R_i = (1 - a_i) * R_{a,i} + a_i * R_{d,i}, \quad (5.24)$$

where $R_{a,i}$ is the result of the approximate reflection rendering, and $R_{d,i}$ is that of the detail reflection rendering.

5.5.2 Results

We implemented our method using OpenGL, and our method runs entirely on a GPU. We currently experiment with just one scene, which is a sunset scene, with parameters: $c = 0.1, \rho = 0.04$. Figure 5.21 shows the comparison result between the reference image, the result without anti-aliasing, the result with 32x MSAA, and the result with our method. The reference image is calculated by the tiling 128x supersampling method. 5.2 shows the computation time comparison of the mentioned methods. Figure 5.22 shows the internal result of our method. Figures 5.22a and 5.22b show the result from the approximate and the detail reflection rendering method respectively. Figure 5.22c shows the importance value of each pixel.

Our method can achieve a similar result and detail to the reference image while being in real-time, unlike the 128x supersampling method.

The rendering times are measured on a laptop with an Intel Core i7 @ 2.50Ghz, Memory 16 GB, and an NVIDIA GeForce GTX 860M.

Table 5.2: The computational time comparison.

Rendering Method	Rendering Time
Reference	8326 ms
No AA	3 ms
32x MSAA	10 ms
Our Method	15 ms

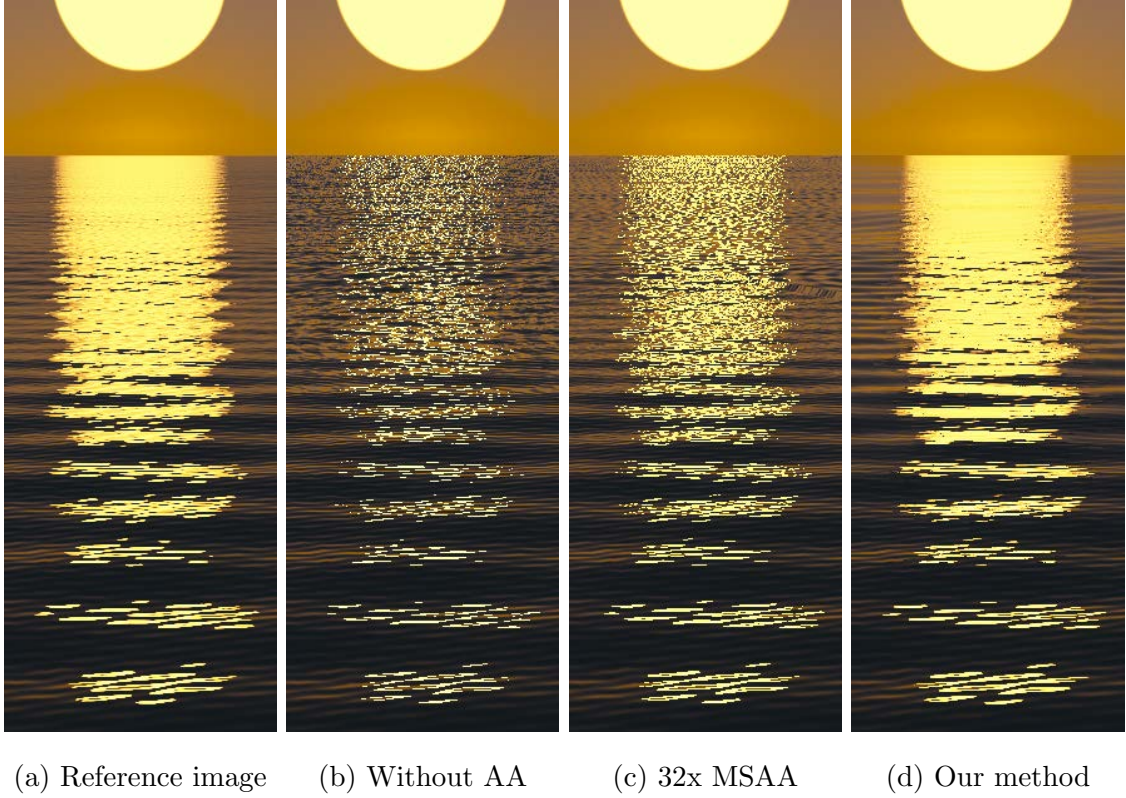


Figure 5.21: The comparison result from the sunset scene.

5.5.3 Discussion

We have presented a real-time rendering method for moving water surfaces with a highly detailed reflection of a disk light source. However, many problems remain, namely: pixel importance value evaluation, limited geometry, and light source support. In our current progress, our method parameters need to be adjusted manually for each scene to achieve the best result. Also, our method only supports sine waves for the water surface geometry and disk light source for a light source type. These problems make our method still impractical for many real applications.

We are working on fixing these problems and making our research useful in

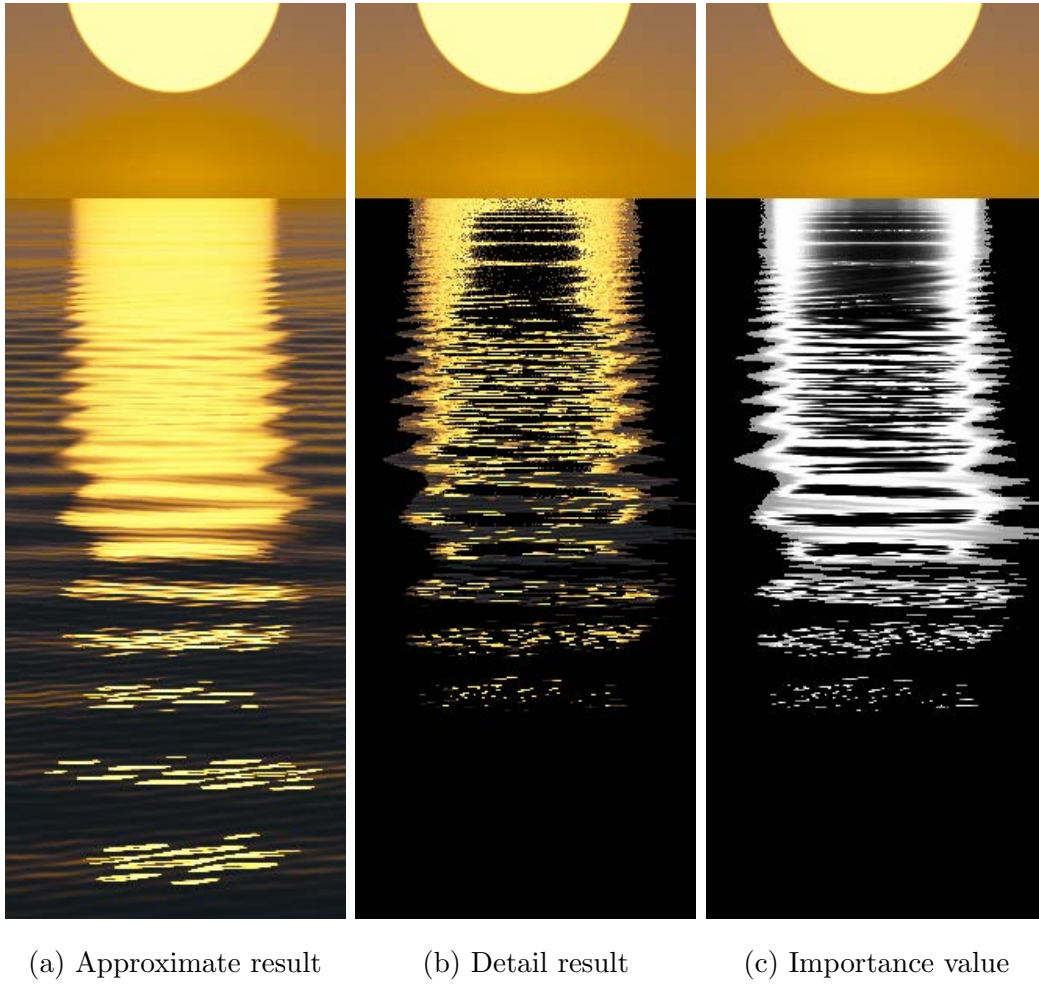


Figure 5.22: The internal result of our method.

rendering beautiful reflections on a water surface in real-time.

5.6 Real-time Micro-specular-highlight Rendering

We introduced another method to compute a better quality of a reflection that close to a camera. We didn't consider a pixel footprint of a simple reflection near a camera; thus, it lacked some quality. This method uses four-pixel corners to determine a quadrilateral pixel footprint. Then the method finds a sharp reflection result from the footprint. The method supports geometric shapes with and without a normal map.

A specular highlight or a specular reflection shows an interaction between a surface and a light source, which renders a characteristic of the material of the surface. Thus, specular highlights are crucial for the quality of a rendering result. However, the conventional method still has trouble with rendering small or complicated sharp specular highlights in real-time.

A small specular highlight is created by a small light source and a mirror-like surface. The problem arises when the highlight itself is too tiny to be captured consistently. The conventional rasterization method usually samples once per pixel at the center, and if the highlight is smaller than the pixel, then the process cannot consistently find the highlight. This inconsistency creates an aliasing in the final result.

We can increase the number of samples per pixel to remove the aliasing, but it is computationally expensive; it requires a large number of samples, particularly for mirror-like surfaces. Another solution is to find the contribution from the whole surface within the pixel by approximating the surface's normal variation as a normal distribution function, which is usually represented by a smooth function. However, this approach usually results in a blurred highlight; it cannot render the distinct appearance of the surface with a sharp specular highlight.

The use of normal mapping and deferred shading also escalates the problem. Normal mapping adds more detail and complexity to the surface, while deferred shading expands the number of possible light sources, which potentially introduces more aliasing.

We propose a method that analytically chooses a reflection vector that produces the most intense reflection within a pixel for each light source. Then our method weights the result with the ratio of the light source size within the pixel reflection

space. Our method produces in real-time a consistent specular reflection without aliasing.

Our main contributions are:

- An approximation of a reflection vector variation within a pixel as a linear combination of four reflection vectors at the four corners of the pixel.
- An analytical method for finding the reflection vector producing the strongest reflected intensity within a pixel.
- A weighting of the lighting of the strongest reflection vector.

5.6.1 Proposed Method

Our main objective is to render a highlight of a light source that is smaller than a pixel by considering the whole perfect specular surface within a pixel to capture the small highlight. We will introduce a brief overview of our method first, then move on to the detail of each component of the method.

The basic idea is to find the sample position that produces the brightest highlight instead of the center within a pixel to compute the intensity. Then we multiply the intensity with the weight representing a fraction of the light source area computed with the pixel reflection space. These two components are called *reflection vector selection* and *result weighting*. Figure 5.23 shows the algorithm of our method.

Reflection Vector Selection

A fraction of the surface within a pixel is, in general, a curved surface patch. Our method first samples reflection vectors at the corners of the pixel, as shown in Fig. 5.24. We assume that a reflection vector within the pixel can be represented as a linear combination of these four vectors. Let us consider a light plane defined as a plane located at $p + l$ with a normal vector of l , where p is the surface center, and l is the unit vector toward the light direction. Then, we compute a quadrilateral by projecting the four reflection vectors onto the light plane as shown in Fig. 5.25. Note that we assume the reflection vectors originate from the center of the surface. This quadrilateral defines the *reflection space*, that is, any reflection vectors within the

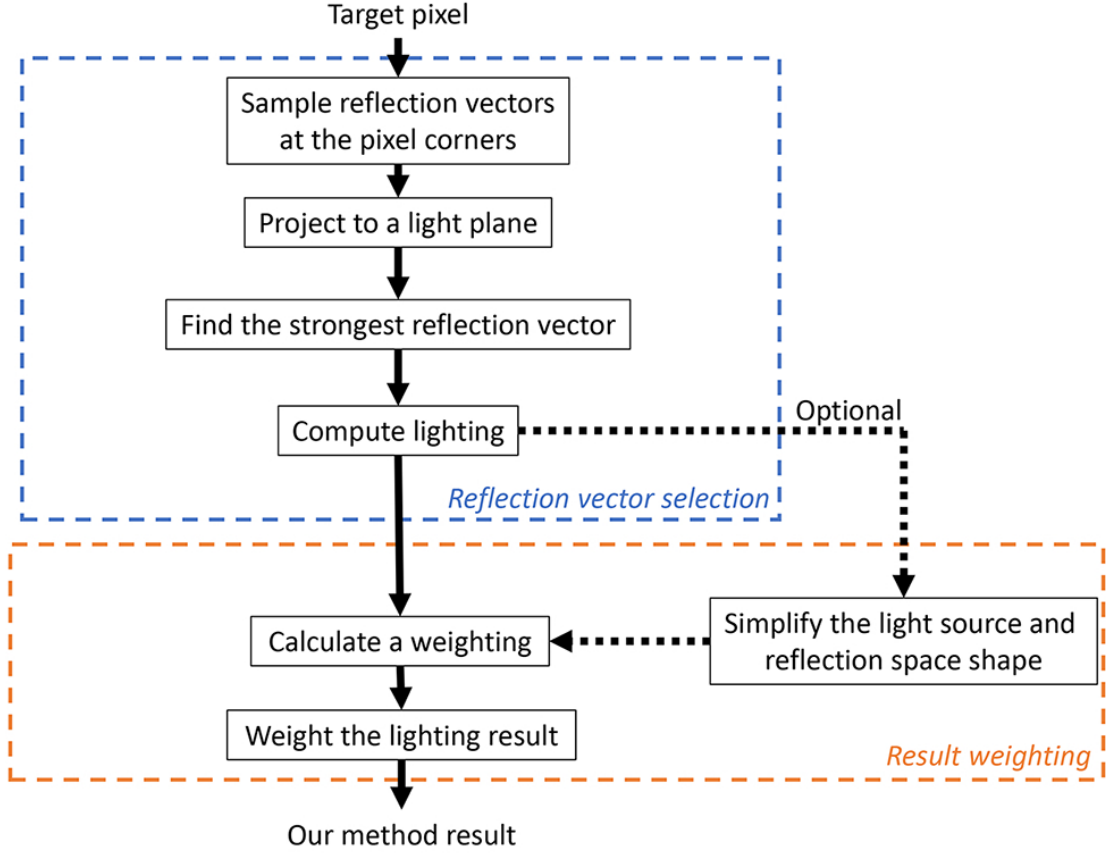


Figure 5.23: An overview of our method.

pixel are projected within this quadrilateral. We assume a reflection vector within a pixel is obtained by bilinear interpolation of the four vectors.

We then compute the most intense reflection vector by finding the point inside the reflection space closest to the center of the projected light source with this simple assumption.

The closest point is analytically determined by using geometric analysis given the shape of the light source. Currently, our implementation supports only a disk light source. The closest point to the disk light source can be solved by checking the projected position of the center of the light source. If the center is in the reflection space, the closest point is the center. Otherwise, the closest point is the point on the reflection space edge with the closest distance to the center.

Then we use the direction from the surface center to the closest point as the most intense reflection vector to compute the lighting. The result is the brightest highlight of any point within the pixel.

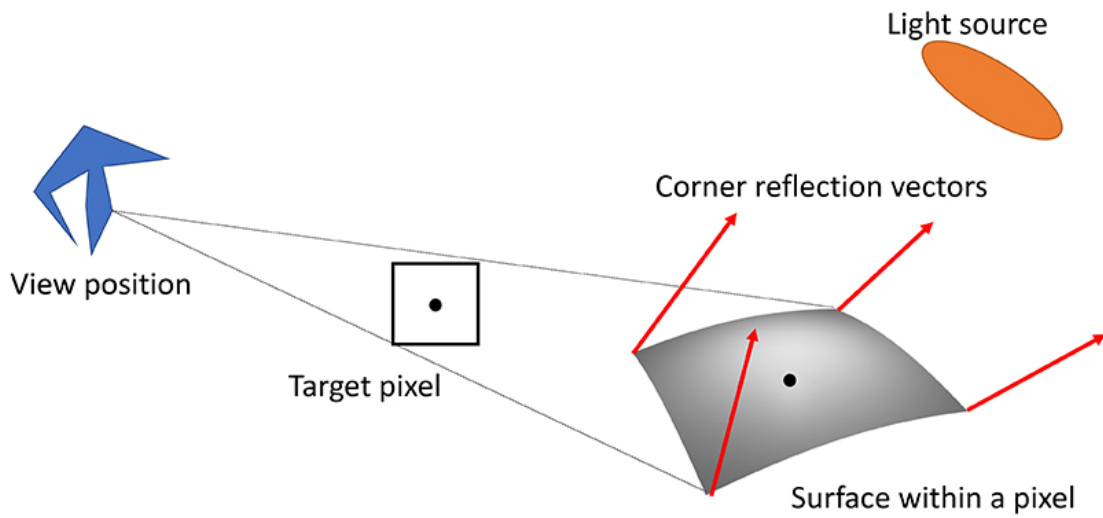


Figure 5.24: A curved surface patch inside a pixel and four samples of reflection vectors at the corners.

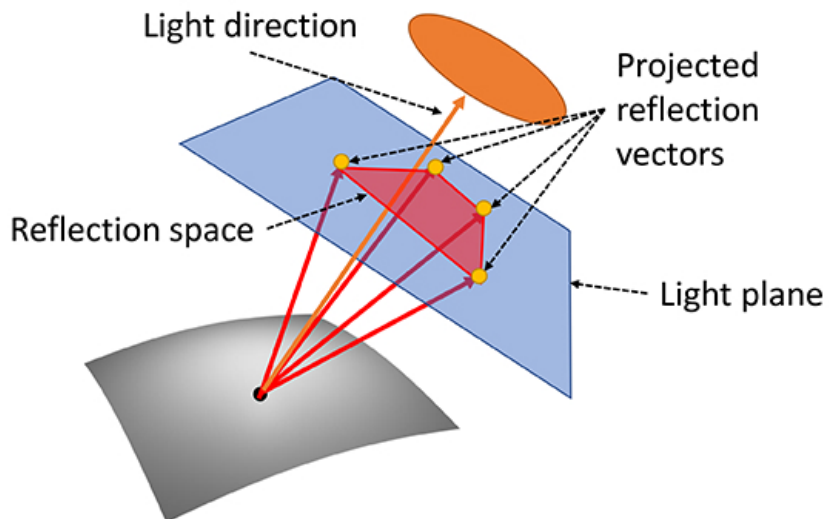


Figure 5.25: Definitions of the light plane and the reflection space.

Result Weighting

To compute the correct reflection intensity for a pixel, we need to compute an integral of the reflected intensity over a surface within the pixel. However, our method has only a single sample of the brightest reflection. We approximate the intensity with the product of the brightest reflection with the fraction of the projected area of the light source within the reflection space. This approximation is valid when the surface exhibits mirror-like reflections and is illuminated by a uniformly bright light source.

We can calculate the fraction of the surface area in the reflection space on the light plane due to our assumption that the reflection of the surface is bilinearly interpolated and an additional assumption that the Jacobian determinant of the bilinear mapping is constant (which is not always the case). Then area on the pixel is proportional to the area on the reflection space. So we only need to find the area of the intersection between the projected light source and the reflection space and the area of the reflection space. Both areas are solved by using geometric analysis on the 2D light plane.

However, the intersection calculation might be too expensive computationally. Hence we further approximate the areas of the projected light source and/or the shape of the reflection space. We are currently using two approximation types: we approximate the projected light source by a circle and the reflection space with a quad.

5.6.2 Results

We implemented our method using OpenGL on a laptop with an Intel Core i7 @ 2.50Ghz, 16 GB Memory, and an NVIDIA GeForce GTX 860M. We experiment with a disk light source in two scenes: ocean and ring. Ocean scene uses height map to tessellate ocean surface geometry and normal map to calculate lighting. Rings scene only has geometry without any normal map.

We compare our method against the reference method, the conventional method, LEAN mapping, NDF filtering, and our method with LEAN mapping. The reference method is the tiling 64x supersampling method, which separates a target result into multiple tiles and renders each of them sequentially. The conventional method is

the rendering with a single reflection sample per pixel. LEAN mapping is used in the ocean scene, while NDF filtering is used in the rings scene. We also combine our method and LEAN mapping in the ocean scene to minimize the error.

Figs. 5.26 and 5.27 shows the comparison in the ocean and the rings scene respectively. Table 5.3 shows the root mean square error (RMSE) of every experiment methods in each scene.

	Scenes	
Method	Ocean	Rings
Conventional	0.0365	0.0294
LEAN Mapping	0.0338	-
NDF Filtering	-	0.0365
Our	0.0344	0.0306
Our and LEAN	0.0316	-

Table 5.3: RMSE of the result of each method when compared to the references.

The ocean scene shows that our method excels when the reflections are close to the camera due to its low normal map resolution, while LEAN mapping failed to process because there are no variance data at the base level of the LEAN map. The normal map has a higher resolution when being far from the camera and causes our method to miscalculate the reflection. The cause is that we only use the pixel corner reflections to approximate the reflection space. But when the normal map texel is smaller than the pixel, the reflection vectors inside the pixel are no longer interpolated by the pixel corner reflections. To solve this problem, we use the LEAN mapping method when the normal map texel is smaller than the pixel. This combination method shows an improvement over the other methods.

The rings scene shows that our method can capture more reflection than the conventional method while retaining the sharp feature of the reflection similar to the reference method.



(a) Reference image (1083 ms) (b) Conventional (22 ms)



(c) LEAN mapping (24 ms) (d) Our method (24 ms)



(e) Our method with
LEAN mapping (25 ms)

Figure 5.26: The comparison result in the ocean scene.

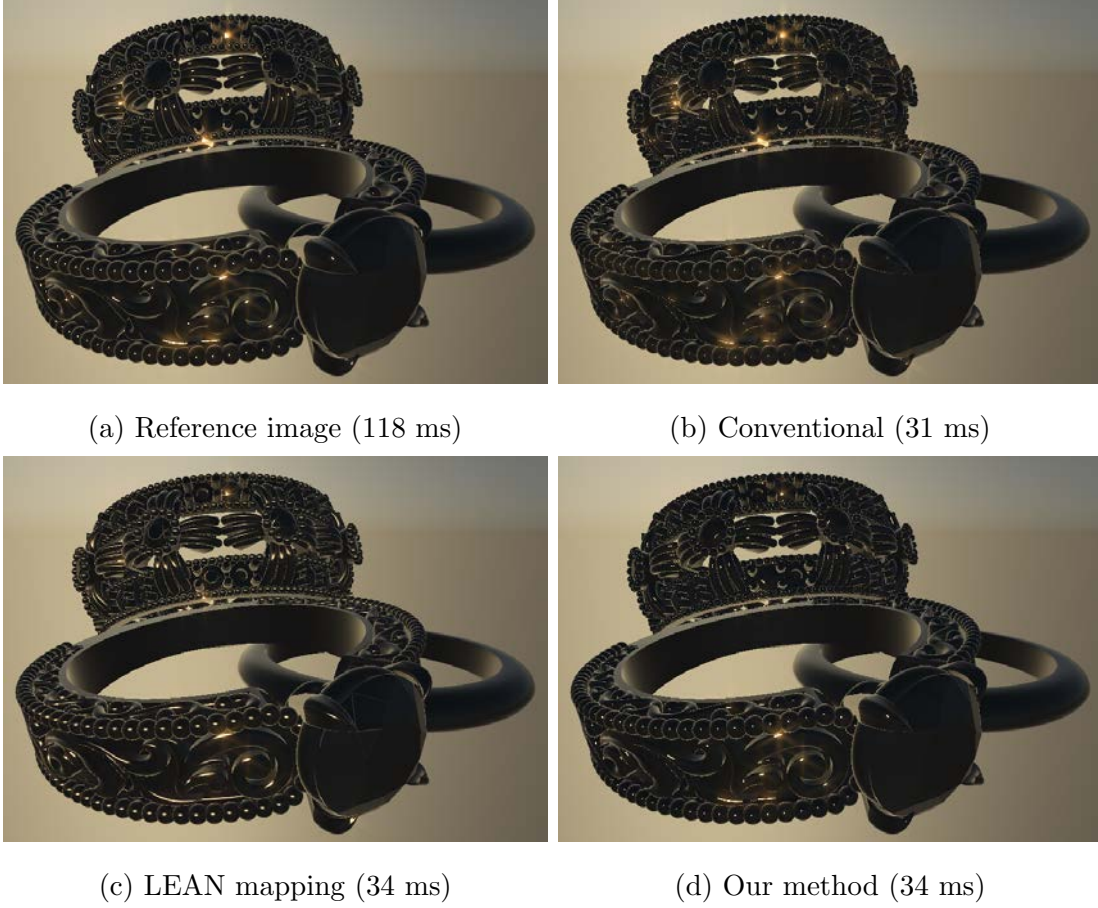


Figure 5.27: The comparison result in the rings scene.

5.6.3 Discussion

Our method and our combination with LEAN mapping show an improvement in both the quality and stability of specular highlights over the other methods. LEAN mapping and NDF filtering overly blur the reflection. However, our method fails to capture the reflection at far distances because there is high normal variation within a pixel, so our curved surface approximation with four reflection vectors is not accurate enough.

Our method can consistently capture a small sharp highlight on a detailed reflective surface without aliasing. However, it doesn't work well when there is a lot of detail inside one pixel. But we combine our method with LEAN mapping to relieve the problem. However, we still need to conduct experiments with different light sources and surface types. We also need to compare our method with the other state-of-the-art techniques, which are temporal anti-aliasing, LEADR mapping, and frequency domain normal mapping.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Our study tackles the problem of complex surface reflection rendering in real-time by improving the quality of rasterization based method. We proposed multiple methods that improve upon each other and suit for reflections in different scenarios. There are three main methods and one work in progress method: glittery surface rendering, spatio-temporal reflection anti-aliasing, adaptive supersampling reflection rendering, and micro-specular-highlight rendering respectively. Glittery surface rendering method is an work in progress method because we didn't fully evaluate the effectiveness of the method yet.

Glittery surface rendering creates sparkles look of glittery surface. Our method stores each orientation of normal map texel or microfacet in a bit field of one variable, which we called a flake data. Thus, we could efficient query a multiple normal direction of each pixel by precomputing a flake map that contains flake data of a surface with mipmapping. Our method can render a glittery surface in real-time and its appearance can be controlled with a simple normal map. However, we didn't evaluate our method with a reference image to clearly numerically shows its effectiveness yet.

Spatio-temporal reflection anti-aliasing computes an integral of reflection in both temporal and spatial space to create aliasing free reflection on a moving water surface. Our method is efficient and effective as an anti-aliasing solution as we expected. The method analytically solves the aliasing problem and thus it does

not have limitations like the temporal reprojection method [53, 64] that suffers from hidden-surface and reflection tracking problem. Our real-time method's limitation is that our method cannot replicate the small detail of the distant reflection. This is because we approximate the NDF and the RDF as a Gaussian distribution. The problem become more apparent when we use a uniform NDF, which calculate a single average NDF for the entire ocean wave. The problem is reduced when we uses per-pixel NDF.

Adaptive supersampling reflection rendering renders a reflection with traditional result, then evaluate the reflection result on each pixel. The evaluation put high priority to the pixels at the edge of reflection, which have high potential to contain high detail reflections. Then our method adaptively supersampling these pixels according to its potential in an efficient parallel computation. Our method show a significantly improvement over our previous method. The method can solve the blur reflection due to wave clamping or spatial anti-aliasing in our main method in real-time. However, the method has multiple manual parameters, which need to be adjusted for each scene.

Micro-specular-highlight rendering samples normal directions at the pixel corners to create a normal direction space of each pixel. The method then choose the brightest normal direction from the space and use it to calculate reflection that was difficult to find with a naive reflection rendering method. Our method create more stable and cleaner reflections than the traditional approach, while using low computational time. The method can capture sharp reflections on the low-resolution surfaces. But our method failed on the high-resolution surfaces because our assumption failed when the normal direction data are smaller than a pixel. We cannot use normal directions at each pixel corner to create an accurate normal space anymore.

All of our methods help improve the traditional rendering method to get closer to the accurate rendering of complex surfaces in real-time. Unfortunately, the quality of these result are not high enough as we expected yet, thus we will continue our work toward the goal.

6.2 Future Work

We scheduled our future work to continue to tackle the complex surface reflection rendering problem in each of our approaches.

Glittery surface rendering still lack the mean to evaluate its effectiveness and has a lot of room to improve its efficiency. We plan to create a reference image by using supersampling method and improve its rendering time by compute lighting only on some important flake directions instead of every flakes like in the current implementation.

Spatio-temporal reflection anti-aliasing can also be significantly improved. We plan to experiment and extend its capability as follows: multiple frame temporal anti-aliasing, efficient multiple light sources, and other types of light sources and waves will be considered. Multiple frame temporal anti-aliasing will simulate the effect of longer persistence of vision in a low light situation and help capture more reflection. Our method can efficiently process multiple light sources by only choosing and processing a small number of light sources that are important to each pixel instead of computing every light source. Other types of waves and light sources are needed to broaden the applicable scenes of our method.

Adaptive supersampling reflection rendering is relying on Gaussian function to determine the sampling density of each pixel. We can further improve its efficiency by using the information from the previous frames to determine the sampling density of each pixel. If the sampling result in the previous frame makes a large difference then we can guess that the pixel at the same location should have the sampling density relative to the previous difference. This method also remove the filter parameters, which we currently manually adjusted, as well.

Micro-specular-highlight rendering can use a better approximation by transforming the light polygon to the screen and compute the light contribution instead of projecting the light polygon to the light plane. Projecting to the light plane assume that the surface reflections are bilinearly interpolated uniformly in the reflection space on the light plane. The assumption is a large source of error of our current methodology. We also plan to include other type of materials and light sources to create a higher quality result.

There are still many topics that we can do to extend our current method capa-

bility and quality. We will continue to study about the complex surfaces rendering to realize the goal of an accurate rendering of complex surfaces in real-time.

Bibliography

- [1] Michael Abrash. “Quake’s lighting model: Surface caching”. In: *Graphic programming black book* (2000).
- [2] Dietger van Antwerpen. “Improving SIMD Efficiency for Parallel Monte Carlo Light Transport on the GPU”. In: *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*. HPG ’11. Vancouver, British Columbia, Canada: ACM, 2011, pp. 41–50. ISBN: 978-1-4503-0896-0. DOI: 10.1145/2018323.2018330. URL: <http://doi.acm.org/10.1145/2018323.2018330>.
- [3] Arthur Appel. “Some Techniques for Shading Machine Renderings of Solids”. In: *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*. AFIPS ’68 (Spring). Atlantic City, New Jersey: Association for Computing Machinery, 1968, pp. 37–45. ISBN: 9781450378970. DOI: 10.1145/1468075.1468082. URL: <https://doi.org/10.1145/1468075.1468082>.
- [4] Rasmus Barringer and Tomas Akenine-Möller. “A4: Asynchronous Adaptive Anti-Aliasing Using Shared Memory”. In: *ACM Trans. Graph.* 32.4 (July 2013). ISSN: 0730-0301. DOI: 10.1145/2461912.2462015. URL: <https://doi.org/10.1145/2461912.2462015>.
- [5] Louis Bavoil and Miguel Sainz. “Screen space ambient occlusion”. In: *NVIDIA developer information: <http://developers.nvidia.com>* 6 (2008).
- [6] Barry G. Becker and Nelson L. Max. “Smooth Transitions Between Bump Rendering Algorithms”. In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’93. Anaheim, CA: ACM, 1993, pp. 183–190. ISBN: 0-89791-601-8. DOI: 10.1145/166117.166141. URL: <http://doi.acm.org/10.1145/166117.166141>.

- [7] Markus Billeter, Ola Olsson, and Ulf Assarsson. “Efficient Stream Compaction on Wide SIMD Many-core Architectures”. In: *Proceedings of the Conference on High Performance Graphics 2009*. HPG ’09. New Orleans, Louisiana: ACM, 2009, pp. 159–166. ISBN: 978-1-60558-603-8. DOI: 10.1145/1572769.1572795. URL: <http://doi.acm.org/10.1145/1572769.1572795>.
- [8] J. BLINN. “Me and my (fake) shadow”. In: *IEEE Computer Graphics and Applications* 8.1 (1988), pp. 82–86. URL: <https://ci.nii.ac.jp/naid/10010178810/en/>.
- [9] James F. Blinn. “Models of Light Reflection for Computer Synthesized Pictures”. In: *Proceedings of the 4th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’77. San Jose, California: Association for Computing Machinery, 1977, pp. 192–198. ISBN: 9781450373555. DOI: 10.1145/563858.563893. URL: <https://doi.org/10.1145/563858.563893>.
- [10] James F. Blinn. “Simulation of Wrinkled Surfaces”. In: *SIGGRAPH Comput. Graph.* 12.3 (Aug. 1978), pp. 286–292. ISSN: 0097-8930. DOI: 10.1145/965139.507101. URL: <https://doi.org/10.1145/965139.507101>.
- [11] Eric Bruneton, Fabrice Neyret, and Nicolas Holzschuch. “Real-time Realistic Ocean Lighting using Seamless Transitions from Geometry to BRDF”. In: *Computer Graphics Forum* 29.2 (May 2010). EUROGRAPHICS 2010 (full paper) - Session Rendering I, pp. 487–496. DOI: 10.1111/j.1467-8659.2009.01618.x. URL: <https://hal.inria.fr/inria-00443630>.
- [12] Brian Cabral, Marc Olano, and Philip Nemec. “Reflection Space Image Based Rendering”. In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’99. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 165–170. ISBN: 0-201-48560-5. DOI: 10.1145/311535.311553. URL: <http://dx.doi.org/10.1145/311535.311553>.
- [13] Edwin Earl Catmull. “A Subdivision Algorithm for Computer Display of Curved Surfaces.” AAI7504786. PhD thesis. 1974.
- [14] Wesley A. Clark. “The Lincoln TX-2 Computer Development”. In: *Papers Presented at the February 26-28, 1957, Western Joint Computer Conference: Techniques for Reliability*. IRE-AIEE-ACM ’57 (Western). Los Angeles, Cali-

- fornia: Association for Computing Machinery, 1957, pp. 143–145. ISBN: 9781450378611. DOI: 10.1145/1455567.1455592. URL: <https://doi.org/10.1145/1455567.1455592>.
- [15] R. L. Cook and K. E. Torrance. “A Reflectance Model for Computer Graphics”. In: *ACM Trans. Graph.* 1.1 (Jan. 1982), pp. 7–24. ISSN: 0730-0301. DOI: 10.1145/357290.357293. URL: <https://doi.org/10.1145/357290.357293>.
- [16] Robert L. Cook, Thomas Porter, and Loren Carpenter. “Distributed Ray Tracing”. In: *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’84. New York, NY, USA: ACM, 1984, pp. 137–145. ISBN: 0-89791-138-5. DOI: 10.1145/800031.808590. URL: <http://doi.acm.org/10.1145/800031.808590>.
- [17] Robert L. Cook and Kenneth E. Torrance. “A Reflectance Model for Computer Graphics”. In: *SIGGRAPH Comput. Graph.* 15.3 (Aug. 1981), pp. 307–316. ISSN: 0097-8930. DOI: 10.1145/965161.806819. URL: <https://doi.org/10.1145/965161.806819>.
- [18] Cyril Crassin et al. “Interactive indirect illumination using voxel cone tracing”. In: *Computer Graphics Forum*. Vol. 30. 7. Wiley Online Library. 2011, pp. 1921–1930.
- [19] Jonathan Dupuy, Eric Heitz, and Laurent Belcour. “A Spherical Cap Preserving Parameterization for Spherical Distributions”. In: *ACM Trans. Graph.* 36.4 (July 2017), 139:1–139:12. ISSN: 0730-0301. DOI: 10.1145/3072959.3073694. URL: <http://doi.acm.org/10.1145/3072959.3073694>.
- [20] Jonathan Dupuy et al. “Linear Efficient Antialiased Displacement and Reflectance Mapping”. In: *ACM Trans. Graph.* 32.6 (Nov. 2013), 211:1–211:11. ISSN: 0730-0301. DOI: 10.1145/2508363.2508422. URL: <http://doi.acm.org/10.1145/2508363.2508422>.
- [21] Eduardo S. L. Gastal and Manuel M. Oliveira. “Adaptive Manifolds for Real-time High-dimensional Filtering”. In: *ACM Trans. Graph.* 31.4 (July 2012), 33:1–33:13. ISSN: 0730-0301. DOI: 10.1145/2185520.2185529. URL: <http://doi.acm.org/10.1145/2185520.2185529>.

- [22] Carlos Gonzalez-Ochoa, Doug Holder, and Eben Cook. “From a Calm Puddle to a Stormy Ocean: Rendering Water in Uncharted”. In: *ACM SIGGRAPH 2012 Talks*. SIGGRAPH '12. Los Angeles, California: ACM, 2012, 3:1–3:1. ISBN: 978-1-4503-1683-5. DOI: 10.1145/2343045.2343050. URL: <http://doi.acm.org/10.1145/2343045.2343050>.
- [23] Cindy M. Goral et al. “Modeling the Interaction of Light between Diffuse Surfaces”. In: *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '84. New York, NY, USA: Association for Computing Machinery, 1984, pp. 213–222. ISBN: 0897911385. DOI: 10.1145/800031.808601. URL: <https://doi.org/10.1145/800031.808601>.
- [24] Henri Gouraud. “Computer Display of Curved Surfaces”. AAI7127878. PhD thesis. 1971.
- [25] Charles W. Grant. “Integrated Analytic Spatial and Temporal Anti-aliasing for Polyhedra in 4-space”. In: *SIGGRAPH Comput. Graph.* 19.3 (July 1985), pp. 79–84. ISSN: 0097-8930. DOI: 10.1145/325165.325184. URL: <http://doi.acm.org/10.1145/325165.325184>.
- [26] Charles Han et al. “Frequency Domain Normal Map Filtering”. In: *ACM SIGGRAPH 2007 Papers*. SIGGRAPH '07. San Diego, California: Association for Computing Machinery, 2007, 28–es. ISBN: 9781450378369. DOI: 10.1145/1275808.1276412. URL: <https://doi.org/10.1145/1275808.1276412>.
- [27] T. HEIDMANN. “Real shadows real time”. In: *Iris Universe* 18 (1989), pp. 28–31. URL: <https://ci.nii.ac.jp/naid/10011323325/en/>.
- [28] Eric Heitz et al. “Real-time Polygonal-light Shading with Linearly Transformed Cosines”. In: *ACM Trans. Graph.* 35.4 (July 2016), 41:1–41:8. ISSN: 0730-0301. DOI: 10.1145/2897824.2925895. URL: <http://doi.acm.org/10.1145/2897824.2925895>.
- [29] Matthias Holländer, Tamy Boubekeur, and Elmar Eisemann. “Adaptive Supersampling for Deferred Anti-Aliasing”. In: *Journal of Computer Graphics Techniques (JCGT)* 2.1 (Mar. 2013), pp. 1–14. ISSN: 2331-7418. URL: <http://jcgt.org/published/0002/01/01/>.

- [30] Rob Hopper and Kai Wolter. “The Water Effects of Pirates of the Caribbean: Dead Men Tell No Tales”. In: *ACM SIGGRAPH 2017 Talks*. SIGGRAPH ’17. Los Angeles, California: ACM, 2017, 31:1–31:2. ISBN: 978-1-4503-5008-2. DOI: 10.1145/3084363.3085044. URL: <http://doi.acm.org/10.1145/3084363.3085044>.
- [31] Wenzel Jakob et al. “Discrete Stochastic Microfacet Models”. In: *ACM Trans. Graph.* 33.4 (July 2014). ISSN: 0730-0301. DOI: 10.1145/2601097.2601186. URL: <https://doi.org/10.1145/2601097.2601186>.
- [32] Jorge Jimenez and Diego Gutierrez. *Screen-Space Subsurface Scattering*. 2010.
- [33] Jorge Jimenez, Veronica Sundstedt, and Diego Gutierrez. “Screen-space perceptual rendering of human skin”. In: *ACM Transactions on Applied Perception (TAP)* 6.4 (2009), pp. 1–15.
- [34] Bongjun Jin et al. “Selective and Adaptive Supersampling for Real-Time Ray Tracing”. In: *Proceedings of the Conference on High Performance Graphics 2009*. HPG ’09. New Orleans, Louisiana: Association for Computing Machinery, 2009, pp. 117–125. ISBN: 9781605586038. DOI: 10.1145/1572769.1572788. URL: <https://doi.org/10.1145/1572769.1572788>.
- [35] James T. Kajiya. “The Rendering Equation”. In: *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’86. New York, NY, USA: ACM, 1986, pp. 143–150. ISBN: 0-89791-196-2. DOI: 10.1145/15922.15902. URL: <http://doi.acm.org/10.1145/15922.15902>.
- [36] James T. Kajiya. “The Rendering Equation”. In: *SIGGRAPH Comput. Graph.* 20.4 (Aug. 1986), pp. 143–150. ISSN: 0097-8930. DOI: 10.1145/15886.15902. URL: <http://doi.acm.org/10.1145/15886.15902>.
- [37] A. S. Kaplanyan et al. “Filtering Distributions of Normals for Shading Antialiasing”. In: *Proceedings of High Performance Graphics*. HPG ’16. Dublin, Ireland: Eurographics Association, 2016, pp. 151–162. ISBN: 9783038680086.
- [38] Brian Karis and Epic Games. “Real shading in unreal engine 4”. In: *Proc. Physically Based Shading Theory Practice* 4 (2013), p. 3.

- [39] Jonathan Korein and Norman Badler. “Temporal Anti-aliasing in Computer Generated Animation”. In: *SIGGRAPH Comput. Graph.* 17.3 (July 1983), pp. 377–388. ISSN: 0097-8930. DOI: 10.1145/964967.801168. URL: <http://doi.acm.org/10.1145/964967.801168>.
- [40] Eric P. Lafortune and Yves D. Willems. “Bi-Directional Path Tracing”. In: *Proceedings of Compugraphics '93*. 1993, pp. 145–153.
- [41] Rynson W. H. Lau. “An Adaptive Supersampling Method”. In: *Proceedings of the Third International Computer Science Conference on Image Analysis Applications and Computer Graphics*. ICSC '95. Berlin, Heidelberg: Springer-Verlag, 1995, pp. 205–214. ISBN: 3540606971.
- [42] Won-Jong Lee et al. “Adaptive Multi-Rate Ray Sampling on Mobile Ray Tracing GPU”. In: *SIGGRAPH ASIA 2016 Mobile Graphics and Interactive Applications*. SA '16. Macau: Association for Computing Machinery, 2016. ISBN: 9781450345514. DOI: 10.1145/2999508.2999523. URL: <https://doi.org/10.1145/2999508.2999523>.
- [43] Norman H. Lehrer. “The Challenge of the Cathode-Ray Tube”. In: *Flat-Panel Displays and CRTs*. Dordrecht: Springer Netherlands, 1985, pp. 138–176. ISBN: 978-94-011-7062-8. DOI: 10.1007/978-94-011-7062-8_6. URL: https://doi.org/10.1007/978-94-011-7062-8_6.
- [44] Timothy Lottes. *FXAA*. https://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf. [Online; accessed 29-March-2019]. 2009.
- [45] Guillaume Loubet et al. “Slope-Space Integrals for Specular next Event Estimation”. In: *ACM Trans. Graph.* 39.6 (Nov. 2020). ISSN: 0730-0301. DOI: 10.1145/3414685.3417811. URL: <https://doi.org/10.1145/3414685.3417811>.
- [46] Adam Marrs et al. “Adaptive Temporal Antialiasing”. In: *Proceedings of the Conference on High-Performance Graphics*. HPG '18. Vancouver, British Columbia, Canada: Association for Computing Machinery, 2018. ISBN: 9781450358965. DOI: 10.1145/3231578.3231579. URL: <https://doi.org/10.1145/3231578.3231579>.

- [47] Joel McCormack et al. “Feline: Fast Elliptical Lines for Anisotropic Texture Mapping”. In: *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '99. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1999, pp. 243–250. ISBN: 0-201-48560-5. DOI: 10.1145/311535.311562. URL: <http://dx.doi.org/10.1145/311535.311562>.
- [48] T. McReynolds and D. Blythe. *Advanced Graphics Programming Using OpenGL*. The Morgan Kaufmann Series in Computer Graphics. Elsevier Science, 2014. ISBN: 9780123814999. URL: <https://books.google.co.jp/books?id=kbFZAwwAAQBAJ>.
- [49] Soham Uday Mehta et al. “Axis-aligned Filtering for Interactive Physically-based Diffuse Indirect Lighting”. In: *ACM Trans. Graph.* 32.4 (July 2013), 96:1–96:12. ISSN: 0730-0301. DOI: 10.1145/2461912.2461947. URL: <http://doi.acm.org/10.1145/2461912.2461947>.
- [50] Morten Mikkelsen. *Simulation of Wrinkled Surfaces Revisited*. 2008.
- [51] Jason Mitchell, Gary McTaggart, and Chris Green. “Shading in valve’s source engine”. In: *Acm siggraph 2006 courses*. 2006, pp. 129–142.
- [52] Gabor Nagy. “Real-Time Shadows On Complex Objects”. In: (Jan. 2000).
- [53] Diego Nehab, Pedro V. Sander, and John R. Isidoro. “The Real-time Reprojection Cache”. In: *ACM SIGGRAPH 2006 Sketches*. SIGGRAPH '06. Boston, Massachusetts: ACM, 2006. ISBN: 1-59593-364-6. DOI: 10.1145/1179849.1180080. URL: <http://doi.acm.org/10.1145/1179849.1180080>.
- [54] Alan Norton, Alyn P. Rockwood, and Philip T. Skolmoski. “Clamping: A Method of Antialiasing Textured Surfaces by Bandwidth Limiting in Object Space”. In: *Proceedings of the 9th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '82. Boston, Massachusetts, USA: ACM, 1982, pp. 1–8. ISBN: 0-89791-076-1. DOI: 10.1145/800064.801252. URL: <http://doi.acm.org/10.1145/800064.801252>.
- [55] Jan Novák, Vlastimil Havran, and Carsten Dachsbacher. “Path Regeneration for Interactive Path Tracing”. In: *Eurographics 2010 - Short Papers*. Ed. by H. P. A. Lensch and S. Seipel. The Eurographics Association, 2010. DOI: 10.2312/egsh.20101048.

- [56] Marc Olano and Dan Baker. “LEAN Mapping”. In: *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D ’10. Washington, D.C.: ACM, 2010, pp. 181–188. ISBN: 978-1-60558-939-8. DOI: 10.1145/1730804.1730834. URL: <http://doi.acm.org/10.1145/1730804.1730834>.
- [57] Namu Podede et al. “Temporal and Spatial Anti-Aliasing for Rendering Reflection on a Water Surface”. In: *ACM SIGGRAPH 2019 Posters*. SIGGRAPH 2019. Los Angeles, California: Association for Computing Machinery, 2019. ISBN: 9781450363143. DOI: 10.1145/3306214.3338599. URL: <https://doi.org/10.1145/3306214.3338599>.
- [58] Ramesh Raskar and Michael Cohen. “Image Precision Silhouette Edges”. In: *Proceedings of the 1999 Symposium on Interactive 3D Graphics*. I3D ’99. Atlanta, Georgia, USA: Association for Computing Machinery, 1999, pp. 135–140. ISBN: 1581130821. DOI: 10.1145/300523.300539. URL: <https://doi.org/10.1145/300523.300539>.
- [59] Vincent Ross, Denis Dion, and Guy Potvin. “Detailed analytical approach to the Gaussian surface bidirectional reflectance distribution function specular component applied to the sea surface”. In: *J. Opt. Soc. Am. A* 22.11 (Nov. 2005), pp. 2442–2453. DOI: 10.1364/JOSAA.22.002442. URL: <http://josaa.osa.org/abstract.cfm?URI=josaa-22-11-2442>.
- [60] Scott D Roth. “Ray casting for modeling solids”. In: *Computer Graphics and Image Processing* 18.2 (1982), pp. 109–144. ISSN: 0146-664X. DOI: [http://dx.doi.org/10.1016/0146-664X\(82\)90169-1](http://dx.doi.org/10.1016/0146-664X(82)90169-1). URL: <http://www.sciencedirect.com/science/article/pii/0146664X82901691>.
- [61] Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. “Adaptive Rendering with Non-local Means Filtering”. In: *ACM Trans. Graph.* 31.6 (Nov. 2012), 195:1–195:11. ISSN: 0730-0301. DOI: 10.1145/2366145.2366214. URL: <http://doi.acm.org/10.1145/2366145.2366214>.
- [62] Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. “Adaptive Sampling and Reconstruction Using Greedy Error Minimization”. In: *ACM Trans.*

- Graph.* 30.6 (Dec. 2011), 159:1–159:12. ISSN: 0730-0301. DOI: 10.1145/2070781.2024193. URL: <http://doi.acm.org/10.1145/2070781.2024193>.
- [63] Fabrice Rousselle, Marco Manzi, and Matthias Zwicker. “Robust Denoising using Feature and Color Information”. In: *Computer Graphics Forum* 32.7 (2013), pp. 121–130. ISSN: 1467-8659. DOI: 10.1111/cgf.12219. URL: <http://dx.doi.org/10.1111/cgf.12219>.
 - [64] Daniel Scherzer, Stefan Jeschke, and Michael Wimmer. “Pixel-correct Shadow Maps with Temporal Reprojection and Shadow Test Confidence”. In: *Proceedings of the 18th Eurographics Conference on Rendering Techniques*. EGSR’07. Grenoble, France: Eurographics Association, 2007, pp. 45–50. ISBN: 978-3-905673-52-4. DOI: 10.2312/EGWR/EGSR07/045-050. URL: <http://dx.doi.org/10.2312/EGWR/EGSR07/045-050>.
 - [65] C. E. Shannon. “Communication in the Presence of Noise”. In: *Proceedings of the IRE* 37.1 (1949), pp. 10–21.
 - [66] Mikio Shinya. “Spatial Anti-aliasing for Animation Sequences with Spatio-temporal Filtering”. In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’93. Anaheim, CA: ACM, 1993, pp. 289–296. ISBN: 0-89791-601-8. DOI: 10.1145/166117.166154. URL: <http://doi.acm.org/10.1145/166117.166154>.
 - [67] B. Smith. “Geometrical shadowing of a random rough surface”. In: *IEEE Transactions on Antennas and Propagation* 15.5 (Sept. 1967), pp. 668–671. ISSN: 0018-926X. DOI: 10.1109/TAP.1967.1138991.
 - [68] Springer Verlag GmbH, European Mathematical Society. *Encyclopedia of Mathematics*. Website. URL: <https://www.encyclopediaofmath.org/>. Accessed on 2016-10-11.
 - [69] W. Strasser. *Schnelle Kurven- und Flächendarstellung auf grafischen Sichtgeräten*. Heinrich-Hertz-Institut für Nachrichtentechnik Berlin, West: Technischer Bericht. Technische Universität Berlin, 1974. URL: <https://books.google.co.jp/books?id=biStNwAACAAJ>.

- [70] Ivan E. Sutherland. “A Head-Mounted Three Dimensional Display”. In: *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I. AFIPS ’68 (Fall, part I)*. San Francisco, California: Association for Computing Machinery, 1968, pp. 757–764. ISBN: 9781450378994. DOI: 10.1145/1476589.1476686. URL: <https://doi.org/10.1145/1476589.1476686>.
- [71] Natasha Tatarchuk et al. “Advances in Real-Time Rendering in Games, Part I (Full Text Not Available)”. In: *ACM SIGGRAPH 2014 Courses*. SIGGRAPH ’14. Vancouver, Canada: Association for Computing Machinery, 2014. ISBN: 9781450329620. DOI: 10.1145/2614028.2615455. URL: <https://doi.org/10.1145/2614028.2615455>.
- [72] Yusuke Tokuyoshi and Anton S. Kaplanyan. “Improved Geometric Specular Antialiasing”. In: *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. I3D ’19*. Montreal, Quebec, Canada: Association for Computing Machinery, 2019. ISBN: 9781450363105. DOI: 10.1145/3306131.3317026. URL: <https://doi.org/10.1145/3306131.3317026>.
- [73] Yasin Uludag. “Hi-Z screen-space cone-traced reflections”. In: *GPU Pro 5* (2014), pp. 149–192.
- [74] Eric Veach. “Robust Monte Carlo Methods for Light Transport Simulation”. AAI9837162. PhD thesis. Stanford, CA, USA, 1998. ISBN: 0591907801.
- [75] Ingo Wald. “Active Thread Compaction for GPU Path Tracing”. In: *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics. HPG ’11*. Vancouver, British Columbia, Canada: ACM, 2011, pp. 51–58. ISBN: 978-1-4503-0896-0. DOI: 10.1145/2018323.2018331. URL: <http://doi.acm.org/10.1145/2018323.2018331>.
- [76] Ingo Wald, Philipp Slusallek, and Carsten Benthin. “Interactive Distributed Ray Tracing of Highly Complex Models”. In: *Rendering Techniques 2001*. Ed. by Steven J. Gortler and Karol Myszkowski. Vienna: Springer Vienna, 2001, pp. 277–288. ISBN: 978-3-7091-6242-2.
- [77] John R. Wallace, Michael F. Cohen, and Donald P. Greenberg. “A Two-Pass Solution to the Rendering Equation: A Synthesis of Ray Tracing and Radiosity Methods”. In: *Proceedings of the 14th Annual Conference on Computer*

- Graphics and Interactive Techniques*. SIGGRAPH '87. New York, NY, USA: Association for Computing Machinery, 1987, pp. 311–320. ISBN: 0897912276. DOI: 10.1145/37401.37438. URL: <https://doi.org/10.1145/37401.37438>.
- [78] John R. Wallace, Michael F. Cohen, and Donald P. Greenberg. “A Two-Pass Solution to the Rendering Equation: A Synthesis of Ray Tracing and Radiosity Methods”. In: *SIGGRAPH Comput. Graph.* 21.4 (Aug. 1987), pp. 311–320. ISSN: 0097-8930. DOI: 10.1145/37402.37438. URL: <https://doi.org/10.1145/37402.37438>.
- [79] Turner Whitted. “An Improved Illumination Model for Shaded Display”. In: *Commun. ACM* 23.6 (June 1980), pp. 343–349. ISSN: 0001-0782. DOI: 10.1145/358876.358882. URL: <http://doi.acm.org/10.1145/358876.358882>.
- [80] Lance Williams. “Casting Curved Shadows on Curved Surfaces”. In: *SIGGRAPH Comput. Graph.* 12.3 (Aug. 1978), pp. 270–274. ISSN: 0097-8930. DOI: 10.1145/965139.807402. URL: <https://doi.org/10.1145/965139.807402>.
- [81] Lifan Wu et al. “Accurate Appearance Preserving Prefiltering for Rendering Displacement-mapped Surfaces”. In: *ACM Trans. Graph.* 38.4 (July 2019), 137:1–137:14. ISSN: 0730-0301. DOI: 10.1145/3306346.3322936. URL: <http://doi.acm.org/10.1145/3306346.3322936>.
- [82] Chris Wylie et al. “Half-tone Perspective Drawings by Computer”. In: *Proceedings of the November 14-16, 1967, Fall Joint Computer Conference*. AFIPS '67 (Fall). Anaheim, California: ACM, 1967, pp. 49–58. DOI: 10.1145/1465611.1465619. URL: <http://doi.acm.org/10.1145/1465611.1465619>.
- [83] Ling-Qi Yan et al. “Position-Normal Distributions for Efficient Rendering of Specular Microstructure”. In: *ACM Trans. Graph.* 35.4 (July 2016). ISSN: 0730-0301. DOI: 10.1145/2897824.2925915. URL: <https://doi.org/10.1145/2897824.2925915>.
- [84] Ling-Qi Yan et al. “Rendering Glints on High-Resolution Normal-Mapped Specular Surfaces”. In: *ACM Trans. Graph.* 33.4 (July 2014). ISSN: 0730-0301. DOI: 10.1145/2601097.2601155. URL: <https://doi.org/10.1145/2601097.2601155>.

- [85] Lei Yang, Shiqiu Liu, and Marco Salvi. “A Survey of Temporal Antialiasing Techniques”. In: *Computer Graphics Forum* 39.2 (2020), pp. 607–621. DOI: <https://doi.org/10.1111/cgf.14018>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14018>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14018>.
- [86] Tizian Zeltner, Iliyan Georgiev, and Wenzel Jakob. “Specular Manifold Sampling for Rendering High-Frequency Caustics and Glints”. In: *ACM Trans. Graph.* 39.4 (July 2020). ISSN: 0730-0301. DOI: 10.1145/3386569.3392408. URL: <https://doi.org/10.1145/3386569.3392408>.
- [87] Tobias Zirr and Anton S. Kaplanyan. “Real-Time Rendering of Procedural Multiscale Materials”. In: *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D ’16. Redmond, Washington: Association for Computing Machinery, 2016, pp. 139–148. ISBN: 9781450340434. DOI: 10.1145/2856400.2856409. URL: <https://doi.org/10.1145/2856400.2856409>.
- [88] M. Zwicker et al. “Recent Advances in Adaptive Sampling and Reconstruction for Monte Carlo Rendering”. In: *Comput. Graph. Forum* 34.2 (May 2015), pp. 667–681. ISSN: 0167-7055. DOI: 10.1111/cgf.12592. URL: <http://dx.doi.org/10.1111/cgf.12592>.
- [89] Matthias Zwicker et al. “EWA Volume Splatting”. In: *Proceedings of the Conference on Visualization ’01*. VIS ’01. San Diego, California: IEEE Computer Society, 2001, pp. 29–36. ISBN: 0-7803-7200-X. URL: <http://dl.acm.org/citation.cfm?id=601671.601674>.

Appendix A

Real-time Spatio-temporal Reflection Anti-aliasing

A.1 Per-pixel NDF

To find p_1 and p_2 of each wave within the pixel, we project pixel square onto the wave plane, then find intersections of a line, which is at the center of the projected square with a direction of each wave direction, and the boundary of the projected square. $p(x, z)$ at the two intersections are p_1 and p_2 of the wave. Fig. A.1 illustrates our method.

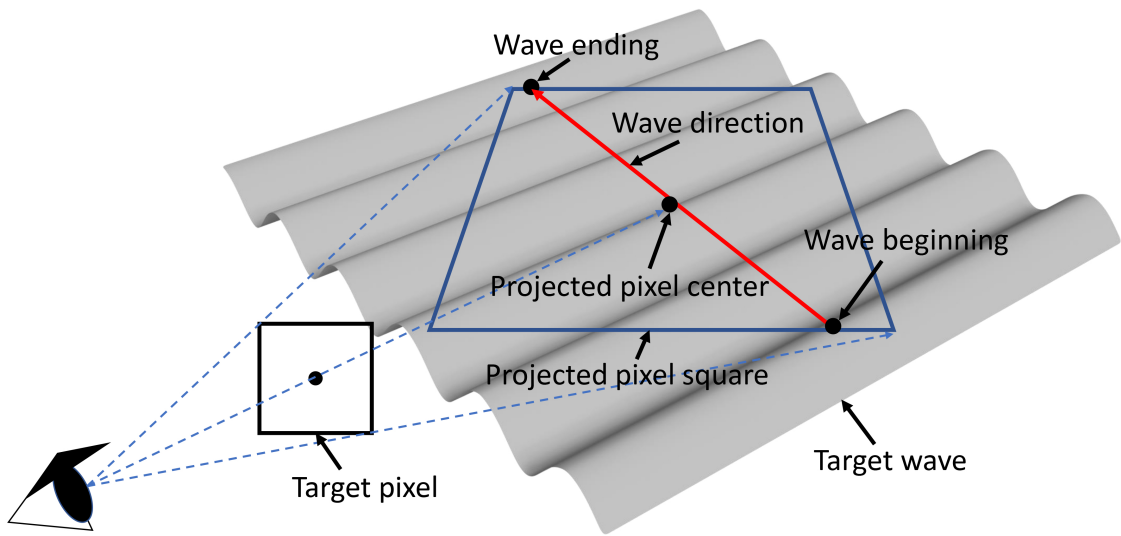


Figure A.1: Wave beginning and wave ending in the image is spatial position of p_1 and p_2 respectively.

The solution of Eq. (5.5) and (5.6) is as follows:

$$B = (B.x, B.z) = (a, b)K, \quad (\text{A.1})$$

$$M = (M.xx, M.xz, M.zz) = (a^2, ab, b^2)J, \quad (\text{A.2})$$

$$K = \frac{A(\sin(2(p_2 + ct)) - \sin(2(p_1 + ct)))}{p_2 - p_1}, \quad (\text{A.3})$$

$$J = \frac{A^2(\sin(2(p_2 + ct)) - \sin(2(p_1 + ct))) + 2(p_2 - p_1)}{4(p_2 - p_1)}, \quad (\text{A.4})$$

A.2 RDF approximation accuracy

We assume that the transformation from the NDF to RDF is linear, which is not true and causes an error. We analyze this error by comparing our approximate RDF with the accurate RDF computed by sampling NDF. Fig. A.2 shows the relative error between them. For each combination of X and Z variances of the NDF, we average the error in multiple view directions, while aligning the view direction to the middle of X and Y axis. The relative error becomes larger when the X variance is low and Z variance is high. The error also increases when both variances increase.

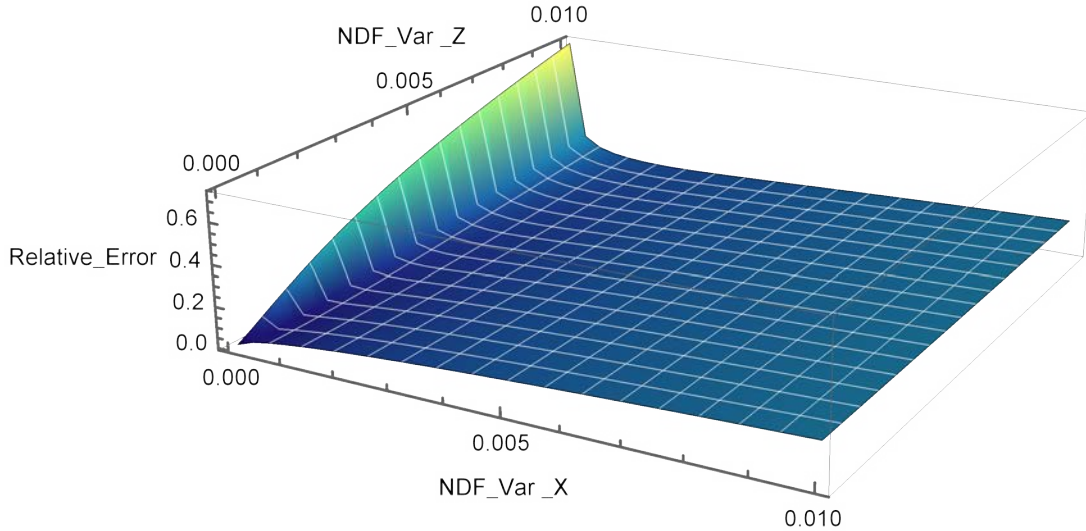


Figure A.2: The relative error of NDF to RDF transformation. Height and color shows the relative error while the other axis represent NDF variance in X and Z axis. Color from blue to green to yellow represents the relative error from low to high.

A.3 Elliptical Gaussian approximation

As in Fig. 5.9, we place one circular kernel at the center of the elliptical Gaussian kernel while the rests are placed in pairs along the major axis symmetrically. Thus the number of circular Gaussian kernels is $2N + 1$, where N is an integer. Then we have $3N + 2$ parameters: weight w_a and variance σ_a^2 of the center kernel, N weights w_i and variances σ_i^2 of the symmetrical kernel pairs, and the distances d_i from the center to each symmetrical kernel pair.

Unlike [47], which used somewhat arbitrary simple formulas, we determine the parameters by minimizing the integral of the squared difference between the elliptical Gaussian kernel and our approximate multiple circular Gaussian kernels. Without loss of generality, we assume that the elliptical Gaussian's minor and major axes are on the X- and Y-axis, respectively. Then the error function ϵ is:

$$\epsilon = \int \int \left(\mathcal{G}_{\Sigma(\sigma_x, \sigma_y), (0,0)}(x, y) - \mathcal{C}(x, y) \right)^2 dx dy, \quad (\text{A.5})$$

$$\begin{aligned} \mathcal{C}(x, y) &= w_a \mathcal{G}_{\Sigma(\sigma_a, \sigma_a), (0,0)}(x, y) \\ &+ \sum_{i=1}^N \left(w_i (\mathcal{G}_{\Sigma(\sigma_i, \sigma_i), (0, d_i)}(x, y) + \mathcal{G}_{\Sigma(\sigma_i, \sigma_i), (0, -d_i)}(x, y)) \right), \end{aligned} \quad (\text{A.6})$$

$$\Sigma(\sigma_x, \sigma_y) = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}, \quad (\text{A.7})$$

where $\mathcal{G}$ is a 2D Gaussian distribution function as in Eq. (5.2).

We can determine the optimal parameters by fixing σ_x to 1 since all the other parameters can be scaled with σ_x . We first used Mathematica to obtain an analytic formula for the double integral in Eq. (A.5) as a function of σ_y and the $3N + 2$ parameters, for each of $N = 1, 2$, and 4. Here is an example for $N = 1$, with three circular Gaussian kernels:

$$\epsilon = \frac{\frac{w_a^2}{\sigma_a^2} + \frac{8w_a w_1 e^{-\frac{d_1^2}{2(\sigma_a^2 + \sigma_1^2)}}}{\sigma_a^2 + \sigma_1^2} - \frac{4w_a}{\sqrt{(\sigma_a^2 + 1)(\sigma_a^2 + \sigma_y^2)}}}{4\pi} + \frac{2b^2 \left(e^{-\frac{d_1^2}{\sigma_1^2}} + 1 \right)}{\sigma_1^2} - \frac{8w_1 e^{-\frac{d_1^2}{2(\sigma_1^2 + \sigma_y^2)}}}{\sqrt{(\sigma_1^2 + 1)(\sigma_1^2 + \sigma_y^2)}} + \frac{1}{\sigma_y} \quad (\text{A.8})$$

We have similar but more complex formulas for five and nine circular Gaussian kernels. Fig. A.3 shows the error of an optimization.

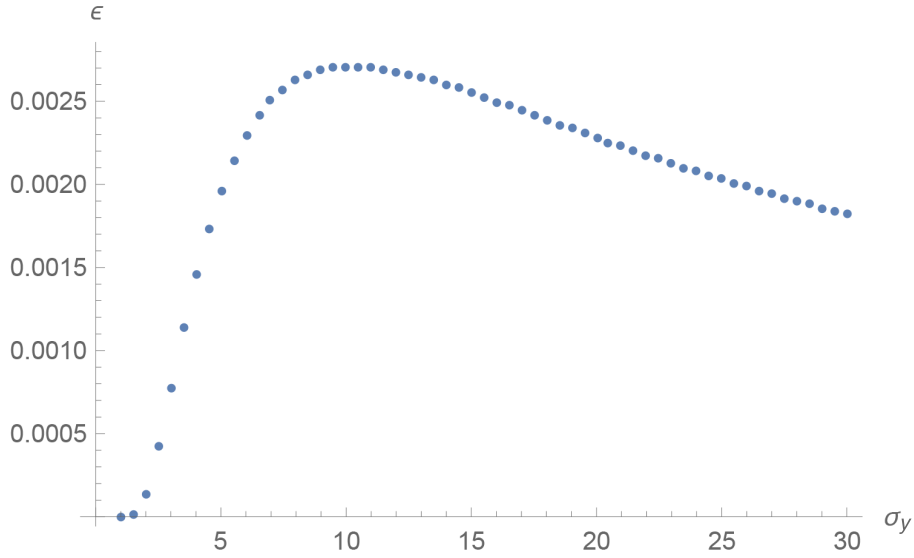


Figure A.3: The error ϵ (Eq. A.5) for three circular Gaussian kernels. The horizontal axis is the major axis standard deviation σ_y .

Next, we sample σ_y at a regular interval and compute the optimal parameters for each of the sampled values. We use Mathematica's NMinimize function for this optimization. The parameters for an arbitrary value of σ_y are obtained by interpolation. We used the curve fitting function of Mathematica for this interpolation. The results of the curve fitting are given in Table 1, 2, and 3 in the ESM.

Achievement

Journal publication:

- Namon Potee, Nelson Max, Kei Iwasaki, and Yoshinori Dobashi. "Temporal and Spatial Anti-aliasing for Rendering Reflections on Water Waves". In: *Comp. Visual Media* (2021). DOI: 10.1007/s41095-021-0204-1. URL: <https://doi.org/10.1007/s41095-021-0204-1>.

Conference publication:

- Namon Potee, Nelson Max, Kei Iwasaki, and Yoshinori Dobashi. "Temporal and Spatial Anti-aliasing for Rendering Reflection on a Water Surface". In: *Proceeding SIGGRAPH '19 ACM SIGGRAPH 2019* (2019). Los Angeles, US, July, Posters Article No. 50.
- Namon Potee, Kei Iwasaki, Yoshinori Dobashi, and Tsuyoshi Yamamoto. "Efficient Adaptive GPU Path Tracing". In: *The International Workshop on Image Electronics and Visual Computing 2017 (IEVC2017)* (2017). Da Nang, Vietnam, February.
- Namon Potee, Yoshinori Dobashi, and Tsuyoshi Yamamoto. "Real-time Glittery Surfaces Rendering". In: *ACM SIGGRAPH VRCAI 2015 14th ACM International Conference on Virtual Reality Continuum and Its Applications in Industry* (2015). Kobe, Japan, October, Posters Article No. 8.

Conference talk:

- Namon Potee, Nelson Max, Kei Iwasaki, and Yoshinori Dobashi. "Micro-specular-highlight real-time rendering". In: *IPSJ SIG Technical Report* (2021).

- Namo Podee, Nelson Max, Kei Iwasaki, and Yoshinori Dobashi. "Adaptive Supersampling for Rendering Reflections on Water Surfaces". In: *IPSJ SIG Technical Report* (2020).
- Namo Podee, Nelson Max, Kei Iwasaki, and Yoshinori Dobashi. "Circular Gaussian anti-aliasing for rendering reflection on water surface". In: *Visual Computing 2019 (Poster)* (2019).
- Namo Podee, Nelson Max, Kei Iwasaki, and Yoshinori Dobashi. "Anti-aliasing for rendering reflection on water surface". In: *IPSJ SIG Technical Report* (2019).
- Namo Podee, Yoshinori Dobashi, and Tsuyoshi Yamamoto. "Combining Cone and Ray Tracing for Ambient Occlusion Computation". In: *IPSJ SIG Technical Report* (2018).
- Namo Podee, Yoshinori Dobashi, and Tsuyoshi Yamamoto. "A Study on Combining Path and Cone Tracing for Efficient Image Synthesis". In: *Electrical and Information Society Hokkaido Branch Joint Convention 2017* (2017).
- Namo Podee, Yoshinori Dobashi, and Tsuyoshi Yamamoto. "GPU Adaptive Path Tracing without Atomic Instruction". In: *GI-CoRE GSQ, GSB, & IGM JOINT SYMPOSIUM 2017* (2017).

Acknowledgements

This work is not mine but many incredibly talented people who supported me for many years.

First, I would like to thank Prof. Yoshinori Dobashi, my advisor, for all the kindness. He cares and supports me in every step of the study and life. He is friendly, kind, and patient with all the students. He gave truthful and thoughtful guidance to everyone. And he also introduced me to the other kind-nature people.

Second, I would like to thank Prof. Nelson Max and Prof. Kei Iwasaki for all the study's help and collaboration. Both are the most phenomenal researcher that I have ever met. I never have thought I could have collaborated with them. I only heard about their name when I began my academic journey through their teaching materials on the internet. They are also kind and patient to me when I'm struggling in the research.

Third, I would like to thank Prof. Tsuyoshi Yamamoto, Prof. Naofumi Aoki, Mrs. Shizuka Maita, and Mrs. Naoko Kitayama for being kind and taking care of me since my study abroad in Japan.

Forth, I would like to thank Prof. Kunio Kondo, Prof. Pizzanu Kanongchaiyos, and Prof. Vishnu Kotrajaras for being incredible teachers and introduced and help me to study abroad in Japan.

Fifth, I would like to thank Mr. Witawat Rungjiratananon for giving me advice and introduced me to Prof. Yoshinori Dobashi.

Sixth, I would like to thank my family and friends for always backing me up for any troubles I had faced or might face.

Last, this work is impossible without many people's help and support. I have to apologize for unable to include everyone's name in here. I wish everyone and the reader peace, health, and happiness. Thank you for being here.