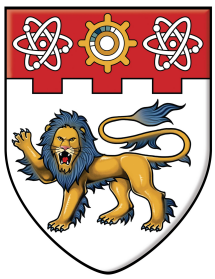


NANYANG
TECHNOLOGICAL
UNIVERSITY

EFFICIENT COMPUTATIONS OF SCALABLE CAUSTIC RENDERING AND RECONSTRUCTION

BUDIANTO TANDIANUS

SCHOOL OF COMPUTER ENGINEERING

2015

EFFICIENT COMPUTATIONS OF SCALABLE CAUSTIC RENDERING AND RECONSTRUCTION

BUDIANTO TANDIANUS

SCHOOL OF COMPUTER ENGINEERING

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy

2015

Acknowledgements

In this opportunity I would like to express my gratitude especially to my supervisors Dr. Henry Johan and Prof. Seah Hock Soon, for the numerous guidance and invaluable advice they have given during my research. I would also like to thank Prof. Seah Hock Soon for providing a great research environment. I deeply appreciate constructive feedbacks given by my internal thesis reviewer Assoc. Prof. Zheng Jiamin and both of my external thesis reviewers. I am also grateful to the lecturers of the courses I took during the PhD study for sharing the knowledge.

During the study, I was acquainted with many wonderful colleagues who gave me great technical, educational, moral, and many other types of supports. I thank Dr. Li Bo for his help in many aspects during my study. I am also thankful to Mr. Nicholas Mario Wardhana, Mr. Koa Ming Di and Dr. Quah Chee Kwang for being interesting discussion colleagues. Many thanks to Mrs. Xu Xiang for being a great friend and giving me invaluable life advice and moral support. Last but not least, many thanks to other colleagues in Nanyang Technological University for the colourful and interesting discussions during my study.

I deeply appreciate the financial support provided by Prof. Seah Hock Soon and Dr. Henry Johan through the CACAni (Computer Assisted Cel Animation) project for the scholarship, and Assoc. Prof. Lin Feng and Prof. Seah Hock Soon in the Non-Rigid Kernel project for the work study and my employment as a Research Associate in Nanyang Technological University. I am also thankful to our technicians Mr. Yee Lee Poh, Mr. Max Lim Tze Yuen, and Mrs. Goh Lay Kian for the technical assistance in the labs.

Last but not least, I also would like to thank my family and all my friends in Singapore, in Indonesia, and in other places around the globe for the support and encouragement during the study.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Abstract

In Computer Graphics, research on caustic can be divided into two categories: caustics rendering and inverse caustics. In general, caustic computations are expensive. Therefore, one of the main challenges in caustic computation is on the scalability issue. Firstly, it is challenging to render real-time caustics of scenes under environment illumination as we must consider all light from environment. Secondly, it requires a very high computational cost to render spectral caustics (rainbow-like colour effects) since we must consider the whole range of visible wavelengths. Lastly, in the inverse caustic computation, i.e. computing the geometry of a caustic object (an object that reflects and/or refracts light) given an input caustic pattern, most work focus only on a single input caustic pattern. For multiple caustic patterns, it is a challenging problem due to the shape and size differences among the input caustic patterns which are difficult to satisfy. In this thesis, we address these scalability issues in caustic research. Our main idea in solving these issues is adjustment of inputs such that the problems are tractable. We present our solutions as follows.

Real-time caustics under environment illumination In this research, we compute real-time caustics of a rigid caustic object under environment illumination (represented as an environment cube map). To achieve this, we precompute the caustic patterns on the surrounding space of a caustic object based on several predefined directional lights. In the rendering, we adjust the environment light source by approximating it as several directional lights by using our proposed environment cube map segmentation technique and integrate their contributions in the rendering. Our technique is able to render cast and volumetric caustics under environment illumination in real-time by using GPU.

Spectral caustic rendering We propose a two-step acceleration scheme by considering spectral characteristics of the scene elements, such as index of refraction distribution of caustic objects and spectral reflectance of surrounding surfaces. In our first acceleration step, instead of spawning rays for every visible wavelength, we adjust the visible

wavelength inputs by clustering them based on the similarity of the refraction angle such that we can represent several wavelengths as one light ray. In the second acceleration step, we compute a scene-dependant importance level of each wavelength cluster in order to determine the refinement iteration amount. Our scheme can achieve speed up of up to around 100 times while maintaining the rendering quality.

Inverse caustics We compute the geometry of a caustic object that can reconstruct a set of input caustic patterns with each caustic pattern is located at an user-input distance from the caustic object. The inputs pose difficult constraints due to the differences in the caustic patterns to be satisfied. To solve this problem, we propose a two-step optimization technique in which we adjust the position and size of the caustic regions in the first step and we adjust the caustic shapes in the second step. We also present an additional step to improve the intensity of caustic patterns. Our technique is able to construct a caustic object for various types of input patterns and we validate the results by using mental ray rendering.

Our work has several applications, such as accelerated rendering for various multimedia applications (e.g. computer games and simulations), and arts (computer-generated movies and pictures, and physical installation of caustic objects).

Keywords: caustics, photorealistic, real-time rendering, environment illumination, spectral rendering, inverse problem, optimization.

Chapter 1

Introduction

1.1 Background and Motivation

Caustics are bright patterns visible on surfaces due to the presence of refractive and/or reflective objects (**caustic objects**) in a scene. Emitted light that hits a caustic object is reflected and/or refracted by the caustic object. Reflection event occurs because the light is reflected by a surface and refraction event occurs because the light transmits to another medium which has a different density which causes the changes in phase velocity (this characteristic is described as index of refraction). Consequently, the light paths change and some of them may converge on points on the surrounding surfaces and those points become relatively brighter compared to other points on the surfaces. This phenomenon is mainly caused by the geometry structure and the material properties of the caustic objects that redirect the light. For example, as shown in an example diagram of caustic formation in Figure 1.1, the incoming directional light is refracted by the spherical caustic object. The refracted light converges in the other end of the caustic object. If we put a surface around the light convergent region, we will see caustic patterns. As we can observe in real life, caustic patterns are prevalent in the scenes that have caustic objects. Figure 1.2 shows photographs or real-life scene containing caustic effects.

In the real world, we may also notice colourful caustics even though the light source

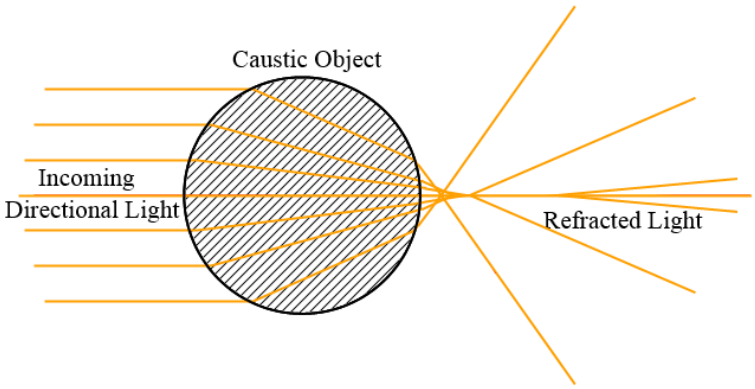
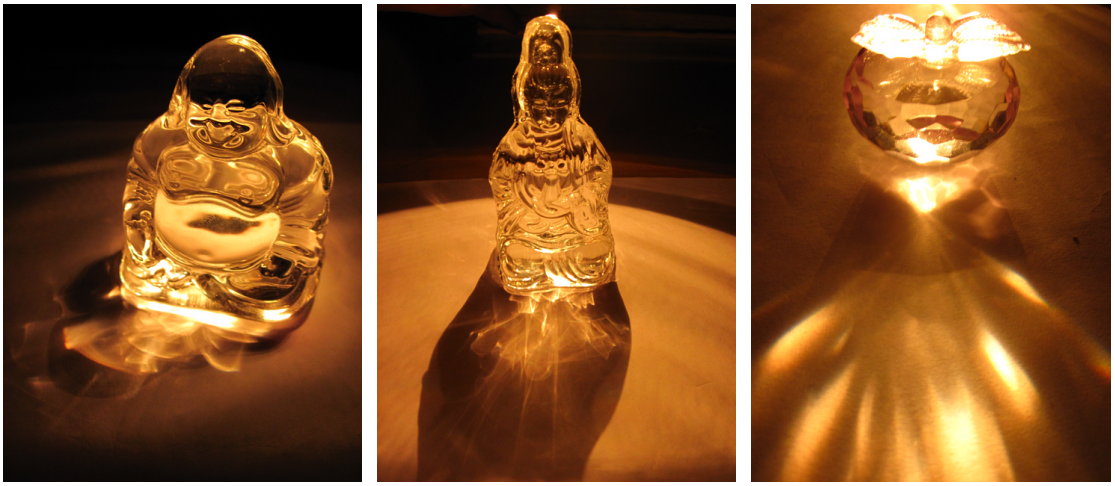


Figure 1.1: Illustration of caustic pattern formation.



(a) (b) (c)

Figure 1.2: Photograph examples of real-life caustic patterns formed on surfaces because the caustic objects (with the shapes of religious figures (a and b) and a fruit (c)) are illuminated from behind.

is a plain 'white light' and the caustic object is colourless. This phenomenon occurs due to the index of refraction difference of the caustic object across the wavelengths. In this case, white light that hits the caustic object will be split into several colour lights (with each corresponding to a wavelength). As a result, we see caustic patterns that are colourful and rainbow-like. We show some examples of real-life spectral caustics in Figure 1.3.

The complicated and interesting caustic patterns may generate the need to have controlled caustic pattern shapes. For example, people may consider producing caustic patterns with certain desired shapes. In this case, the question is not 'how to generate caustic

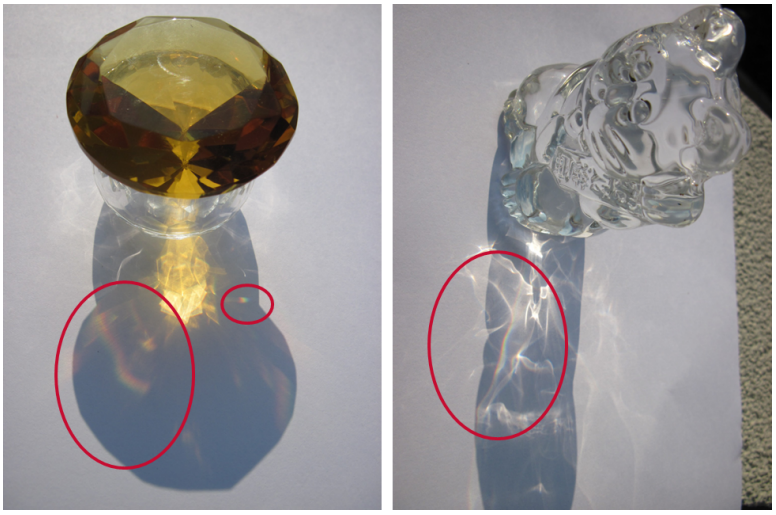


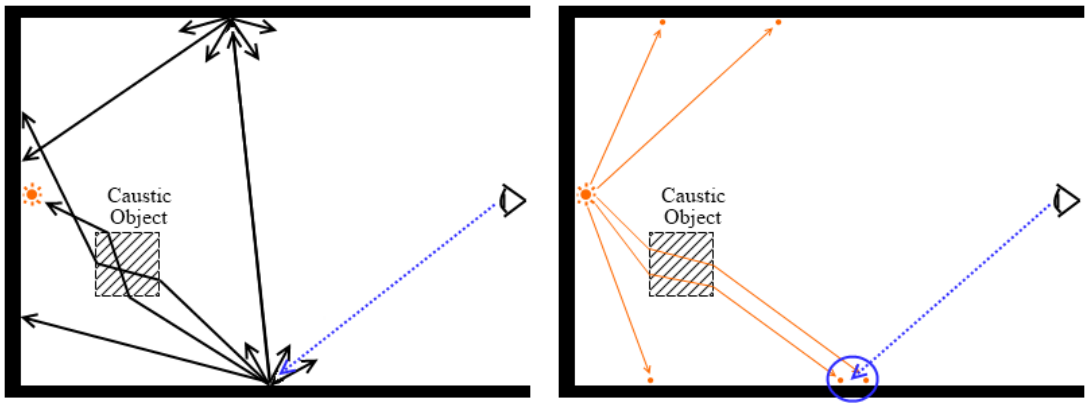
Figure 1.3: Photographs of real-life spectral caustics.

patterns', but 'what is the caustic object shape that can produce the desired caustic pattern shapes'. In order to solve this problem, we have to perform inverse caustic computation, that is computing the geometry of a caustic object such that when it is illuminated it can produce the desired input caustic patterns. This interesting problem has some applications such as in visual arts and security.

Let us now move our discussion on caustics in computer graphics. Caustics are important factor for visual realism in computer generated images of scenes with caustic objects. The reason is that, in order for the audience to consider the rendering result to be realistic, they should see the caustic patterns which are caused by the reflection and refraction characteristic of caustic objects. As an analogy, the audience may expect to see sky reflection effect on a glossy car surface, or subsurface scattering effect on a glass of milk.

The most straightforward way to generate caustic patterns is by spawning and tracing a set of rays over the hemisphere of a visible point in the scene. The rays are traced (and in turn many rays will be spawned every time a ray hits a surface) to the light source. If any of the traced ray hits the light source, then the irradiance or brightness of the said visible point in the scene is increased. This technique is also illustrated in Figure 1.4a and is similar to the distributed ray tracing [1, 2].

This rendering technique has a major drawback, i.e. wasted ray intersection computa-



(a) Distributed Ray Tracing (b) Photon Mapping

Figure 1.4: Two basic techniques for computing caustics.

tion since many of the spawned rays are not guaranteed to hit the light source, especially if the size of the light source is small. We will discuss in more detail the related rendering techniques based on this ray propagation idea in Section 2.1.1.

In order to make caustic computation more efficient, we can use the photon mapping technique [3]. In photon mapping, photons (packets of light energies) are traced and spawned from a light source (instead of toward a light source as in distributed ray tracing). As a result, the caustic computation becomes more efficient. The emitted photons are deposited on surfaces and in the later step (photon gathering), for each visible point on a surface, the nearby stored photons are gathered in order to approximate the irradiance or brightness of the caustic patterns. Photon mapping is illustrated in Figure 1.4b.

Even though the photon mapping technique can improve the efficiency of caustic computation, however, it is still expensive. This is due to several factors: intersection computation between photons and geometries, storage for photons, and searching of nearby photons in the photon gathering step. Nevertheless, photon mapping is still a favourite technique to compute caustics (Section 2.3). We will also provide a review on some photon mapping work in Section 2.1.2.

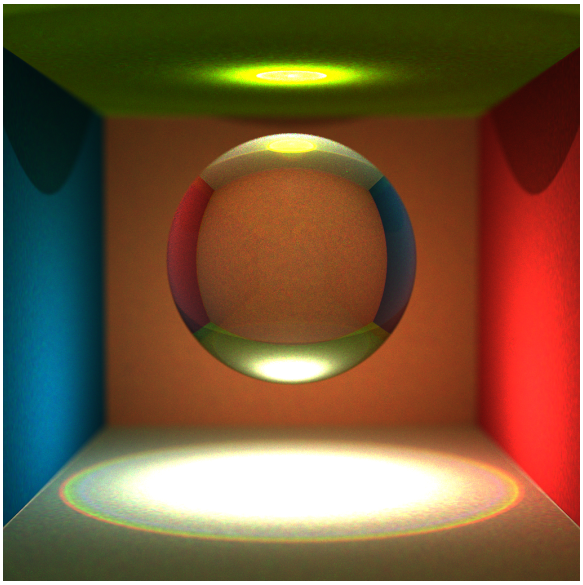
Photon mapping is also used to generate caustics in established rendering engines such as mental ray [4]. Several scene properties related to caustic rendering can be set by users.

For example, some basic caustic rendering parameters such as photon power and number of photons emitted by light sources, refraction indices of caustic object, number of gathered photons and maximum number of gathered photons in caustic receiver surface (generally a diffuse surface). It also provides additional parameters to enable artists to tune further, such as shininess and transparency of the caustic object. As mental ray is an established renderer and it is used in movie industry, hence we use it to generate reference images in several occasions in this thesis. In terms of accuracy, because mental ray uses photon mapping which is a biased method, its rendered results are biased.

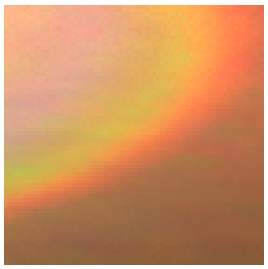
For real-time caustic rendering, due to the computational cost as explained above, several compromises have to be made. For instance, only considering the first refraction, approximating multiple refractions, or precomputing the caustic patterns. We provide some review on the related work on real-time caustic rendering in Section 2.3.2.

In the pursue of ultimate realism, it is important to generate spectral caustic effects. In many implementations of spectral caustic rendering, it is common to consider only wavelengths that correspond to red, green, and blue colours during rendering. However, as shown by Lai and Christensen [5], this approach is not able to produce accurate results. Thus, it is necessary to take into account the visible wavelengths (from 380 nm to 780 nm) for physically-based rendering (spectral rendering). In this case, each parameter used in rendering such as light power, index of refraction, and surface reflectance is not represented by a single value or three values (each corresponds to red, green, and blue colour), but by a set of values over the visible wavelengths.

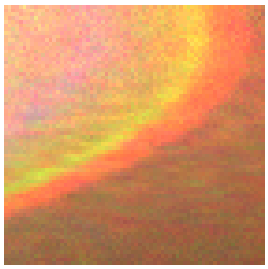
A straightforward way to perform spectral caustic rendering is to render the interaction between densely sampled wavelength with caustic objects and combine the results (we call this a "Brute Force method"). The obvious problem of this approach is its high computational cost. One recent implementation of spectral caustics is provided by Hachisuka [6]. He performs spectral caustic rendering by using the Stochastic Progressive Photon Mapping (SPPM) technique [7]. In his implementation, he randomly chooses a visible wavelength to be processed in each SPPM iteration. He assumes that all wavelengths share the same stat-



(a) Scene configuration



(b) Brute Force, F10 light



(c) Random, F10 light

Figure 1.5: Comparison of Brute Force and Toshiya's random sampling method. (a) Scene configuration. (b) Brute Force. (c) Random sampling. All the experiments are done with the same total number of 802000 iterations.

istical information (i.e. gathering radius and flux). However, by doing so, the wavelength that is chosen randomly in the beginning will have different amount of contribution compared to the wavelengths chosen in subsequent iterations.

In Figure 1.5, we show a comparison between SPPM Brute Force rendering and SPPM rendering with random wavelength selection (Hachisuka's approach). The implementations of both approaches are based on Toshiya's implementation [6]. The renderings were done using the F10 light source [8]. As we can see in the figure, the rendering results based on Hachisuka's random wavelength sampling are generally noisier due to several peaks in the light Spectral Power Distribution (SPD, a function of light power over wavelength) that are chosen only a few times in the random wavelength selection. To render spectral caustics,

we have to compute the caustic pattern for every visible wavelength. For instance, we perform separate or independent photon mapping pass for each wavelength. Consequently, it is computationally expensive to generate such effect. Thus, there is also the need to be able to render spectral caustics in a relatively faster speed. We discuss several related work in spectral rendering in Section 2.2. Most of the work improve the computation by compressing or representing the spectral information with fewer values. The existing work, however, is for general rendering and it is not optimized specifically for caustic rendering.

Research on inverse caustics in computer graphics emerges recently. Existing work solves inverse caustic problem only for a single caustic pattern, i.e. computes the caustic object shape which can reconstruct the single input caustic pattern. They employ various algorithms or techniques to achieve the goal, such as Direct Search Algorithm [9] and Simulated Annealing [10]. Non-linear optimization technique, such as Simulated Annealing, is preferred due to the non-linearity nature of the problem. The inverse caustic problem is basically a combinatorial problem that requires finding the optimized combination in the solution space that can yield the best caustic reconstruction.

The existing work can reconstruct the input caustic pattern pretty well, all frequencies in the input caustic pattern are mostly preserved in the reconstruction. As they only consider a single caustic pattern, there are no shape differences between several caustic patterns that can physically prevent the reconstruction feasibility. Hence, their optimization focuses on rearranging the caustic object surface such that it is smooth and at the same time it can reconstruct the input caustic pattern.

Despite numerous work in caustics, caustic computation is still expensive especially if the problem is scaled up. The followings are three challenges in scaling up caustic computations:

- **Light source** In most of the work on real-time caustics, caustic object is illuminated by one or few point or directional light sources. The main challenge in scaling up this computation is caustic rendering under environment illumination. In the environment illumination, the caustic object is illuminated by many lights from every

direction in the environment. Solving this problem will enable us to generate real-time caustics under environment illumination in some real-time applications such as computer games.

- **Wavelength** Given an incident light to a caustic object, due to the differences in the index of refraction of each wavelength, the light (which physically is a combination of light power over the visible wavelength range) is split. As a result, sometimes we can observe rainbow caustics in real life. In many of the work, the caustic object is always assumed to have a constant index of refraction value, with at most only three with each corresponds to red, green, and blue light wavelengths because typically our rendering framework and image storage are RGB-based. A more accurate rainbow caustic computation is slow and difficult as we have to take into account the whole visible wavelength range (380 nm to 780 nm). By tackling this problem, we can accelerate spectral rendering in some applications such as product design involving glasses, diamonds, etc.

- **Single input caustic pattern** In the inverse caustics, we compute the caustics object geometry that can produce the input caustics pattern. The problem becomes difficult if we have to reconstruct multiple input caustic patterns by using only one caustic object. This is due to the differences among the input caustic patterns which constrain the optimization process. By obtaining solution of this problem, we can produce interesting art pieces, perform information encoding and validation tests.

In this thesis, we propose methods to solve the scalability problems mentioned above. We present the overview of our work in the following Section 1.2.

1.2 Overview and Contributions of Our Research

Our research work aims to solve the scalability issues in caustic computation. Our main strategy in solving these issues is by adjusting the inputs. For instance, for the case with

many lights (environment illumination), we cluster the lights. Similarly, in spectral caustics rendering, we cluster the wavelength based on refraction similarity. In inverse caustics, we adjust the input parameters (such as position of input caustic patterns) in order to mitigate the input caustic pattern constraints. In the following sections, we present our contributions in more detail.

1.2.1 Real-Time Approximate Caustics under Environment Illumination

In this first work, our contribution is the efficient way for generating real-time caustics under environment illumination. Our proposed technique can achieve real-time rendering with visual quality which is comparable to the offline rendering results produced by mental ray.

We precompute the caustic patterns of a caustic object for several directional light sources. The precomputed caustic patterns are recorded on a set of concentric spheres (**caustic spheres**) surrounding the caustic object. It is important to precompute caustic patterns in order to avoid expensive photon tracing and photon gathering in the rendering. In the rendering, given the direction of the light source, we choose and interpolate the caustic patterns of several nearest precomputed directional light sources.

Our technique can be extended to multiple directional lights by repeating the techniques of the single light source rendering for several directional light sources. One extension of our rendering technique is to render approximate caustics under environment illumination. In the straightforward rendering using environment illumination approximated as a cube map, each pixel in the map acts as a single directional light source. It is inefficient to render by iterating through all pixels. To solve this, we approximate the input environment illumination as several directional light sources. We determine the approximate light sources by segmenting the environment map using our proposed segmentation technique.

Using our technique, it is also possible to render volumetric caustics. We render the

volumetric caustics using ray casting and we sample the caustic spheres for every sample point in the cast ray.

Applications Some of the main applications are photorealistic caustic rendering in interactive applications. For instance, we can generate convincing caustic rendering under environment illumination in computer or video games, in product demo, or in old/heritage building walkthrough simulator. Our work is suitable for these applications as most real-time or interactive computer graphics applications nowadays illuminate scenes using an environment map instead of just a single or few point light sources.

1.2.2 Spectral Caustics Rendering

In our second work, our contribution is a two-step acceleration scheme for spectral caustics rendering that can achieve rendering results close to the reference result with huge acceleration magnitude.

Our proposed two-step acceleration scheme for full spectral caustic rendering is as follows. Firstly, to reduce the computational cost of handling visible wavelengths, we cluster them based on the similarity in refraction angles. This clustering takes into account the characteristics of a caustic object, i.e. index of refraction. The benefit of the clustering is the reduced intersection test during the ray tracing and photon tracing, as a bulk of rays of several wavelengths is reduced to a single ray. It is also worth mentioning that our proposed clustering is similar to the recent trend in ray tracing acceleration by packeting coherent light rays [11]. Basically, the ray packeting approach exploits coherency of the light ray direction in the spatial domain whereas our acceleration scheme exploits the index of refraction similarity in the spectral domain. Secondly, we compute the importance level of each wavelength cluster based on its perceptual and geometrical factor. In computing the perceptual factor, we take into account the spectral property of material reflectance, light power, and human eyes sensitivity. As for the geometrical factor, we sample the materials surrounding the caustic object in order to know approximately how much each of surface

material may receive caustic patterns. In our experiment, we perform the rendering using Stochastic Progressive Photon Mapping (SPPM) technique [7] implemented in a GPU ray tracer engine called OptiX. We determine the number of SPPM iterations based on the computed importance function.

Applications Our two-step acceleration scheme has an application in the offline spectral rendering of caustics. By using our two-step acceleration scheme, we can improve the rendering speed. Our work can be used in caustic object design and visualization software. Moreover, our second acceleration step is also applicable to general global illumination rendering as it considers the reflectance of the caustic receiver surface.

1.2.3 Caustic Object Computation based on Multiple Caustic Patterns

We propose a two-step optimization method augmented with an intensity correction step for solving inverse caustic problem based on multiple caustic patterns. Our result is validated by rendering caustic patterns of the generated caustic object by using mental ray rendering software.

Computing the geometry of a caustic object given a set of input caustic patterns such that the computed caustic object can reconstruct the input caustic patterns is generally difficult. This is due to the differences among the caustic patterns in terms of shapes and intensity. Hence, our final contribution in this thesis is a technique for solving this problem.

As the constraints are too difficult, we propose a two-step optimization method in order to solve this problem. Basically, we relax the input constraints by allowing slight adjustments to the inputs. In the first optimization step, we iteratively adjust the positions and sizes of the input caustic patterns and in the second optimization step, we adjust the shapes of the input patterns. In these optimizations, we use simulated annealing. After these two optimization steps, we add an additional computation step, called intensity correction. The intensity correction improves the caustic pattern intensities by iteratively adjusting the reconstructed caustic pattern intensity.

Applications There are several possible applications of this work, such as in art, (e.g. generate interesting caustic effects), information encoding (the secret information is encoded as caustic patterns), and validation tests (to validate the quality of produced caustic objects by comparing the similarities between the real-world caustic patterns and the simulated caustic patterns). We will discuss the applications in more detail in Section 5.6.

1.3 Thesis Organization

The organization of this thesis is as follows. In Chapter 2, we provide a literature review which explains some related work in computer graphics, i.e. general rendering techniques (ray propagation and photon mapping), spectral rendering, caustic-related research (offline and real-time caustic rendering, and inverse caustics). We discuss our research including the implementation and results of real-time caustics rendering under environment illumination in Chapter 3. In Chapter 4, we present our work in spectral caustics rendering. In Chapter 5, we present our contribution in caustic object computation based on multiple caustic patterns. Finally we provide conclusions and propose future work in Chapter 6.

Chapter 2

Literature Review

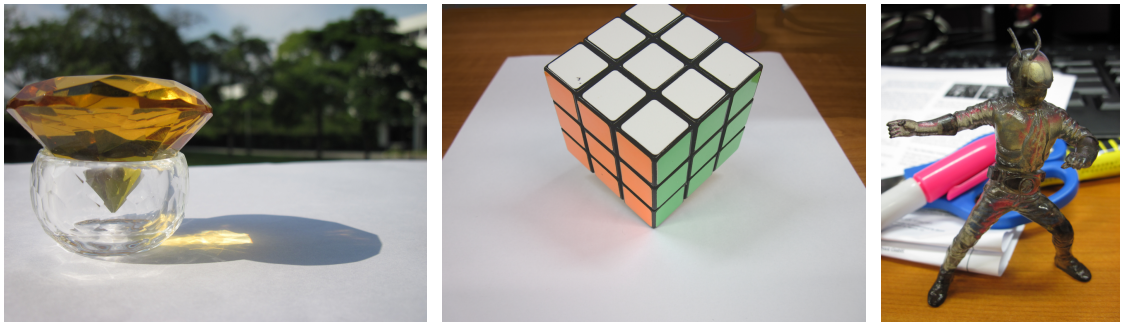
2.1 Rendering Techniques

In real life, light arriving at our eyes undergoes many reflections and/or refractions between objects beforehand. This multiple reflections and/or refractions phenomenon is generally known as global illumination. In contrast, illumination due to only a single reflection is called local illumination (e.g. the basic phong reflection model rendering in openGL fixed pipeline). Some examples of global illumination effects which are not possible to be simulated using local illumination :

- **Caustics** Caustics are formed due to the radiance of multiple light paths converge into a point on a surface. The light paths which are originally uniformly distributed are redirected due to the refraction and reflection caused by the objects (Figure 2.1a).

- **Colour bleeding** When two surfaces are near to each other, the colour transfer between these surfaces can be seen due to light reflection. In the reflection, the first surface reflects light with certain wavelength and the light then arrives at other surface (Figure 2.1b).

- **Subsurface scattering** Subsurface scattering effect is noticeable inside a translucent or partially transparent solid object. It happens due to the material of the object which



(a) Caustics (b) Colour Bleeding (c) Subsurface Scattering

Figure 2.1: Example photographs of real-life global illumination effects.

allows some light to enter and exit the object and at the same time scatters the light inside the object (Figure 2.1c).

Rendering global illumination typically involving solving Kajiya's rendering equation [12];

$$L_o(\mathbf{x}, \omega_\mathbf{o}) = L_e(\mathbf{x}, \omega_\mathbf{o}) + \int_\Omega L_i(\mathbf{x}, \omega_\mathbf{i}) f_r(\mathbf{x}, \omega_\mathbf{o}, \omega_\mathbf{i}) \cos(\mathbf{N_x}, \omega_\mathbf{i}) d\omega_\mathbf{i}, \tag{2.1}$$

where L_o is the outgoing radiance from a point $\mathbf{x}$ on a surface with direction $\omega_\mathbf{o}$, L_e is the emitted radiance from point $\mathbf{x}$ to direction $\omega_\mathbf{o}$, L_i is the incoming radiance to point $\mathbf{x}$ from direction $\omega_\mathbf{i}$, f_r is the Bidirectional Reflectance Distribution Function (BRDF), and $\mathbf{N_x}$ is the normal at point $\mathbf{x}$ on the surface.

The outgoing radiance from a point on a surface is the sum of the emitted radiance of that point and the integration of the reflection from the incoming radiance from the upper hemisphere. Each incoming radiance may come from other point in the scene, thus Equation 2.1 is basically a recursive equation. Note that in Equation 2.1 there is a BRDF term. The BRDF term specifies the amount of light reflected to a direction given an incoming light from a certain direction since typically light does not reflect equally to all directions.

Using the regular expression notation proposed by Heckbert [13], the overall light paths in global illumination are typically $L(S|D|V)^*E$ (with L is the light source, S is specular surface, D is diffuse surface, V is 3D volume with light scattering property such as open

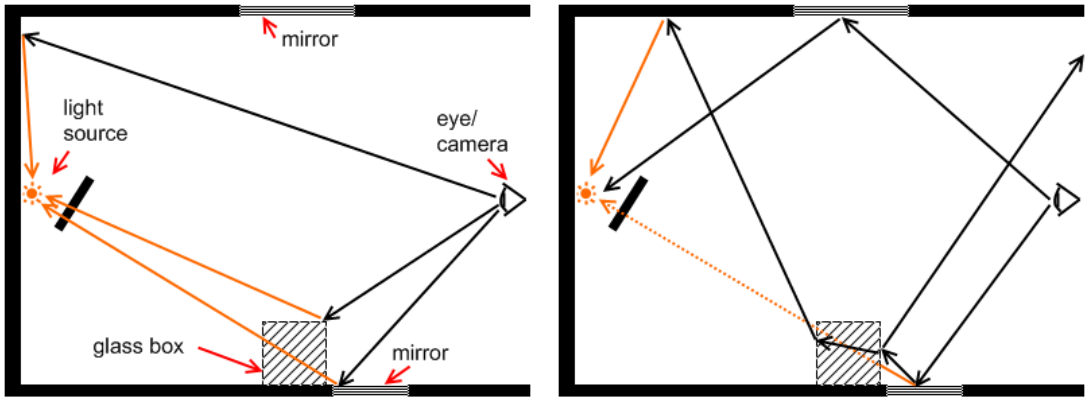
air space with participating media or the inside of a solid object, and E is the eye/camera).

In this section, we present two categories of related work that solve the Kajiya equation or generate global illumination effect, work that based on ray propagation (Section 2.1.1) and based on Photon Mapping (Section 2.1.2).

2.1.1 Ray Propagation Rendering Techniques

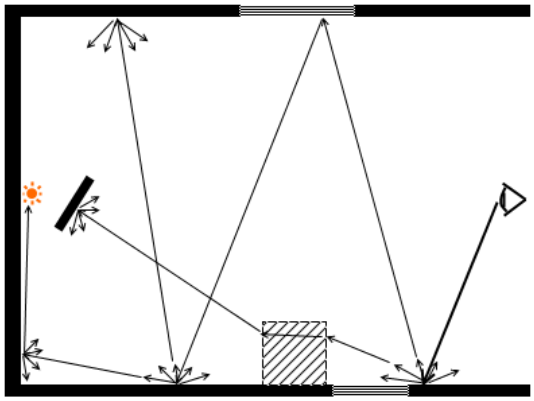
Solution of Kajiya's rendering equation [12] (Equation 2.1) generally involves recording the ray propagation and its interaction in the scene. One of the earliest techniques is the ray casting method proposed by Appel [14]. In ray casting, for each pixel he shoots a ray and computes the intersection between the ray and solid objects. The colour for that particular pixel is computed as the colour of the surface weighted by the cosine between the normal of the surface and the light direction. His technique is only able to compute simple shading of the visible surface, and many effects are not computed (such as shadow, reflection, and refraction). Figure 2.2a shows the illustration of the ray casting, where the black rays are viewing rays and orange rays are the shadow rays. As seen in Figure 2.2a, refraction, reflection, and occlusion from the light source are ignored. Moreover, the surface colour is shaded only according to the material colour of the surface and the angle between the normal of the surface and the direction to the light source.

The ray casting is improved by Whitted [15] by generating three additional rays on the intersection point, which are reflection ray, refraction ray, and shadow ray. The improved ray casting is widely known as ray tracing. The reflection ray is computed as perfect reflection by assuming the surface material is a specular surface. Refraction ray is computed by using the classic Snell's Law if the material is a transparent/glass material. The shadow ray is basically a testing ray to check if the light source is occluded or not by shooting a ray from a point on the surface to the light source. For secondary reflection and refraction rays, once those rays intersect other points, another secondary rays might be generated again. Thus, the colour of a pixel is the product of radiance of all rays spawned by the primary

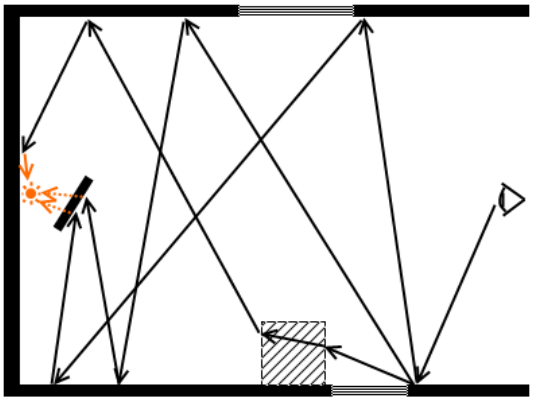


(a) Ray casting

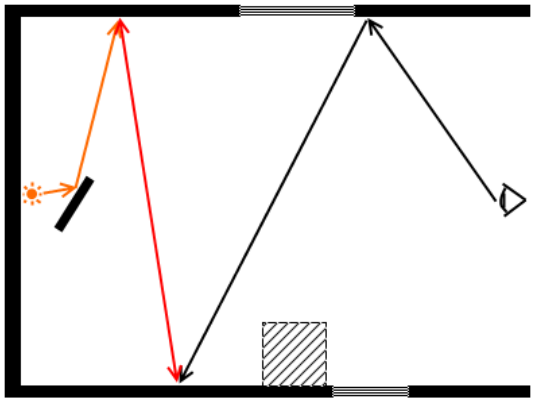
(b) Ray tracing



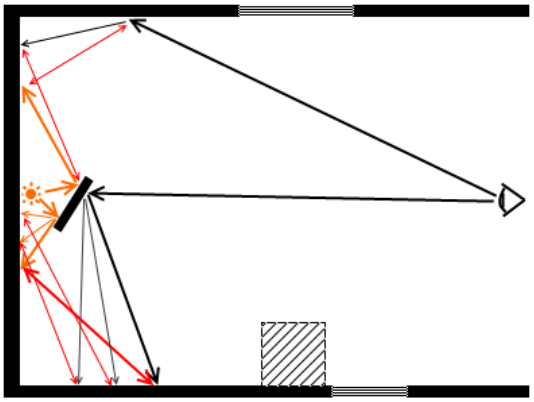
(c) Distributed ray tracing



(d) Path tracing



(e) Bidirectional path tracing



(f) Metropolis light transport

Figure 2.2: Comparisons between ray propagation rendering techniques. The scene consists of a point light source (on the left), a camera (on the right), a glass box (with diagonal shading), two mirror walls (with horizontal shading), and a diffuse wall.

rays shot through that pixel. Figure 2.2b illustrates the ray tracing technique. It traces additional reflected ray if the viewing rays hit specular (e.g. mirror and glass) surfaces, and it traces refracted rays if the rays hit glass surfaces. Shadow rays drawn with dotted lines show that the surface is occluded by something, thus it is in shadow and there is no light contribution calculated for the surface.

For every intersection, however, ray tracing only computes the direct contribution from the light sources and it always assumes every reflection to be a perfect reflection (as in mirror reflection). In Kajiya's equation (Equation 2.1), the irradiance contribution for every point in the scene is not determined by only light from one light direction, but from the whole hemisphere. Cook et al. [1, 2] propose the distributed ray tracing which is the extension of the Whitted ray tracing. In distributed ray tracing, for every point on the surface intersected by primary or secondary rays, they generate a number of secondary rays covering the hemisphere. Distributed ray tracing results with more accurate computation but with a significant increase in computational cost. Figure 2.2c shows an example of distributed ray tracing. Every time a ray intersects a surface, it randomly generates secondary rays in order to compute the contribution from the whole hemisphere.

Another option for distributed ray tracing is path tracing [12]. Instead of generating many rays for every intersection, path tracing only randomly traces a single new secondary ray for every intersection point. As an analogy, distributed ray tracing is a Breadth First Search (BFS) traversal and path tracing is a Depth First Search (DFS) in tree structure with every node in the tree corresponds to a ray. Path tracing is essentially an unbiased and consistent rendering technique, as the value of each pixel is an average of the radiance of all rays shot through the pixel, and the result converges to an accurate result as the number of generated rays are increased. Since path tracing requires a large amount of rays, this technique generates noisy results with less rays. Figure 2.2d presents an example of path tracing. Path tracing randomly generates new set of paths passing through a pixel as shown in Figure 2.2d (the first path segments of the set of paths are assumed to be near to each other since they pass through the same pixel, thus they are drawn as only a ray).

In the scene in which the light source is heavily occluded, many rays in path tracing may fail to sample the light source. Bidirectional path tracing was developed concurrently by Lafortune and Willems [16] and Veach and Guibas [17] in order to solve this problem. In bidirectional path tracing, besides tracing a viewing ray from the camera, they also trace a light ray from the light source simultaneously and they attempt to connect both of these two rays. Figure 2.2e shows the segment connecting viewing ray and light ray as a two-edged red arrow.

As the paths of viewing rays and light rays are recomputed every time a new ray is cast through a pixel, the bidirectional path tracing might fail to sample the light source, thus still exhibiting high variances in the result. To reduce the variance, Veach and Guibas [18] propose metropolis light transport technique. Similar to bidirectional path tracing, metropolis light transport trace both viewing and light rays. Once the paths are connected, they adaptively mutate the existing paths by removing and inserting new path vertices. As a result, metropolis light transport generates results with lower variance. Moreover, metropolis light transport reduces the ray tracing cost as it uses existing paths (Figure 2.2f).

In all of these techniques, it is essential to be able to check the intersection between rays with objects (or primitives forming the objects in the scene). However, the brute force approach which is checking the intersection with all the objects (or primitives) in the scene costs much of the rendering time. For example, in a scene which consists of 3 million primitives, all these primitives are checked in order to determine the nearest intersection for every cast ray. Thus, in order to accelerate the intersection tests, the 3D scene is partitioned and the primitives belonging to every partition are recorded in acceleration structures. During the ray traversal, only primitives stored in the partitions passed through by the rays are checked for intersections. Several acceleration structures have been proposed, such as uniform grid [19], octree [19], kd-tree [20], and Bounding Interval hierarchy [21].

2.1.2 Photon Mapping

Photon mapping is an approximate solution of the rendering equation which is biased and consistent [3]. Additional processing will yield results that converge (consistent), but not necessarily correct (biased). Similar to bidirectional path tracing, the photon mapping technique traces viewing rays (from camera) and light rays (from light sources). Instead of connecting the viewing rays and light rays, photon mapping separates the tracing into two passes. In the first pass, the photon mapping technique shoots photons (in this case, light rays are represented as packet of energy called photons) into the scenes. The photons are reflected and/or refracted (with the path as $L\{S|D|V\}^*D$) (Heckbert's notation) and their energy are stored on the surface they hit. In the second pass, viewing rays are traced from the camera to compute the visible points and the irradiance on those points. Since the visible points most likely might not have photons stored on those points, it is necessary to gather the photons stored around the visible points. In this photon gathering, a fixed amount of photons are gathered within a given maximum searching radius. Thus, the rendering quality mainly depends on two parameters, the number of photons to gather and the maximum gathering radius. Figure 2.3 illustrates the photon mapping technique.

The irradiance estimate in the photon gathering is computed by summing up the power of all of the gathered photons and divided by the gathering area (Equation 2.2).

$$L_o(\mathbf{x}) \approx \frac{1}{\pi r^2} \sum_{i=1}^{n} \Delta\Phi_i(\mathbf{x}), \tag{2.2}$$

where L_o is the total irradiance at the visible point $\mathbf{x}$, r is the gathering radius, n is the number of gathered photons, and $\Delta\Phi_i$ is the photon power. By using this approximate, some points may become blurry due to the uniform averaging. Thus, Jensen [3] proposes using filtering which gives more weight to the photons nearer to the visible points, such as cone filtering and Gaussian filtering.

Jensen and Christensen [22] extend the photon mapping technique for computing illumination in participating medium (such as smoke). Instead of storing the photons on

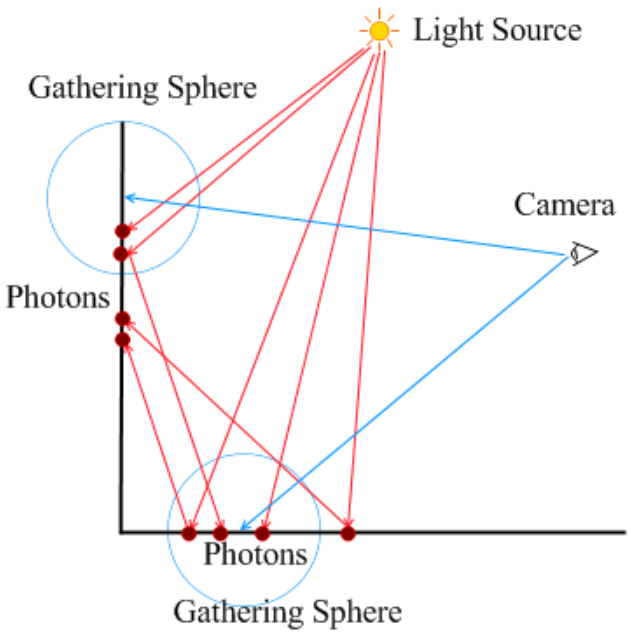


Figure 2.3: Several photons are emitted from the light source and deposited on the surface. In order to compute the irradiance at a visible point, several photons around the visible points are gathered. In this simple example, three photons are gathered within a specified maximum gathering radius. On the horizontal wall, three photons are gathered. However, on the vertical wall, only two photons are gathered as the third photon is outside of the maximum gathering radius.

diffuse surfaces, they store the photons in the participating medium. (hence the light path is $L\{S|D|V\}^{+}V$). The irradiance estimate (Equation 2.2) is modified by introducing a scattering term and changing the averaging term from a circle area (πr^2) to a sphere volume ($\frac{4}{3}\pi r^3$).

By emitting photons equally, some invisible points in the space might have a large density of photons whereas visible points might a have small density. This results in inaccuracy on the visible points and redundant storage or processing for invisible photons. Peter and Pietrek [23] introduce a new step before the photon emission, which is importon emission. The importons are basically photons emitted from camera which are used to determine the visibility of the points in the scene. In the next step in the photon emission, as the photons hit a point in the scene, importons around the hit point is gathered to determine the importance of that point. By using the importon, the photon emission can be refined such that

more photons are shot to the visible parts of the scene.

Schjøth et al. [24] trace the photon differentials instead of the regular photon paths. Similar to the ray differential, during the tracing they also compute the divergence of the photon path. The spread of the photon divergence for each photon is stored in the surface. In the photon gathering, for each visible point they collect the surrounding photons and weigh the contribution radiance based on the distance between the visible point and the centre of the photon differential. This technique is able to reduce noise significantly. However, they need to store additional information, which is the divergence of photons. The photon differential becomes much more complicated on subsequent reflections and refractions. Thus, this technique works well for single reflection or refraction, such as caustics caused by the reflection of a metallic object or by refraction of a water surface.

Instead of shooting all photons and gather them in two passes, Hachisuka et al. [25] split the photon mapping passes into many passes with each pass they only shoot a small amount of photons. For every visible point, they gather the surrounding photons and adaptively adjust the gathering radius based on the amount of photons gathered. Afterward, the photons are discarded, they then shoot another batch of photons and do the photon gathering again whilst at the same time they adjust the gathering radius. By discarding and generating a small amount of photons in every pass, their technique uses a smaller amount of memory and is able to generate high quality results comparable to the traditional photon mapping.

Purcell et al. [26] present the first GPU implementation technique of photon mapping. They store photons in uniform grid in order to accommodate the limitation of GPUs in 2005 which prohibit creation and traversal of kd-tree used in the general photon mapping technique. They propose using the bitonic merge sort in the fragment shader to index and pinpoint the photon locations in the grid. However, this is quite expensive since it requires $O(\log^2 n)$ rendering passes. As such, they propose another approach that uses vertex shader to decide in which location the photon is stored in the grid. Since each cell can contain multiple photons, they suggest dividing each cell into sub-cells, and every photon is stored in a different sub-cell. To decide which sub-cell, they use stencil buffer to route the photon

to an empty sub-cell. In the photon gathering, they propose a kNN-grid method in which they firstly search photons in a cell and then search photons in the nearby cells, gradually increasing the gathering radius. Since they use uniform grid to store photons, their technique suffers from redundant memory allocations as some cells might not have photons. Moreover, they do not use any acceleration structures to compute intersection, thus it will benefit more if they store the scene in an acceleration structure.

Spencer and Jones [27] store photons in a balanced hierarchical kd-tree structure and they adaptively traverse the kd-tree to cut down the photon gathering time. They stop the traversal if the density of the photons are already low in a node and does not make a difference if they traverse further down. To speed up the kd-tree traversal during photon gathering, they cluster photons first before storing the cluster or only the cluster centres into the kd-tree. Budge et al. [28] cluster photons in 6D space (comprise 3D final hit locations and 3D normal of the surface hit by the photon), Wang et al. [29] cluster photons based on only the 3D final hit locations, and Chen et al. [30] cluster photons based on the photons traversal history. Mara et al. [31] conclude that storing photons in screen space tiles yields the best performance and quality.

As photon mapping technique is very expensive to be computed in a single workstation, Güenther et al. [32] present a photon mapping implementation in a distributed environment which consists of multiple workstations. In order to reduce the bottleneck in the network transfer, each workstation computes separately a smaller photon map, which is then combined with the photon maps computed by other workstations. In order to improve the performance, they filter the photon in image space instead of 3D space. By using 9 to 36 workstations, their implementation is able to reach up to six times speed improvement.

To solve the storage problem, Hachisuka et al. propose a progressive method called Progressive Photon Mapping (PPM) [25]. In PPM, instead of emitting and storing all photons, they iteratively re-emit a fraction amount of photons (and discard the photons afterwards) and progressively refine the flux and gathering radius. Fu and Jensen extend the work slightly by performing refinement locally for each visible point [33]. Hachisuka et al. then

extend their work into Stochastic Progressive Photon Mapping (SPPM) [7] for rendering visual effects that requires multiple samples per pixel. The SPPM is a slight novel extension of PPM that in addition to re-emit photons for each iteration, they also reshoot the camera ray through each pixel. All samples of each pixel share the same statistical information such as flux and gathering radius. Hence, SPPM can achieve faster convergent compared to PPM. The SPPM convergence is then further enchanced by combining it with Metropolis Sampling [34] or by using deterministic approach (quasi-Monte Carlo) [35].

Stopping criteria in PPM-based techniques are mostly based on the number of iterations or the rendering time. It would be better if the stopping criteria is based on the convergent and error. Hence, Hachisuka et al. propose an error estimation framework that can compute the statistical error of each iteration [36]. If the error is below a threshold, then the rendering process is stopped.

Knauss and Zwicker later proves that the local statistics information of each PPM iterations, which is the gathering radius, does not need to be stored [37]. Hence, the memory usage in PPM can be reduced further. On the contrary, Kaplanyan and Dachsbacher still consider the gathering radius and they propose a technique for automatic gathering radius [38] computation.

PPM has several extensions such as for rendering animation and volumetric effect. One of the main disadvantages of PPM is the inefficiency when it is applied to dynamic scenes due to the recomputation in every animation frame. Weizz and Grosch mitigate this problem by recording the intersection points [39]. Another improvement work in analysing the geometrical property of the scene was proposed by Hachisuka et al. [40]. They analyse the path visibilty in the scene in order to accelerate the computation.

As with the photon mapping technique, PPM can also be extended to render volumetric effect. Instead of reducing the gathering radius as in the original PPM, Jarosz et al. iteratively reduce the beam radius [41].

2.2 Spectral Rendering

Spectral rendering takes into account the spectral information of densely sampled visible wavelengths and thus it produces more accurate results compared to the general RGB-based rendering [42, 43]. The wavelength factor incurs costs in performance (processing every visible wavelength) and storage (storing the result for each visible wavelength). Due to these factors, RGB-based rendering is more commonly used. For example, Shi et al. convert the necessary spectral information to RGB representation when they render polluted water [44]. Dong [45] proposed a rendering framework for a scene mixed with a different material representation, i.e. spectral (for objects with thin films) and RGB (for objects with simple materials that can be rendered using a traditional RGB approach).

By taking spectral information into account, interesting effects, such as light interference on a thin layer [46, 47, 48, 49, 50, 51, 52, 53], diffraction [54, 55, 56], chromatic lens aberrations [57], and rainbow [58, 59], can be produced. Another interesting work by Xing et al. [60] can perform relighting of photographs. Spectral information is also useful in volume rendering [61, 62].

The bottleneck in spectral rendering is the integration of the information over the visible wavelengths. There is existing work in optimizing this computation. For example, the spectral information can be compressed by using a linear transformation over a set of linear basis functions [63]. Also, it is possible to compress the spectral information using the wavelet transform [64, 65]. Afterwards, progressive integration is done from one level to another level. Xu and Sun compress the surface BRDF by using Fourier transform and spherical harmonics [66]. Another approach is to minimize the spectral information by simplifying it while keeping the error to a minimum. For example, Zeghers et al. suppress/simplify spectral information over nearby wavelengths by connecting their two suppressed local peaks [67], Iehl and Perochè [68] represent a range of wavelengths as a constant value.

Spectral computation can be optimized by analysing the spectral information and performing subdivision or decomposition as necessary. For example, Deville et al.'s approach

(which is the improvement of Meyer's work [69]) takes into account discontinuities and peaks in the spectral domain and they subdivide the spectral information so that it is possible to integrate analytically each of the subdivided range [70]. Meanwhile, Sun et al. take a different approach by decomposing the spectral information into smooth parts (that are transformed using Fourier basis) and spiky parts [71, 72]. They consider the spiky parts as light sources such as CIE Standard Illuminant F11 which have peaks in some of the wavelengths. If they compress solely using Fourier basis, then the spiky parts will incur large errors.

Radziszewski et al. define a probability distribution function of the visible wavelengths based on their trial-and-error experiments [43]. They use the probability distribution function to sample wavelengths to be combined into a cluster in their photon mapping-based spectral rendering. However, they do not take into account light Spectral Power Distribution (SPD, a distribution of light power over wavelength domain), material reflectance and the index of refraction. Instead of using a trial-and-error probability distribution function, Evans and McCool sample a set of stratified random wavelengths [73] based on the SPD of the light source and put them inside a cluster. In this case, they do not take into account the material property of the caustic object. As a result, light rays with wide angular refraction difference might be grouped into a cluster.

It is also possible to perform spectral rendering by using photon mapping. However, the storage of the photons will become a big issue since we have to store the photons of all wavelengths. Lai and Christensen shoot photons for each wavelength, and they store the photons in RGB representation [5]. During photon gathering, they convert the photons energy back from RGB to spectral. However, due to the metamerism phenomenon, the conversion from RGB to spectral might not yield correct results. In a spectral rendering framework proposed by Iehl and Perochè [74], they shoot and deposit photons with randomly chosen wavelengths. They also save the first intersection of these photons with caustic objects. In the rendering during photon gathering, if the nearby visible point contains these photons, they then spawn photons from the first intersection but with different

wavelengths.

In the GPU implementation [6] of the Stochastic Progressive Photon Mapping [7] by Hachisuka, he randomly selects a wavelength for photon shooting in each iteration and converts the photons into RGB format for photon gathering. In addition, the input for material reflectance is in RGB format. When a photon hits the surface, he converts the material reflectance from RGB to spectral representation by using Smit's proposed technique [75]. He also approximates the CIE XYZ by using gaussian quadratures [76], hence he does not need to save the eye response function for each wavelength.

Elek et al. propose a spectral ray differential rendering technique [77]. They consider the difference of refraction direction with respect to wavelength changes. They use the difference in refraction direction as photon spread/differential (instead of based on the emitted light spread as originally proposed by Schjøth et al. [24]). By applying their technique to an existing approximate caustic rendering technique (proposed by Wyman and Davis [78]), they can achieve real-time rendering. Due to the nature of their proposed spectral differential (1D spread over the wavelength for each photon), their caustic reconstruction result exhibits line patterns.

For more review on spectral work, readers are advised to read Devlin et al.'s state-of-the-art report on spectral rendering [79]. Their report covers various spectral rendering techniques and systems as well as visual effects that are generated with spectral renderers such as polarization and fluorescence effects. As noted by Devlin et al., there were still limited spectral renderer systems as of 2002. From some post-2002 related work we provide in this section, we can see that spectral rendering work is still growing.

2.3 Caustic Rendering

Caustics are formed by light rays focused on surfaces caused by reflective and/or refractive objects. Several techniques of caustic rendering have been proposed, ranging from offline rendering to real-time rendering. In order to generate caustics in real-time, some tech-

niques assume only a single reflection and/or refraction (eg. underwater caustics). Some later techniques are able to handle real-time caustics with more than a single reflection and/or refraction. Moreover, some offline spectral caustics work and real-time rendering of caustics caused by non-homogenous caustic objects are also presented.

2.3.1 Offline Caustic Rendering

One of the most commonly used techniques to compute caustics is photon mapping [3]. It computes photons in $L\{S|D|V\}^*D$ and stores in a global map. However, caustics are less visible in the rendering using the global map as global map stores not only caustic photons, but also diffuse interreflection photons. To generate more pronounced caustic patterns, another photon map called caustic map is used. In this technique, the photons are shot directly to the reflective/refractive surface and they are recorded in the caustic map when they hit a diffuse surface (hence, the photon path is LS^*D). Jensen extends the technique for generating caustics on non-lambertian surface by also storing the incoming photon direction [80]. In this case, he takes into account the BRDF of the non-lambertian surface (in the original photon mapping, the BRDF of a diffuse surface is always 1, thus he does not need to store the incoming photon direction).

$$L_o(\mathbf{x}, \omega_\mathbf{o}) \approx \frac{1}{\pi r^2} \sum_{i=1}^{n} f_r(\mathbf{x}, \omega_\mathbf{o}, \omega_\mathbf{i}) \Delta\Phi_i(\mathbf{x}, \omega_\mathbf{i}), \tag{2.3}$$

where L_o is the total irradiance at point $\mathbf{x}$ which is viewed from direction $\omega_\mathbf{o}$, r is the gathering radius, n is the number of gathered photons, f_r is the BRDF at point $\mathbf{x}$ on the surface, $\omega_\mathbf{i}$ is the incoming photon direction, and $\Delta\Phi_i$ is the photon power.

In photon mapping rendering, isolated photons can produce unpleasant polka dot patterns. To remove this effect and noise, Spencer and Jones [81] relax and redistribute the photons stored on the surface based on the blue noise distribution. As a result, they can reduce the noise and at the same time maintain the sharpness of the caustic patterns.

As PPM is an extension and enhancement of photon mapping, most rendering methods

based on PPM can render caustics with significant acceleration compared to the traditional photon mapping. Similar to the caustic rendering in the original photon mapping technique, Dodge et al. also separates the computation between diffuse radiance and caustics [82]. To achieve this, they perform path tracing and analyse the path. Based on the path, they decide to compute the diffuse radiance of caustics. Georgiev et al. [83] and Hachisuka et al. [84] took another approach by combining path tracing and photon mapping. Their proposed method try to connect the light paths from the eye with the scattered photons.

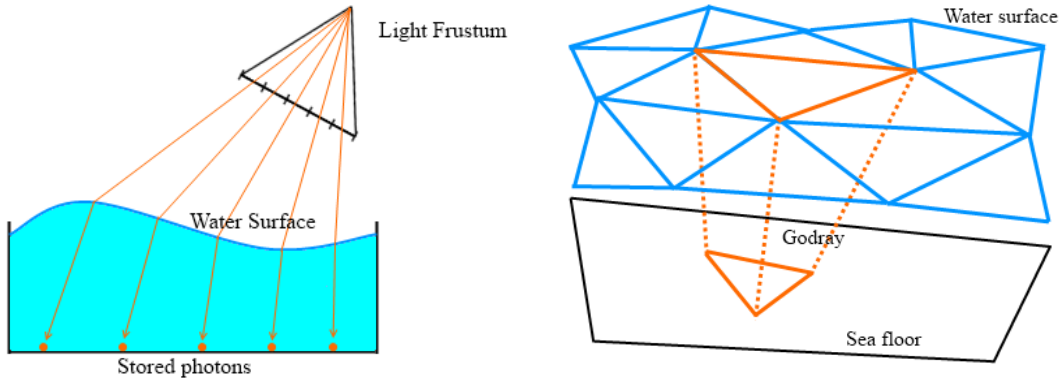
2.3.2 Real-Time Caustic Rendering

Single Reflection and/or Refraction

One of the most common caustic effects caused by single refraction is underwater caustic effect due to the refraction by water surface. Generally, underwater caustics are easy to compute since it only require single pass photon tracing. To achieve this, water surface is represented as a height field and the photon tracing (each photon is represented as a point) can be done easily by rendering the water surface from the light source position in which the photon is traced in the fragment shader [85, 86]. If the underwater surface is a simple object (e.g. flat surface), then the intersection can be computed analytically [87]. Figure 2.4a illustrates how to render caustics using this technique. By representing each photon as a point, this technique may suffer from noisy caustics due to the undersampling.

In underwater we see not only caustics on the surface, but also god rays/light shafts that travel to the seabed. In order to visualize the god rays/lightshafts, the underwater photon path can be rendered as line primitives [88]. Generally, by using fragment shader, the rendering is done from two directions, one from the light source (viewing the water surface to compute refracted photon paths) and one inside the underwater (to render all the photon paths as line primitives.) Similar approach has been used recently to render single scattering volumetric caustics in general scenes [89, 90].

Considering the fact that in general water surface is rendered as a triangle mesh, it



(a) Heightfield-based caustic rendering. (b) Triangle sweeping-based caustic rendering

Figure 2.4: Two examples of real-time caustic rendering techniques.

is possible to compute the refraction direction of the photon on each vertex of the water surface triangle. Afterwards, an illumination volume is built by sweeping every triangle down the underwater with the sweeping direction determined by the refraction direction of the photon on each vertex. By using this approach, asides from preventing noisy caustics, it is also possible to generate underwater caustics and the underwater god rays/light shafts [91, 92, 93, 94, 95, 96, 97]. Instead of sweeping the triangles based on the refraction, they can be swept based on the reflection direction in order to generate caustics formed by metallic surface [95, 98]. This approach is feasible to be implemented in GPU since every vertex of the water surface mesh corresponds to each fragment. Figure 2.4b shows the basic idea of this approach.

Wand and Straßer [99] propose a real-time single reflection caustics method. In their technique, for each point on a specular surface, they generate a caustic cube map by rendering the light source. The cube map is re-projected to the diffuse surface in the scene. Thus, caustics on every diffuse surface is the accumulation of the cube map projection of many points on the specular surface. As a result, this technique requires huge storage in order to store the cube map of many points on the specular surface.

Multiple Reflections and/or Refractions

Achieving caustics arising from multiple reflections and/or refractions in real-time (using GPU) is an ambitious attempt as the photon direction coherency breaks down after first reflection and/refraction. One possible approach for rendering caustics caused by multiple reflections and/or refractions is by precomputing the caustic information for each caustic object. In the rendering, based on the relative position and orientation of the caustic object to the light source, the precomputed data is sampled and applied the precomputed caustics to the scene.

Iwasaki et al. [100, 101, 102] precompute a table storing the final outgoing direction of a photon from a point on a caustic object, given the incoming direction to a point on a caustic object. In the precomputation, they compute accurate multiple reflections and/or refractions, that may involve photons exiting and re-entering caustic object. The exiting and re-entry is not saved in the precomputation. Hence, their proposed method works best for convex caustic objects.

For caustics under a directional light, Wyman et al. [103] precompute the caustic patterns of a set of predefined directional lights. For each light, they compute its photon paths and record their intersection with the space surrounding the caustic objects. In this case, they discretize the surrounding space into a uniform grid or a set of concentric shells with uniform radii between consecutive concentric shells. Since in a scene, the incoming light mostly does not coincide with the light directions in the precomputation, they choose the three nearest precomputed light directions by using sorting and continue by rotating the three chosen light directions along with their caustic patterns such that they coincide with the incoming light direction. Their technique, however, is inefficient in the storage since they store the irradiance uniformly in the space. Since light attenuates quadratically, more information should be stored in the space near the caustic object. Moreover, their technique only supports single light and they do not take into account light occlusion, and in the rendering they need to use several workstations.

Liu et al. [104] precompute light loci by using several directional lights. Instead of storing the entry and exiting (along with incoming and outgoing) photons direction, they firstly identify the points in the space which has high density of photons based on the number of photons passing through those points. Based on this information, they parameterize the focused points as light loci and store them as the precomputed data. Compared to Iwasaki et al.'s precomputation [100, 101, 102] and Wyman et al.'s precomputation [103], Liu et al.'s precomputation require less memory storage. However, due to the parameterization on a few selected points in space, their precomputation data might miss some caustic patterns.

Computing photons paths in GPU for the second and subsequent reflections/refractions is not a trivial task. Thus, for computing caustics caused by two or more reflections and/or refractions in real-time, the approximate intersection can be computed by using line-search technique. In this method, caustic object surface is approximated by using several heightmaps, and the intersection is computed by finding a point along the refracted path that intersects one of the heightmaps. Mostly current real-time GPU-based caustics (caused by two or more reflections and/or refractions) are based on this approximation technique [105, 106, 107].

Using the aforementioned GPU approximation refraction techniques some real-time GPU-based caustic rendering techniques with multiple reflections and/or refractions are proposed [78, 108, 109, 110]. As with other GPU based caustic rendering, photons from each fragment are shot by rendering the scene from the light source (light space). In this approach, photons are represented as polygons [78, 111] or points [78, 108, 109, 110].

In the polygonal representation [78, 111], they represent photons as points. When the photons leave the caustic objects, they are used to construct caustic polygons. The photons used to form a caustic polygon are the photons which are adjacent to each other in the original light space. In this approach, they cannot use the illumination volumes from the beginning since after one reflection and/or refraction the coherency of photon directions breaks down which make the subsequent illumination volume tracing become difficult. Instead, they form the caustic polygon (in other word, one infinitely thin slice of an illumination

volume) on the photons leaving the caustic object. In this representation, they can reduce noise in caustics (since they render caustics as polygons rather than points). However, this representation might lose some caustic details and suffer from high fill rate.

The point representation suffers from noisy caustic pattern due to undersampling (less photons). To reduce the noise, Wyman and Dachsbacher [112] propose splatting the photons with varying radii. However, with large splat the caustics will become too blurred and with small splat noise will appear. Wyman later proposed splatting the photons in multi-resolution caustic maps [113] or adaptively refine the photon shooting by using deferred shading [114].

Caustics of Non-Homogeneous Caustic Object

So far, the proposed caustic generation techniques assume the caustic object to be homogeneous (index of refraction is uniform inside the caustic object). Consequently, the photon paths inside the object are straight since there is no difference in index of refractions inside the caustic object. Thus, once knowing the photon hit location and direction on a surface of the caustic object, we only need to compute the exiting point. However, for non-homogeneous caustic object, the photon path will be bent due to the difference of the index of refractions inside the caustic object. Generally, solution for this rendering consists of three passes, which are voxelization of the caustic object (with each voxel stores an index of refraction), photon or light tracing for storing the irradiance in the voxel, and viewing ray tracing for computing visible points in which we are going to sample the irradiance.

Ihrke et al. [115] use an uniformly voxelized caustic object and they trace light as wavefront patches formed by several light particles. Since the caustic object is non-homogeneous, they propagate the wavefront in n (index of refraction field) by using an equation based on the Eikonal equation (Equation 2.4).

$$n\frac{d}{dt}\left(n^2\frac{d\mathbf{x}}{dt}\right) = \bigtriangledown n, \tag{2.4}$$

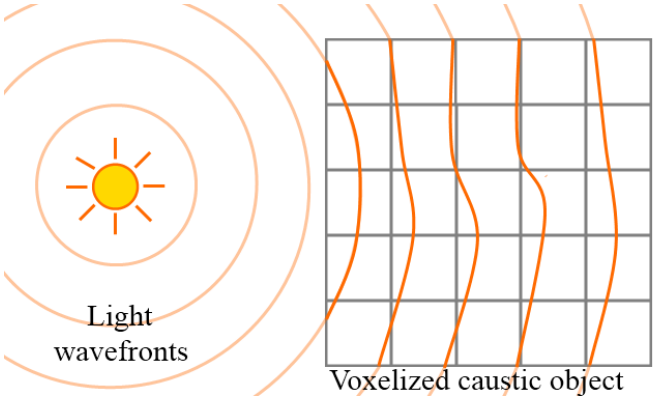


Figure 2.5: Ihrke et al.'s solution for real-time non-homogeneous caustic rendering [115].

where t is the time and $\mathbf{x}$ is the position in the space.

As the wavefront particles visit each voxel, the wavefront particles deposit the irradiance into the voxel. Lastly, they trace the viewing ray which is also based on Eikonal equation since the viewing ray is likely to be bent due to the difference of index of refractions between the voxels.

Their technique is able to render caustics of non-homogeneous object in real-time. However, the tracing in each voxel is very expensive and they need to pre-compute again if the relative orientation and position of the caustic object to the light source is changed.

Instead of voxelizing the caustic object uniformly, Sun et al. voxelize the caustic object in multiresolution voxels in which several voxels which have almost the same index of refractions are grouped together [116]. By using this, they are able to speed up the photon and viewing ray tracing considerably. For the photon tracing path, similar to Ihrke et al [115], they compute the photon paths based on the Eikonal equation.

2.3.3 Inverse Caustics

Given only a caustic pattern, the caustic object that can produce such output is computed. This is a hard problem since the *a priori* knowledge of the input is not available. One example is the work presented by Bottino et al. [117] and Mitra et al. [118]. They compute a 3D geometry that can satisfy the inputs which consist of a set of silhouettes or shadow

patterns. In the inverse caustics research, the main goal is to compute the shape of the reflective or refractive caustic object in order to reconstruct the input radiance distribution (or caustics).

Patow and Pueyo [119] compute the reflector shape in an optical set (consists of a reflector, light source, and diffusor) given the radiance distribution as an input. They represent the reflector as grids and they iteratively adjust the grid vertices such as positions and number of vertices based on the similarity with the intended radiance distributions. The whole process took many days even though they can obtain radiance distribution similar to the input. They improve the work by allowing users to set the range of the solution space [120] (the lower bound and the upper bound of the reflector shape). Hence, a user has more control in determining the reflector shape. They later increase the performance by using GPU [121] and they can reduce the processing time into magnitude of hours. In real life we might need a reflector system which consists of a multiple reflector. Oliker present a solution for such system, but with the assumption of only two mirrors in the system [122].

In parallel with the aforementioned work, there are some work in representing the reflector surface as a parametric surface such as a Bezier surface [123], a NURBS surface [124], and a B-Spline surface [125, 126]. As a result, during the optimization they optimize the control points instead of grid vertices which in the end can produce smooth reflectors in a relatively fast speed (due to the small number of parameters to be optimized). However, as Papas et al. mention [127], the parameterized technique has a difficulty with highly complex caustic images, and they show that Finckh et. al's method [126] cannot reproduce all frequencies of the caustic pattern.

Weyrich et al. generate a microgeometry reflector given a single reflected caustic pattern input [128]. The caustic object is subdivided into uniform cells (facets), and they compute the optimized orientation of each cell that can produce a caustic pattern similar to the input pattern. Similarly Kiser et al. [129] also subdivide the caustic object into uniform cells. Papas et al. [127] improve the work of Weyrich et al. [128] by generating a refractor caustic object on a larger scale. Moreover, they are able to prevent noise on the reconstructed

caustic pattern by computing the surface of each facet based on the Gaussian distribution. Similar to Weyrich et al. [128], they employ several optimization costs in order to generate the caustic objects. The generated caustic object can be smoothened by applying a post-processing computation, i.e. solving poisson-based equation [128, 127, 130].

In the aforementioned work, almost all cells have different shapes compared to each other. On the other hand, Yue et al. [131] emphasize on modularity by reconstructing an input caustic pattern from a caustic object which consists of many smaller pieces of caustic object cells with predefined shapes. Their caustic object cells are divided into ten types with each type refracts light to a predefined direction.

In the similar spirit as inverse caustic research, Papas et al. improve their work further to the reconstruction of intended refraction effects [132]. Given a seemingly random image pattern, they compute the shape of the uniformly subdivided caustic object such that the viewed refraction of the random image pattern through the caustic object is meaningful (i.e. has a certain distinct shape recognized by our brain).

2.4 Comparison with Our Work

2.4.1 Real-Time Approximate Caustics under Environment Illumination

Our proposed technique performs real-time caustic rendering under environment illumination. Our proposed technique precomputes the caustic patterns and uses them in the rendering. The following is the comparison between our work and the previous work.

We firstly compare the precomputation. Iwasaki et al.'s precomputation [100, 101, 102] does not store light path of the light which exits and re-enters caustic object. Hence, if a plain diffuse surface is placed on that path, no caustic pattern due to that light path will be formed. Our precomputation approach, however, precomputes caustic pattern on the empty space surrounding caustic object, hence can handle such situation. Our approach is

similar to Wyman et al.'s precomputation [103]. However, we specify the radii of concentric shells (that store caustic patterns) taking into account light attenuation. As we show in Section 3.5.2, our results show closer quality to the mental ray rendering. Liu et al.'s precomputation involves approximation of the precomputed bulk light path [104]. Our caustic pattern precomputation, however, does not involve approximation of the caustic patterns. As our caustic precomputation is done by considering multiple refractions, the computation is more accurate than the work that approximates multiple refractions [78, 108, 109, 110].

Compared to the previous work, our work supports rendering under environment illumination thanks to our preprocessing step that performs environment cube map segmentation. Each cube map segment is treated as a directional light source and we integrate caustic patterns of all light directions (from environment cube map segmentation). Our work also handles occlusion caused by objects around the caustic object.

2.4.2 Spectral caustic rendering

Our proposed acceleration rendering consist of two steps that considers the caustic object, surface, light power, and human visual properties. We also consider the geometry surrounding the caustic object.

One main difference between our work with the previous work is that we consider the properties of caustic object. Our contribution in this aspect is that we analyse the index of refraction of the caustic object over the visible wavelengths and we cluster the wavelengths based on refraction direction similarities. In the implementation, our work process spectral information with 1 nm spacing instead of compressing the spectral information or using simpler representation as done by some of the previous work [63, 64, 65, 66, 67, 68, 69, 70, 71, 72]. Our implementation is thus suitable for GPU as for each light ray we can do integration (i.e. compute the light power reflected by a surface over a range of wavelengths) in parallel.

In our second acceleration step, we compute the importance level based on various

aforementioned factors, unlike the previous work that do not consider those factors [73, 43, 6]. Thus, the results of using the previous work may be noisier or may take longer time to compute, especially on the difficult case with the light source that has sharp peaks in its Spectral Power Distribution.

Elek et al.'s approach [77] is similar to ours with some differences. Firstly, our approach is based on ray packeting (clustering wavelengths based on angular refraction similarity) and our result does not exhibit line patterns. In our formulation, we use angular refraction difference instead of refraction direction difference. Note that our acceleration schemes are applied to an offline renderer. Similar to Elek et al.'s work, our acceleration schemes can also be applied to a real-time approximate caustic renderer. It is possible to combine our work with Elek et al.'s work. For example, we cluster the wavelengths and pass this information to Elek et al.'s spectral differential for deciding which wavelengths to sample. Another possibility is using our second acceleration scheme to decide the amount of resources (such as number of photons) in the rendering of each randomly sampled wavelength in Elek et al.'s spectral ray differential renderer.

2.4.3 Caustic Object Computation based on Multiple Caustic Patterns

We compute the geometry of a caustic object given a set of input caustic patterns such that the computed caustic object can reconstruct the input caustic patterns at several distances. Similar to previous work [128, 127], we represent our caustic object as a grid of cells. We also use Simulated Annealing in the optimization.

In all previous work, the input is only a single caustic pattern. On the other hand, we propose a new challenge in which we compute the geometry of a caustic object based on a set of input caustic patterns. To solve the caustic pattern difference problem that preclude reconstruction of all input caustic patterns, we perform optimization on the caustic patterns, namely choosing a good adjustment on their parameters (i.e. refining the positions and sizes of input caustic patterns). We also iteratively refine the caustic object geometry in the last

step, the intensity correction step. As such, our optimization is slightly different to the previous work that performs optimization and refinement only on the caustic object itself.

Chapter 3

Real-Time Caustics in Dynamic Scenes with Multiple Directional Lights

In this chapter, we present a real-time caustic and volumetric caustic rendering technique for a scene with multiple directional light sources. The scene is assumed to contain one caustic object and several diffuse objects which receive cast caustic. We also extend the method to perform rendering under environment illumination taking into account light occlusion. Figure 3.1 shows some examples of our rendering results. In our proposed rendering technique, we firstly precompute the reflective and/or refractive caustic patterns at the surrounding of caustic objects based on a set of directional lights. Afterwards, we use the precomputed caustic patterns during the rendering pass in order to efficiently compute the caustic intensities at arbitrary locations. As our technique is based on precomputation, the number of polygons only slightly affects the caustic computation during rendering. Specifically, it only affects the fillrate in rendering. In contrast, the number of polygons affects rendering time significantly in ray tracing-based methods as we have to perform searching operations. Our proposed rendering technique has some differences with the technique presented by Wyman et al. [103] and we discuss these in Section 3.1. In order to achieve real-time performance for the rendering under environment illumination, we approximate the environment illumination as a set of directional lights using our proposed environment

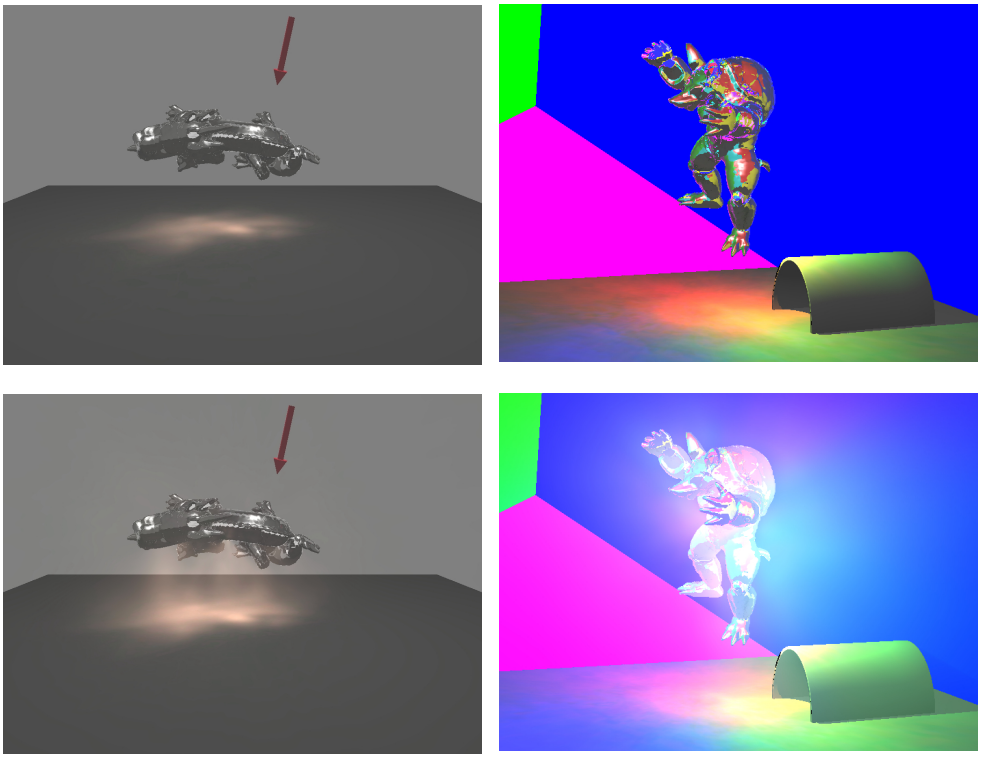


Figure 3.1: Caustic rendering using our proposed technique. First column shows caustics under one directional light source (light direction indicated by the arrow). Second column shows caustics under environment illumination. The first row shows only the cast caustics, and the second row shows cast caustics and volumetric caustics.

cube map segmentation technique (a cube map is a set of six textures, with each texture corresponds to one cube face). We show the main steps of our technique in Figure 3.2.

To evaluate the rendering quality of our results, we compare the images generated using our technique with the images generated using mental ray (available in Maya 2010). We use photon mapping in mental ray to generate caustic patterns. Mental ray is a well-known renderer commonly used in movie industry to generate visual effects for some well-known movies with outstanding visual quality. As we show in Section 3.5.2, our technique can generate caustics which are visually similar to the caustics generated using mental ray.

3.1 Precomputing Caustic Patterns

The purpose of the precomputation is to record the caustic intensities of a caustic object

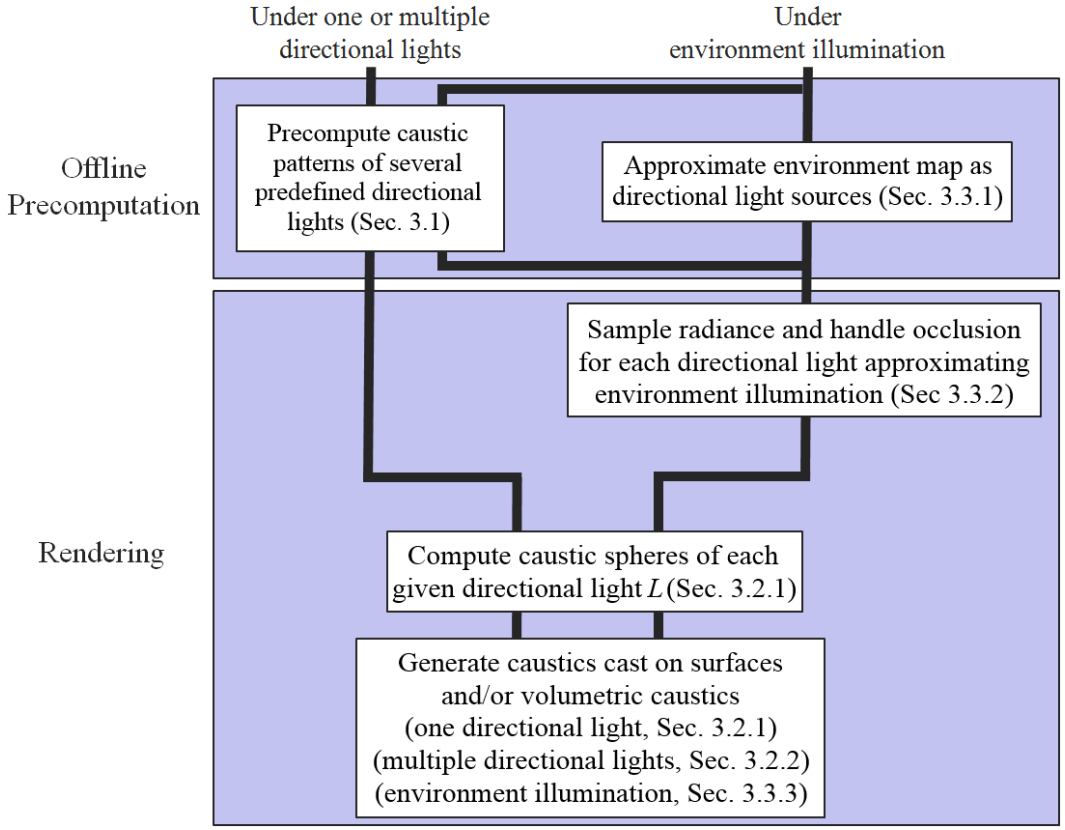


Figure 3.2: The main steps of our technique.

at the 3D local region based on several predefined directional lights. In this precomputation, caustic patterns of only several predefined directional lights are precomputed since it is impossible to record the caustic patterns of all possible directional lights. Hence, in the rendering, given an input directional light, its caustic patterns are computed by bilinearly interpolating the precomputed caustic patterns of several predefined directional lights. The caustic intensities of each predefined directional light are recorded on a set of concentric spheres with varying radii (**caustic spheres**). We use a spherical representation in storing the precomputed caustic patterns since we record caustic intensities of all directions surrounding the caustic object. Moreover, during the precomputation we can easily compute the intersections between the reflected and/or refracted light with the caustic spheres by using ray-sphere intersection computation. Caustic intensities are recorded on several concentric spheres with varying radii in order to cover the 3D space surrounding the caustic object and we precompute the caustic intensities by using photon mapping [133].

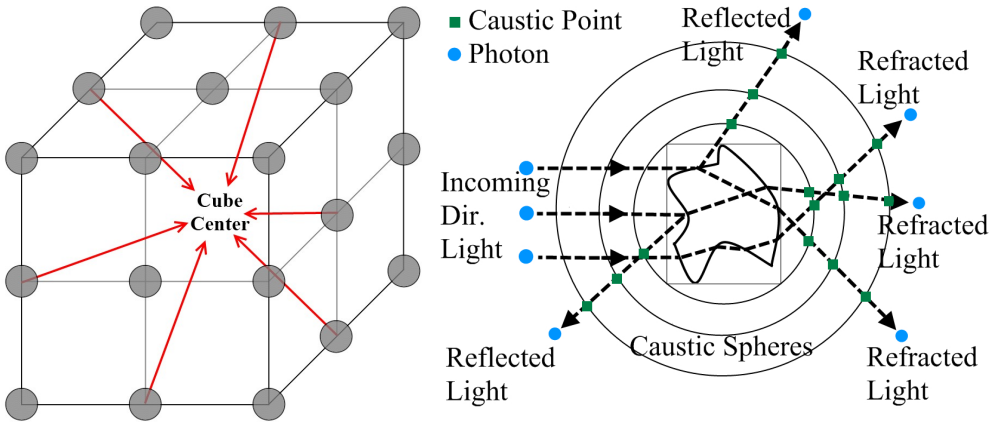


Figure 3.3: (a) Predefined light directions for the precomputation are computed based on the directions from each vertex, centre of each edge, and centre of each face to the cube centre. We show some example directions by using arrows. (b) Precomputation of caustic patterns. The incoming directional light (implemented as photons) is reflected and refracted by the caustic object.

We propose the following precomputation. We sample 26 light directions based on a cube, which consist of the directions from the cube vertices (8 directions), the centre of the cube edges (12 directions), and the centre of the cube faces (6 directions) to the centre of the cube. The predefined light directions are illustrated in Figure 3.3a. Then, the reflective and the refractive caustic patterns (for each directional light) on a set of concentric spheres centred at the origin of the caustic object are computed and recorded as shown in Figure 3.3b.

Unlike Wyman et al. [103] who linearly change the radii of the spheres, we suggest a quadratic function to determine the radii of the caustic spheres since caustics weaken non-linearly due to light attenuation. Thus, the changes in caustic patterns are visually more apparent near the caustic object, thus a higher priority is given to the recording of caustic patterns near the caustic object. The benefit of using this approach is that we can densely record caustic patterns on the space near to the caustic object than the space far from the caustic object. We can save more space by using this quadratic approach compared to the uniform approach. Uniform approach stores caustic pattern on equal radii, hence uniform approach is redundant as it uses the same amount of caustic spheres to store caustic patterns

far away from the caustic object and caustic patterns near to the caustic object. Thus we have the advantage of reduction of the number of caustic spheres while maintaining visual quality. The formula for computing caustic radii is shown in Equation 3.1.

$$r_i = r_{\min} + (r_{\max} - r_{\min})((i-1)/(s-1))^2, \tag{3.1}$$

where r_i is the radius of the i-th caustic sphere, $r_{\min}$ is the minimum radius, $r_{\max}$ is the maximum radius, and s is the number of caustic spheres. We set $r_{\min}$ to be slightly greater than the distance from the centre of the caustic object to the nearest surface of the caustic object. Assuming the maximum dimension of the object's bounding box d is max{width, height, depth}, we set $r_{\max}$ to $4d$ based on our experiment in which caustic patterns were barely noticeable on the caustic sphere with the radius of $4d$. If we infinitely increase the number of precomputed caustic patterns, the radii difference between caustic spheres will approach zero, which is essentially the same as precomputing caustic for every point in the space surrounding the caustic object.

The caustic patterns of each caustic sphere are stored in a 2D image such that the x and y axes of the image represent the longitude and latitude of the caustic sphere. The issues of storing the caustic patterns in this format are the redundant storage near the poles and the requirement of using trigonometric functions to sample the caustics during the rendering. However, the advantage is caustic spheres in latitude-longitude format can be stacked into a 3D texture (which contains several layers of 2D textures) and therefore we can efficiently sample the caustics at any point on and between the caustic spheres (or layers of a 3D texture) since GPUs can efficiently sample and interpolate texels with a single sampling instruction. Moreover, the computations of trigonometric functions in current GPUs are relatively cheap. Another option is to store the caustic patterns in other format such as cube map that can provide relatively more uniform sampling compared to longitude-latitude format. However, additional operations are needed to sample and interpolate caustic points across cube map stack or array. Moreover, not all of the current GPUs support cube map

Figure 3.4: Caustic patterns for gargoyle model (with refractive indices for red, green and blue components are 1.5, 1.51 and 1.52, respectively) for two different light directions.

arrays. One possible option is to store each cube map as six separate square textures, but this will incur additional operation in shader as we need to choose which texture side to sample.

From experimental results, the resolution of the caustic pattern images should be at least 512×256 in order to generate good quality caustics. The height of the caustic images is half of the width since we sample the caustic sphere $[0..2\pi]$ in longitude direction and $[0..\pi]$ in latitude direction. Generally, we suggest 16 caustic spheres ($s = 16$), thus the total memory needed to store the precomputed caustic patterns of one caustic object is 512 × 256 (dimension of caustic pattern image) × 3 colour channels (RGB) × 16 caustic spheres × 26 light directions = 156 MBytes. We can precompute polychromatic caustics by setting different refractive indices for the red, green, and blue components of the directional light. We show some examples of caustic pattern images in Figure 3.4.

Instead of storing the precomputed data in latitude-longitude format, we may store them by using a Precomputed Radiance Transfer (PRT) [134] technique which represents the precomputed data as a set of constant values (of basis functions, such as Spherical Harmonics). One main advantage of PRT is more uniform reconstructed data, with less distortion compared to latitude-longitude format. Moreover, PRT approach can save tremendous amount of memory space. One main disadvantage of PRT is the complexity in the computation, as we have to perform rotation and reconstruct various points of the caustic spheres in real-time in the rendering.

The precomputation algorithm is shown in Algorithm 3.1.

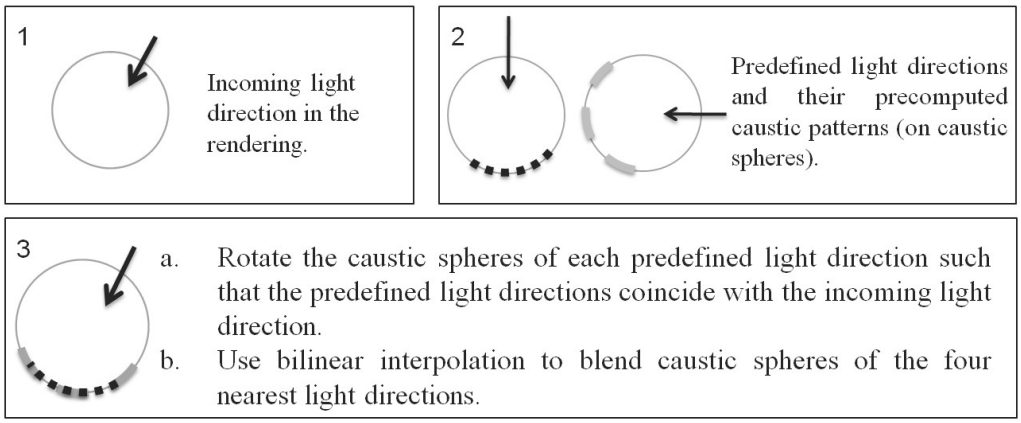


Figure 3.5: Caustic spheres interpolation. For the sake of clarity, we illustrate the process in 2D using only two caustic spheres.

Algorithm 3.1: Precomputation of caustic patterns.

```
1 Sample predefined light directions;
2 foreach predefined light direction do
3     Compute caustic patterns on each caustic sphere by using photon mapping [133];
4     Save the precomputed caustic patterns of each caustic sphere as an image;
```

3.2 Rendering Caustics under Directional Lights

3.2.1 Single Directional Light

Given an arbitrary light direction L, caustic spheres of L can be computed by interpolating the precomputed caustic patterns since most likely L does not coincide with any of the precomputed 26 light directions. Thus, in our approach, we propose to choose the four precomputed light directions nearest to L. Similar to Wyman et al. [103], in order to alleviate the ghosting effect (transition between two patterns, with one pattern fades out and another pattern fades in) due to interpolation, caustic spheres of the four nearest light directions are rotated to align their directions with L. Then, we blend them using bilinear interpolation to produce the caustic spheres of L. This interpolation scheme is illustrated in Figure 3.5.

The caustic intensity at a point on a surface is trilinearly interpolated (point A in Figure

3.6) or extrapolated (point B in Figure 3.6) using the intensities at the eight nearest samples in the caustic spheres of L. Using the above method for computing the caustic intensity at an arbitrary point, we propose the following techniques to generate two types of caustics, caustics cast on receiver objects and volumetric caustics.

Caustics cast on receiver objects To render these caustics, firstly a **temporary cube map** which stores the information of receiver points (coordinates and depth value of each receiver point) is computed. The receiver point information can be computed by rendering the surrounding scene from the centre of the caustic object. Caustic intensities at the receiver points (Figure 3.6) can then be computed based on the coordinates of the receiver points (sampled from the temporary cube map) by sampling the caustic spheres of L. The caustic intensities are then multiplied with the dot product between the normals at the receiver points and the directions to the centre of the caustic object (in order to take into account the cosine term in illumination) and we store them in a caustic cube map. Every pixel of the caustic cube map contains information about the caustic intensities in RGB channel and the depth value (is copied directly from the temporary cube map) in the A (alpha) channel of the cube map. In our experiment, the resolution of the caustic cube map is $512\times512\times6$.

The caustic cube map is then sampled in order to determine the caustics at visible points from the camera. For every visible point, we compute its local coordinates with respect to the caustic cube map and sample the corresponding entry in the caustic cube map. If the depth of the visible point is the same as the depth value stored in the cube map, we add the caustic intensity, otherwise we do not add. The sampling is based on the shadow map technique originally proposed by Williams [135].

Volumetric caustics In the presence of participating media (assumed to be homogeneous), the volumetric caustics can be generated by using the ray casting technique [136]. A ray is cast to the scene for every pixel on the screen and the caustic intensity at each sample point on the ray is integrated. Typically, we use 128 samples on the rays. In order to check whether each sample point is blocked by occluders, its distance to the centre of the

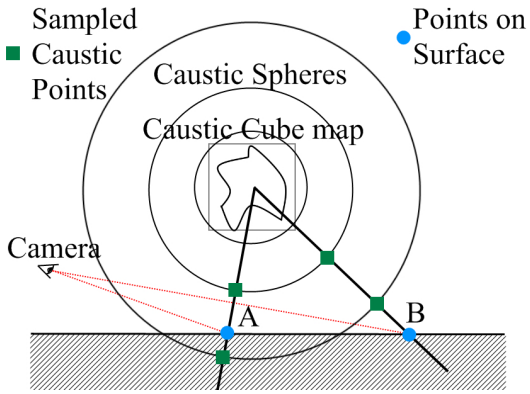


Figure 3.6: Computation of caustic intensities.

caustic cube map is compared with the depth stored in the cube map. If the distance is less than or equal to the depth stored in the cube map, the caustic intensity from the caustic spheres is sampled. Otherwise, this sample point can be skipped as it is blocked by the occluders. In the integration, we consider the light attenuation from the sample point to the camera and the scattering phase function (can be computed using either the Rayleigh phase function [137] or the Henyey-Greenstein phase function [138]).

3.2.2 Multiple Directional Lights

To generate caustics under q numbers of directional lights, we apply the algorithm for one directional light to each light and accumulate the computed caustic patterns on the caustic cube map. Then, the accumulated caustic patterns are projected to the scene. Similarly, for volumetric caustic rendering, for every sampling point on the cast ray we sample caustic spheres of all q directional lights and integrate them. This technique is used to render approximate cast caustics and volumetric caustics under environment illumination (Section 3.3). The cast caustics and volumetric caustic rendering algorithm is presented in Algorithm 3.2.

Note the weight in Line 5 of Algorithm 3.2 is set to 1.0 unless it is rendering under environment illumination (Section 3.3). For rendering under one directional light, Lines 2 - 6 are executed only once.

Algorithm 3.2: Rendering cast caustics and volumetric caustics.

```
1  Render the surrounding of a caustic object into a temporary cube map;
   /* Compute caustic spheres and caustic cube map.                    */
2  foreach directional light L of all q directional lights do
3      Choose four nearest predefined light directions;
4      Rotate their caustic spheres and blend them;
5      Weight them and accumulate them into the caustic spheres;
6      Accumulate the caustic intensities to the caustic cube map;
   /* Compute cast caustics.                                           */
7  foreach visible point in the scene do
8      dist = distance from the visible point to the caustic object;
9      if dist = distance in the caustic cube map then
10         Render the caustic point (by sampling the RGB channel of the caustic cube
           map);
   /* Compute volumetric caustics.                                     */
11 foreach pixel in the screen do
12     Shoot a ray through that pixel to the scene;
13     foreach uniform step along the ray do
14         acc = 0;
15         dist = distance between the point in the current step and the caustic object;
16         if dist ≤ distance in the caustic cube map then
17             Sample the caustic spheres of all q directional lights and accumulate to
               acc;
18         Add the pixel value in the screen with acc
```

3.3 Rendering Caustics under Environment Illumination

In our technique, we represent the environment illumination as an environment cube map. Every pixel in the cube map represents a distant directional light, thus the total irradiance at a point in a scene is the sum of radiance from all pixels in the cube map. However, integrating the radiance from all pixels in the cube map during rendering is impractical. To solve this issue, we segment the environment cube map into several important light regions and represent each of them with a directional light. We then sample the important light regions in the rendering.

3.3.1 Environment Cube Map Segmentation

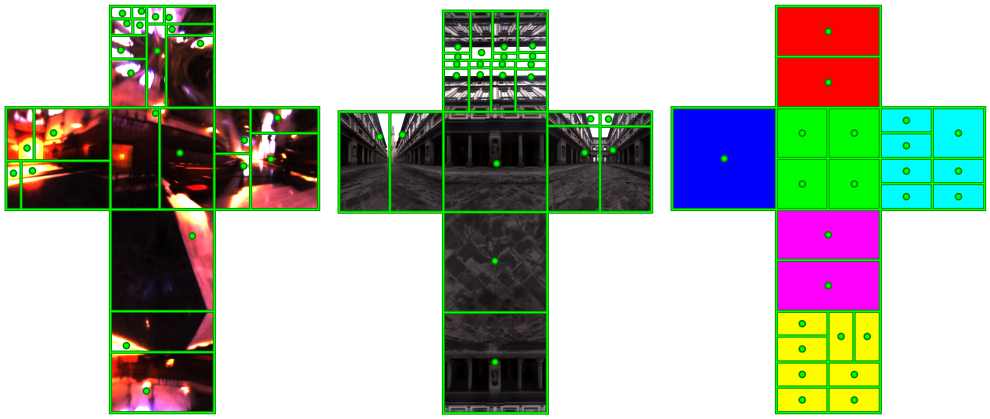
Real-time rendering techniques often approximate an environment illumination with several lights. Debevec [139] recursively segments the environment map (latitude-longitude format) into two regions having almost equal total radiance until a number of iterations. In our technique, to account for light occlusion, we need to represent an environment illumination as a cube map. Therefore, we cannot use Debevec's segmentation algorithm [139]. A novel environment map wavelet sampling is proposed by Clarberg et al. [140]. The method redistributes many sample points after doing wavelet decomposition. However, for our rendering method we need a small number of directional lights as caustic computation is costly. Moreover, the sample point distribution depends on how we generate the initial random points (e.g. Uniform Random or Hammersley), and it also depends on the initial seed. For our caustic rendering using a few sampling, the result rendering will be quite different for every seed we use or every algorithm we use for generating the initial random points. Recently, the algorithm for segmenting environment cube map into several area light sources is proposed by Annen et al. [141]. For our purpose, we cannot use Annen et al.'s method [141] since on a cube map face that has almost homogenous radiance, the method only produces one large light region which is not suitable to be represented using only one directional light.

For rendering caustics under environment illumination, we propose the following environment cube map segmentation algorithm for computing important light regions. We assume that the importance of a light region is determined by the total radiance of the region.

Given an environment cube map and the number of intended light regions q, our algorithm divides the environment cube map into q important non-overlapping rectangular light regions. The algorithm starts with six regions, with each corresponds to one face of the cube map. We prioritize segmenting the region having the most total radiance until we obtain q light regions. If the total radiance of a region is below a certain threshold (almost no radiance), then it will be discarded and will not be processed further. Afterwards, repeatedly select the region which has the most total radiance from all faces and subdivide that region into two new regions having the same total radiance. In the subdivision, we test segmenting the region horizontally and vertically, and then choose the one closer to a square shape. In the segmentation, we scan the region (up to down if horizontal segmentation, left to right if vertical segmentation), and compute the total pixel radiance of both sides. The scanning is stopped when the total pixel radiance of both sides are similar. The total pixel radiance can be computed faster by using Sum Area Table (SAT).

In order to choose either horizontal or vertical segmentation, we compute which segmentation yields two regions with the aspect ratios nearer to 1.0 (closer to a square). The aspect ratio is the ratio of the dimension with the bigger magnitude (max{width, height}) to the dimension with the smaller magnitude (min{width, height}) of a rectangular region. Regions with aspect ratio closer to 1.0 are preferable since we represent each important light region with a directional light source. For doing so, the region is segmented horizontally and vertically, and we compute their aspect ratio deviations. The deviation is computed as the sum of the squared deviation of the new two regions (Equation 3.2).

$$Dev = (A_1 - 1.0)^2 + (A_2 - 1.0)^2, \tag{3.2}$$



(a) Grace cathedral (b) The Uffizi Gallery (c) Synthetic map

Figure 3.7: Results of our environment cube map segmentation. We segment each cube map into 24 regions. The green lines are the region boundaries and the green dots are the light directions representing the regions. Note the brighter cube faces are subdivided more compared to the darker faces.

where Dev is the deviation, A_1 and A_2 are the aspect ratio of the two new regions from the subdivision.

We select the segmentation that yields a smaller Dev and continue with performing the segmentation until q light regions are obtained. For each light region (of selected q regions), we store the weighted centre (with the weight is the radiance of each pixel in the region) of the region as the important light direction along with the region boundary (i.e. the coordinates of the two opposite corners of the rectangular region). Figure 3.7 shows the results of our environment cube map segmentation algorithm.

Taking into account the light occlusion effect (details in Section 3.3.2), there are three approaches in computing the important light regions and their corresponding directional lights during the rendering. In the first approach, given an environment illumination, we recompute on-the-fly the directional light for each important light region based on the unoccluded pixels of the region and in the second approach we recompute the light regions (perform the environment cube map segmentation) on-the-fly. However, these two approaches suffer from temporal incoherence and some undesirable effects such as jittering and popping-in popping-out of caustic patterns.

In the third approach, the light regions and the directional lights are precomputed without considering occluders and we use this precomputed information in the rendering. By using this approach, parts of caustic patterns become dimmer or disappear in the presence of occluders without having the temporal incoherence problems. As such, we adopt this approach for segmenting the given environment illumination.

The segmentation algorithm is shown in Algorithm 3.3.

Algorithm 3.3: Environment cube map segmentation.

1 Generate six regions with each is one of the cube map faces;
   /* *total_regions* = 6. */
2 Discard regions whose total radiance are below threshold;
   /* *total_regions* becomes $\leq$ 6 after discarding. */
   /* Next, do the iterative segmentation until we obtain q regions. */
3 **while** *total_regions* $< q$ **do**
4 Select the region with the most total radiance;
5 Compute Dev (Equation 3.2) for each segmentation direction;
6 Choose segmentation direction (horizontal or vertical) with the lower Dev and segment the selected region into two regions;
7 Discard regions whose total radiance are below threshold;

3.3.2 Directional Light Radiance Sampling Taking Into Account Occlusion

As the directional lights are derived from an environment cube map, the radiance of each directional light is the total radiance of all unoccluded pixels in the important light region (corresponding to the directional light). To compute the total radiance, we render the important light region taking into account light occlusion (by rendering the occluders as black colour) into a texture (or a 2D image) and sum up the radiance of all pixels in the import-

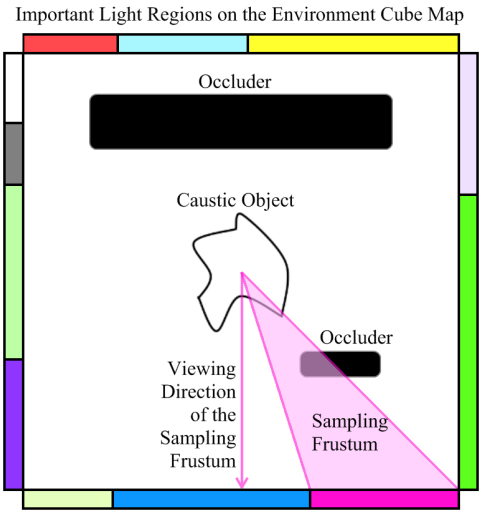
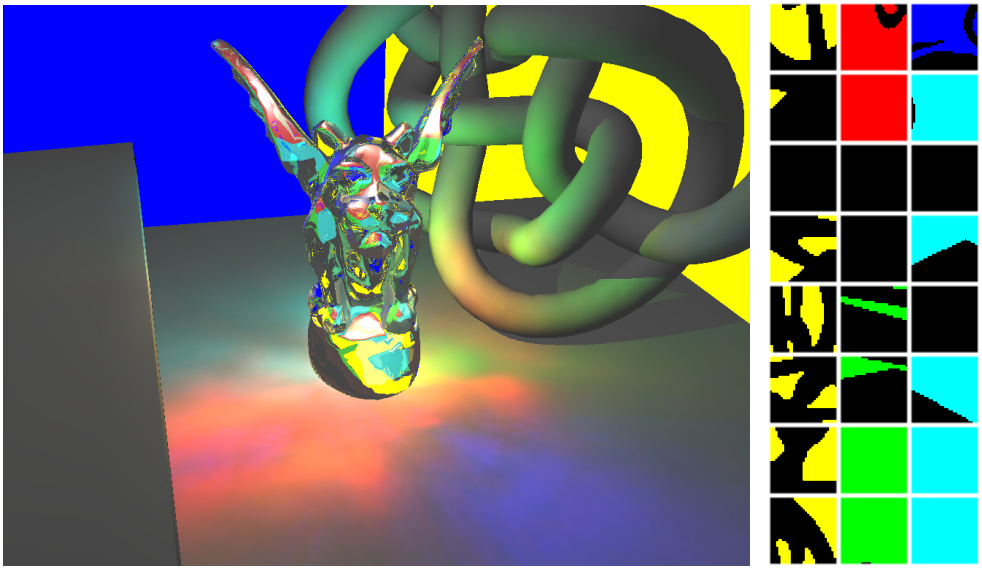


Figure 3.8: We sample the important light regions by rendering them. The sampling frustum for the rendering is based on the boundary information of the light region with the viewing direction is the direction to the environment cube map face containing that particular region. The occluders are rendered as black objects.

ant light region. This computation uses the information of each important light region and its corresponding light direction computed using our environment cube map segmentation algorithm.

Since we represent the environment illumination as a cube map, it is easy to set up a sampling frustum (a frustum is a 3D rendering region with the shape of a truncated pyramid in which all visible 3D objects are inside this region) for rendering the important light regions. The sampling frustum uses the boundary information of the important light regions (obtained from our environment cube map segmentation algorithm). The coordinates of the two opposite corners of the rectangular important light region are used to specify the left, right, bottom, and top boundaries of the sampling frustum. The direction of the sampling frustum is the direction from the centre of the caustic object to the face of the environment cube map where the important light region lies. Figure 3.8 illustrates how we sample the important light and Figure 3.9 shows an example of an environment map sampling.

The occlusion handling under environment illumination algorithm is presented in Algorithm 3.4.



(a) (b)

Figure 3.9: Environment map sampling. (a) shows the scene configuration with the environment map shown in Figure 3.8. (b) shows the environment map sampling by rendering 24 light regions. Note the occluders are rendered as black objects.

Algorithm 3.4: Occlusion handling.

1 **foreach** *light direction* L **do**

2 Render the segmented region (output of Algorithm 3) corresponding to L;

3 Render the occluders as black colour;

4 Compute the total intensity of the rendered image, which serves as the weight for L. (The weight is used in Line 5 of Algorithm 3.2);

3.3.3 Caustic Computation

Since we approximate the environment illumination as a set of directional lights, we compute the final caustics by integrating the caustic patterns of all directional lights (with each is weighted by the total radiance of the sampled important light region of that particular light direction).

3.4 GPU Implementation

We present the GPU implementation of our caustic rendering under q directional lights. We use OpenGL and Cg in the implementation. At the time of this research, we focus on GPUs which support at least OpenGL 2.1.

There are three visual effects in the rendering, namely refraction effect, caustics cast on diffuse surfaces, and volumetric caustics. We use the image-based method proposed by Wyman et al. [106] to render the refraction effect of the caustic object. They approximate the intersection points where the viewing rays exit the caustic object by using the precomputed distance between two points on the caustic object.

The caustic rendering consists of three major stages: storing the precomputed caustic spheres into memory, computing the directional light radiance, and computing caustic intensity. Storing the caustic spheres is done only once when we load the caustic object to the scene. We recompute the light radiance and caustics only if there are changes in the scene (the caustic object, surrounding objects, and/or environment light are transformed).

3.4.1 Storing Caustic Spheres

We store the precomputed caustic spheres (caustic images) of each light direction in a 3D texture, so that we can directly use the trilinear interpolation provided internally by the graphics API when we sample the caustic spheres. We test two alternatives to store the precomputed caustic spheres of all light directions, multiple 3D textures and a single 3D texture.

Multiple 3D textures The first method is to use several 3D textures and each 3D texture stores the caustic spheres of one light direction. By storing the precomputed caustic spheres of each direction in a 3D texture, we need to do multiple rendering passes (q passes) in order to accumulate the caustic patterns from all light directions.

Single 3D texture The second alternative is to store the caustic spheres of all light directions into **a single 3D texture** by tiling. As the result, we are able to accumulate the caustic

patterns from all q light directions either in single pass or multiple passes.

3.4.2 Computing Directional Light Radiance

The caustic intensity of each light direction depends on light radiance from that particular direction arriving at the caustic object. We can estimate the radiance in the presence of occluders by using the technique described in Section 3.3.

To account for occlusion, we render the segmented regions of the environment map from the centre of the caustic object to a texture array, with each slice in the texture array corresponds to one segmented region. The rendering is done to a low resolution texture array, i.e. $32\times32\times q$. Afterwards, we generate low resolution textures for each slice in the texture array by using the mipmapping [142]. Thus, we can sample the average radiance of a light region reaching the caustic object by sampling the topmost level of the mipmap. Since the radiance value that we obtain from the mipmapping is the average value, we multiply it with the number of pixels in that light region in the original environment cube map in order to get the total radiance.

3.4.3 Caustic Sampling

There are two ways for computing the caustic intensity from all directional lights (which we obtain from environment cube map segmentation). First, for each light we sample the four nearest precomputed caustic spheres directly (**direct sampling**). Second, we accumulate the caustic spheres of all lights beforehand (**compiled**) into compiled caustic spheres and then we sample these compiled caustic spheres in the rendering.

Direct sampling For each light, and for every point in the scene where caustic intensity need to be computed, we sample the caustic intensities from the caustic spheres of the four nearest precomputed light directions and interpolate them. This method is straightforward, however, for every point in the scene we need to do the same operations (e.g. rotating the caustic spheres of the four nearest light directions for every point in the scene). This

Algorithm 3.5: Algorithm of compiled method.

```
1 Initialize the compiled 3D texture, set all texel values to zero;
2 foreach directional light L_i of all q directional lights do
3     Choose four nearest predefined light directions;
4     Rotate their caustic spheres and blend them;
      /* Note the caustic spheres are in 3D texture
         latitude-longitude representation                    */
5     Weight the blended caustic spheres with power of L_i;
6     Accumulate the weighted blended caustic sphere to the compiled 3D texture;
```

becomes a bottleneck in volumetric caustic rendering since we need to do this process for every sample point on the cast ray and it greatly decreases the performance as we need to sample many points on the cast ray, and for every point we need to iterate through all q light directions.

Compiled In the compiled method, to avoid the redundant operations as the ones in the direct sampling, we accumulate the caustic spheres from all light directions into a compiled single 3D texture beforehand. For every light direction L_i $(i = 1, ..., q)$, we rotate and blend the caustic spheres of the four nearest light directions (Figure 3.5), multiply the blended caustic spheres with the radiance of L_i, and accumulate them into the compiled 3D texture. In the rendering, whether casting the caustics or rendering volumetric caustics, we just need to sample the compiled 3D texture. We show the algorithm for the compiled method in Algorithm 3.5

3.4.4 Rendering Passes

We can perform the rendering in either multiple passes or single passes.

Multiple passes The iteration is done in CPU, with each iteration corresponding to one directional light. In every iteration, we pass the information of one directional light such as the four nearest precomputed light directions and their rotation matrices to the fragment shader. Using this technique, we can store the precomputed caustic patterns in the GPU in either multiple 3D textures or a single 3D texture.

Single pass The iterations for all directional lights are performed in GPU. We store the necessary information computed in CPU in the Texture Buffer Object because of its efficient data storing and retrieval capability in GPU. Since Texture Buffer Object is basically a 1D array, we store the information of the lights as a series of blocks of 16×4 float numbers (for each light, 1 block for the four nearest precomputed light indices in the 3D texture, 1 block for the weights, 1 block for the light intensities, 12 blocks for the rotation matrices and 1 unused block). In the shader, we just need to loop through all the q light directions, sample the information from the Texture Buffer Object, rotate and blend the caustic patterns, and accumulate them. In this technique, we can store the precomputed caustic spheres in GPU only in a single 3D texture.

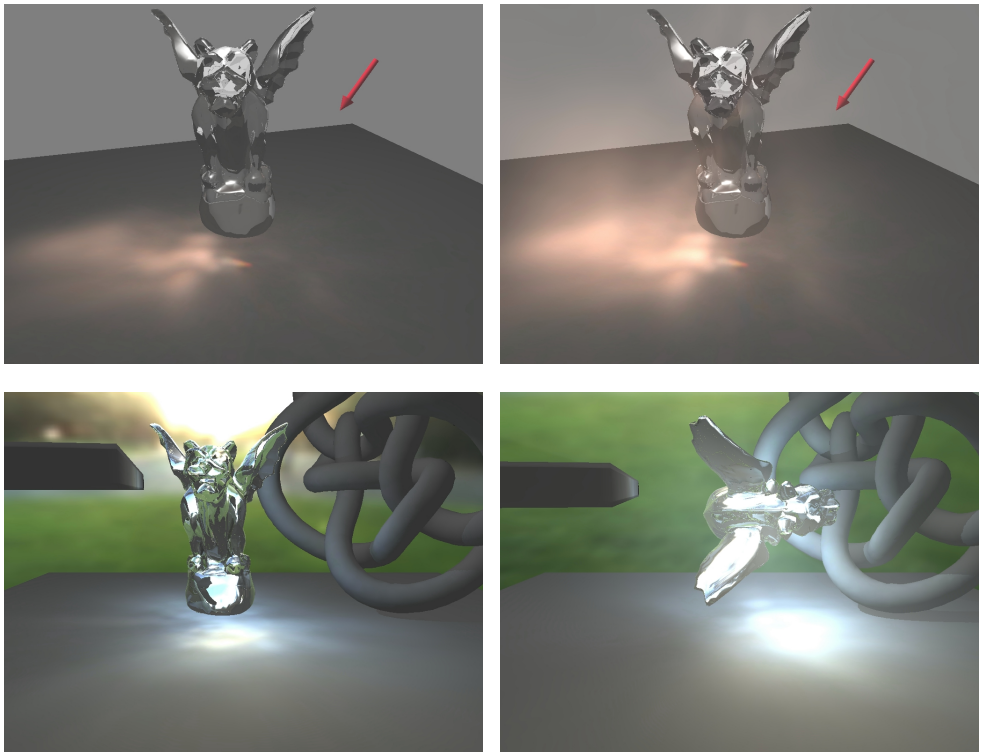
3.5 Results

We performed the experiments on a PC with an Intel Core i7 2.67 GHz and an Nvidia GTX 285. The image size of our real-time rendering results are 1024×768 pixels. We provide visual comparisons, quantitative analysis, and rendering speed performance for several experiment results.

The scenes we used in our experiments consist of around 5K vertices/10K triangles to around 50K vertices/115K triangles as the GPU we used is able to handle real-time rendering of scenes up to several hundred thousands of polygons. The scenes consist of one caustic object and up to three diffuse objects. Basically, the effect of the number of diffuse objects is negligible as each object is a group of polygons.

3.5.1 Rendering Results using Our Technique

Figure 3.10 shows our rendering results under one directional light ($q = 1$ and $s = 16$) and environment illumination ($q = 24$ and $s = 16$). Since the caustics under environment illumination are computed from many light directions, they are blurred. From our experiments, the visual differences of the caustics between $q = \{32, 48\}$ and $q = 24$ (number of



(a) Only cast caustics (b) With volumetric caustics

Figure 3.10: Caustics under one directional light source with the light direction is denoted by the arrow (top row) and caustics under environment illumination (bottom row). (a) shows caustics cast to the scene and (b) shows the result with volumetric caustics.

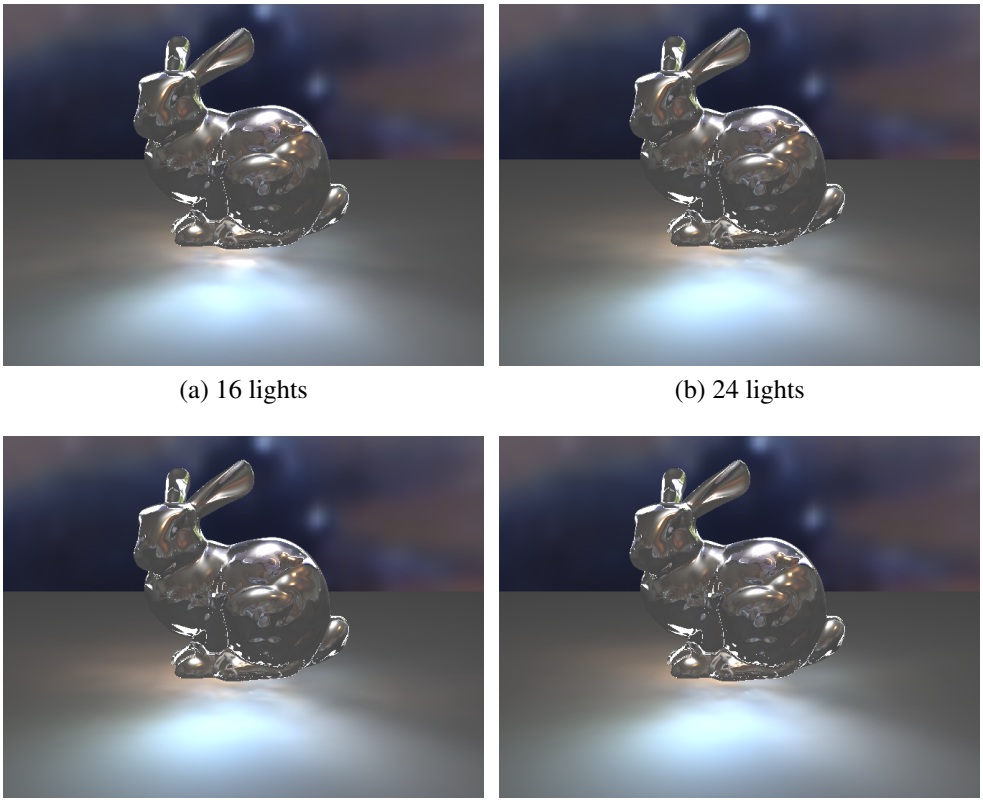
directional lights) were hardly noticeable.

For justification on the suggested number of light sources, we show the rendering results for various number of light sources in Figure 3.11 and the image differences in Figure 3.12. Note that we show the differences by using colour mapping, which ranges from blue (less differences) to red (more differences), as we show in Figure 3.12e.

We performed quantitative analysis by comparing images generated using our techniques and an image generated using mental ray. The rendered images are the images of the scene in Figure 3.14 (consists of around 40K vertices and 115K triangles). We implemented the widely used Root Mean Square Error (RMSE) and Information Content Weighted Structural Similarity Index (IW-SSIM) [143]. When we compute the RMSE, we firstly convert the images from RGB representation to CIELAB space as the euclidean distance between two colours in CIELAB space approximately corresponds to the difference accord-

ing to human perception. The IW-SSIM computations are done in grayscale representation which are converted directly from RGB representation, as suggested by its author in his website [144]. When we compute the errors, we only consider the pixels on the floor as we focus our analysis on the generated cast caustics. Moreover, due to the different renderer used in renderings, the difference in the pixels belonging to the caustic object and the environment will affect the analysis. Hence, we discard the pixels belonging to the environment illumination and the caustic object.

We would also like to note that quantitative analysis of image quality is still an ongoing and active research, as it is still challenging to accurately model the human perceptual evaluation. For example, Average RMSE is pixel-based and does not consider the structures or features in the images, IW-SSIM improves it by using a multi-resolution evaluation and takes into account some existing perceptual models. However, IW-SSIM only considers



(a) 16 lights (b) 24 lights

(c) 32 lights (d) 48 lights

Figure 3.11: Experiment with various number of light sources

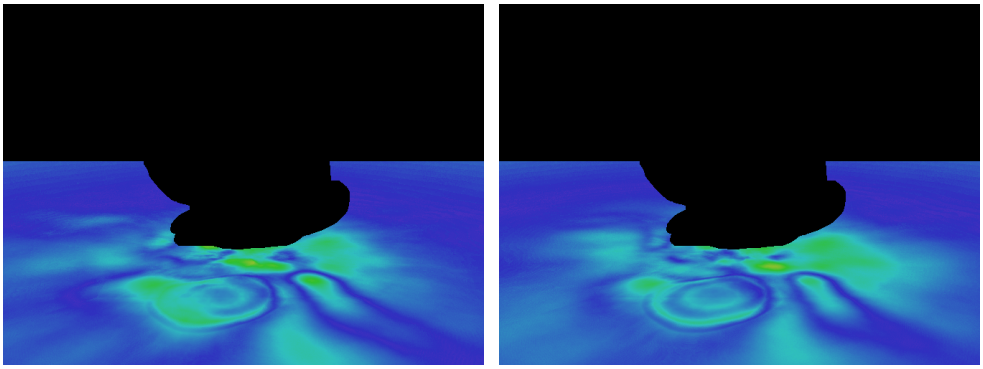
luminance factor and does not consider colour variation in the images. As such, image quality analysis is only one of the considerations in judging image quality and it is still not perfectly robust yet.

Table 3.1 and Figure 3.13 show that the average RMSE has a tendency to decrease and the IW-SSIM has a tendency to increase as we increase the number of light sources. Because our proposed method is an approximation and clustering technique, there might be some noise in the quantitative result, i.e. the RMSE does not always monotonically decrease and the IW-SSIM does not always monotonically increase. For example, we can see the average RMSE increases for 16 light sources and the IW-SSIM similarity is decrease for 48 light sources. Maximum RMSE shows a more consistent result by having the maximum RMSE monotonically decrease up to until 56 light sources.

Table 3.1: Quantitative results for our various experiments with various number of light sources for the scene shown in Figure 3.14. RMSE is Root Mean Square Error, lower values mean lesser difference. IW-SSIM is Index similarity, higher values mean greater similarity.

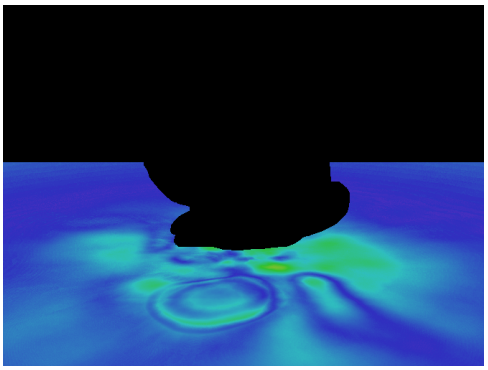
Experiments	RMSE		IW-SSIM
	Average	Maximum	Index
Quadratic $s = 16$ $q = 8$	26.633	154.863	0.8915
Quadratic $s = 16$ $q = 16$	19.710	142.986	0.8990
Quadratic $s = 16$ $q = 24$	21.500	141.429	0.9093
Quadratic $s = 16$ $q = 32$	20.950	135.046	0.9153
Quadratic $s = 16$ $q = 40$	20.905	130.513	0.9155
Quadratic $s = 16$ $q = 48$	20.386	122.740	0.9144
Quadratic $s = 16$ $q = 56$	20.511	113.535	0.9172
Quadratic $s = 16$ $q = 64$	20.597	113.776	0.9175

Moreover, under any number of light directions, the visual differences of $s = 16$ and $s = 32$ (number of caustic spheres) were also not very apparent. The justification is shown in Figures 3.14 and 3.15. From the histogram plot in Figure 3.16, we can see that the frequency of L2 difference for Quadratic s = 16 and s = 32 are already reduced ahead of other caustic sphere combinations at around L2 difference of 60. As rendering by using fewer caustic spheres consumes less resources, we suggest rendering with $q = 24$ and $s = 16$.

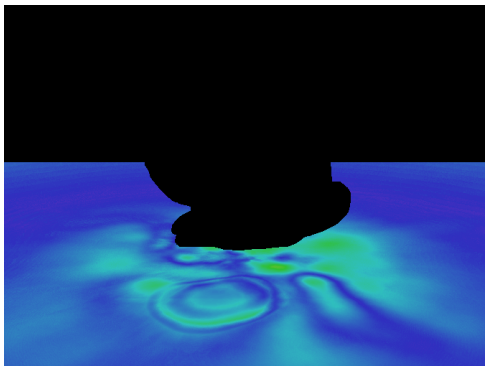


(a) 16 lights

(b) 24 lights



(c) 32 lights



(d) 48 lights

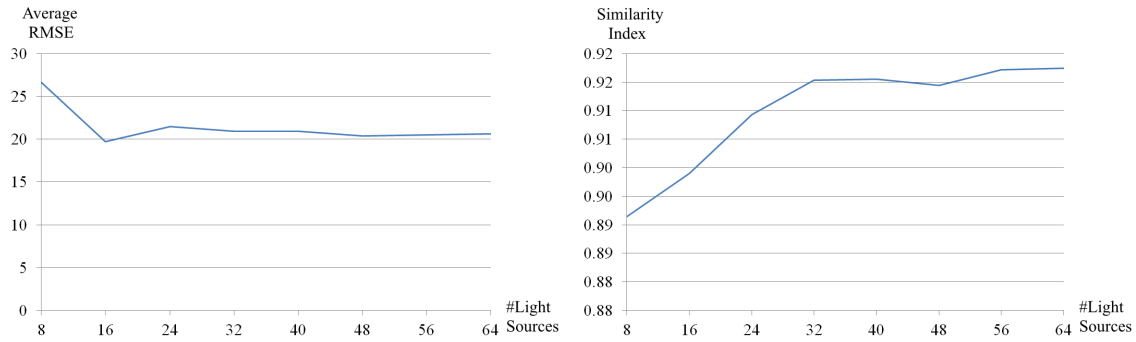
(e) Difference colour mapping

Figure 3.12: (a - d) Image difference of rendering with various number of light sources. (e) Differences colour mapping that we use for various image differences in this chapter, from the lowest difference colour on the left end (blue) to the highest difference colour on the right end

3.5.2 Comparisons With Uniform Radii Caustic Spheres

Figure 3.14 shows the comparison of caustic rendering using our quadratic radii caustic spheres and uniform radii caustic spheres proposed by Wyman et al. [103].

As seen in Figure 3.14, the quadratic radii with fewer caustic spheres achieves similar visual results as the uniform radii which uses more caustic spheres. This is because light attenuates and therefore it is sufficient to densely sample near the caustic object and sparsely sample far from the caustic object. With few caustic spheres ($s = 16$) circular banding artefact is visible near the caustic object in the rendering results using uniform radii caustic



(a) Plot of average RMSE

(b) Plot of similarity index IW-SSIM

Figure 3.13: Quantitative measurement plots of our results with various number of light sources.

Table 3.2: Quantitative rendering results for our various experiments for the scene shown in Figure 3.14. RMSE is Root Mean Square Error, lower values mean lesser difference. IW-SSIM is Index similarity, higher values mean greater similarity.

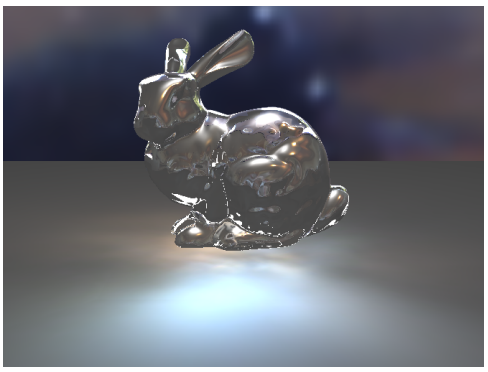
Experiments	RMSE		IW-SSIM
	Average	Maximum	Index
Quadratic $s = 8\ q = 24$	21.707	173.991	0.9055
Quadratic $s = 16\ q = 24$	21.500	141.429	0.9093
Quadratic $s = 32\ q = 24$	22.449	164.174	0.9052
Uniform $s = 8\ q = 24$	26.699	210.858	0.8881
Uniform $s = 16\ q = 24$	24.203	146.541	0.9001
Uniform $s = 32\ q = 24$	23.882	155.682	0.8968

spheres. The zoom-in view of the artefact is shown in Figure 3.14d (Note that we slightly adjust the zoomed-in view colour for the sake of presentation). It happens due to the insufficient number of caustic spheres near the caustic object thus the difference of the bright caustic patterns between caustic spheres is very conspicuous. This artefact does not exist in the rendering using quadratic radii since we sample densely near the caustic object, that is the spacing between caustic spheres near the caustic object is small. Note that the artefact also does not exist in the locations far from the caustic object due to the weak caustic patterns in those locations.

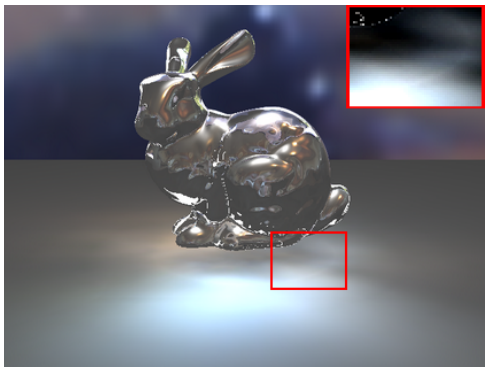
We also compare the images generated using quadratic radii and uniform radii with mental ray result and we show the image differences between Figures 3.14a - 3.14e and Figure 3.14f, with the brighter pixels means larger difference (Figure 3.15). The image

(a) Quadratic radii ($s = 8$)

(b) Quadratic radii ($s = 16$)



(c) Quadratic radii ($s = 32$)

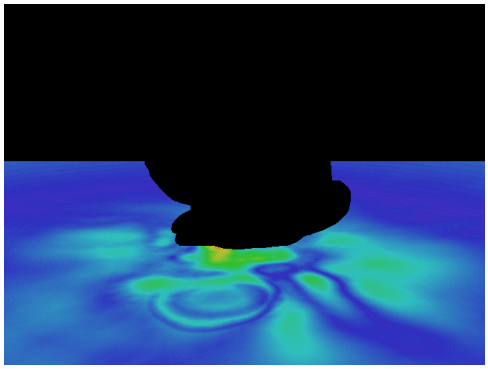


(d) Uniform radii ($s = 16$)

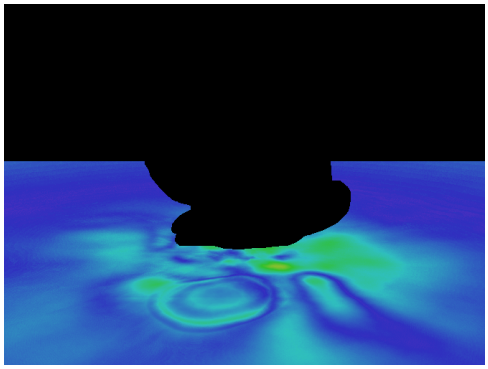
(e) Uniform radii ($s = 32$)

(f) mental ray

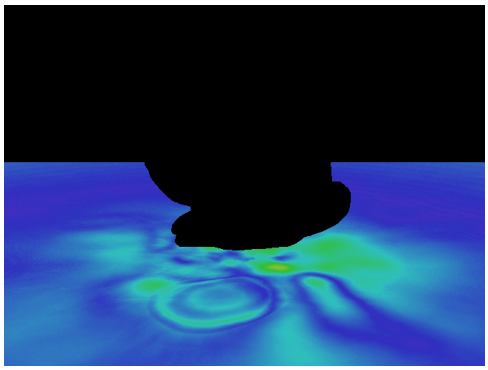
Figure 3.14: Rendering comparisons of the images generated using quadratic radii caustic spheres ((a) to (c)), uniform radii caustic spheres ((d) and (e)), and mental ray (f). We rendered (a) - (e) using 24 directional lights.



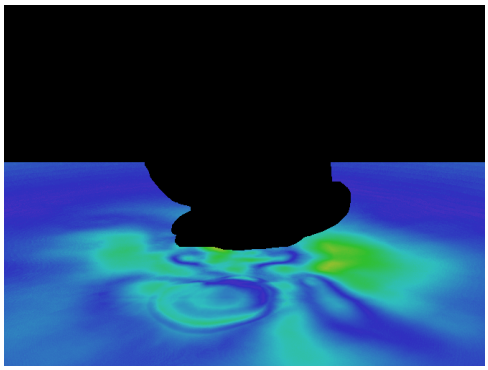
(a) Quadratic radii ($s = 8$)



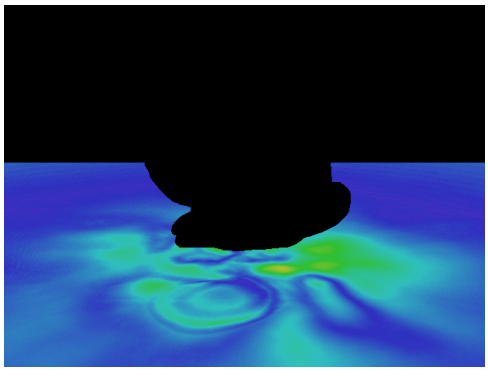
(b) Quadratic radii ($s = 16$)



(c) Quadratic radii ($s = 32$)



(d) Uniform radii ($s = 16$)



(e) Uniform radii ($s = 32$)

Figure 3.15: Image differences between images generated using various quadratic and uniform caustic sphere radii and mental ray in Figure 3.14.

differences are computed by using pixel-wise Euclidean distance.

As seen in Figure 3.15, image differences rendered using quadratic radii with $s = 16$ caustic spheres and uniform radii with $s = 32$ exhibit less difference compared to the images rendered using quadratic radii with $s = 8$ caustic spheres and uniform radii with $s = 16$ caustic spheres. Moreover, image difference rendered using quadratic radii with

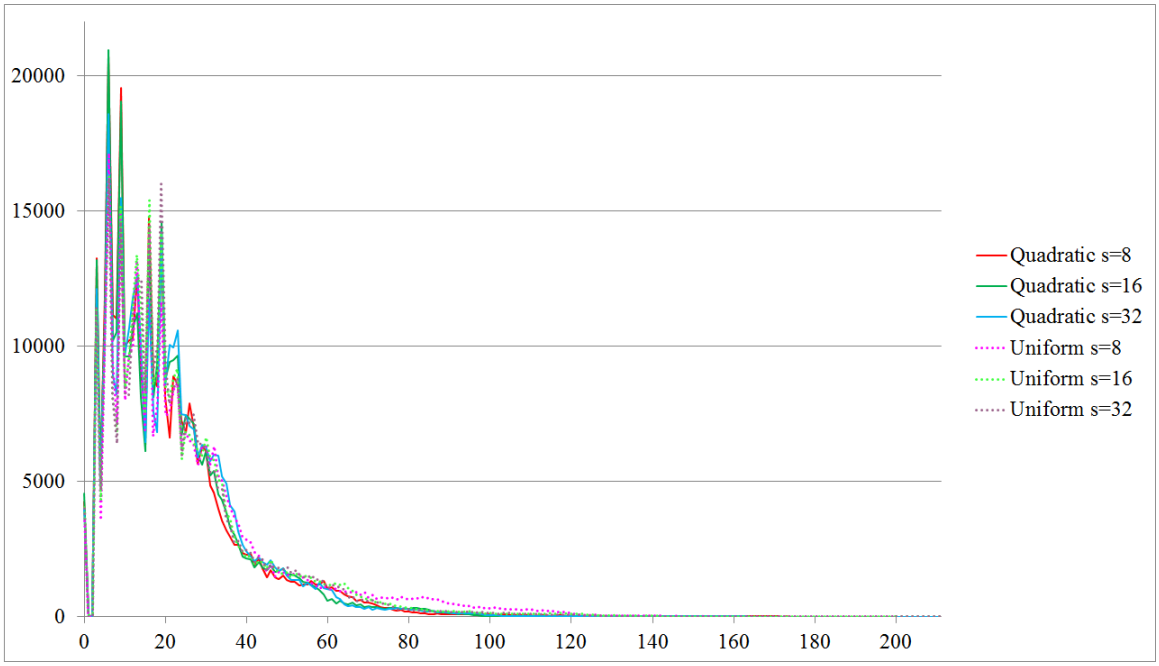
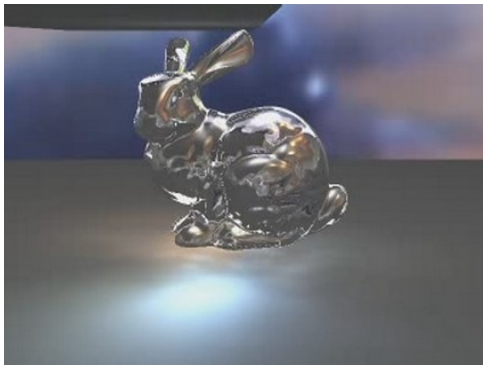


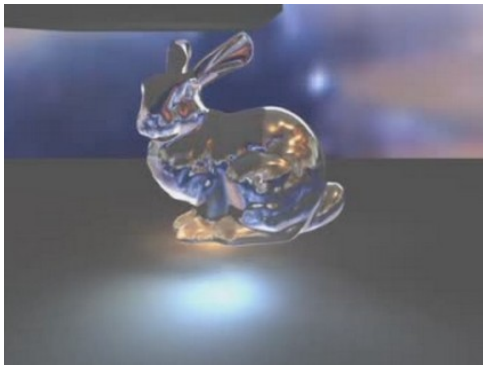
Figure 3.16: Histogram of RMSE pixel differences between our rendering results with various caustic spheres, quadratic and uniform, with $s = \{8, 13, 32\}$ and mental ray results for the scene shown in Figure 3.11. Horizontal axis is the L2 difference bin and vertical axis is the frequency of the pixel. Hence, higher frequencies on the left side mean the image has overall less differences, and higher frequencies on the right side mean the image has overall greater differences.

$s = 16$ has slightly less difference compared to the image difference rendered using uniform radii with $s = 32$. However, the cost difference shown in Table 3.2 shows that the image degrades for rendering with 32 quadratic caustic spheres. Moreover, it shows that its image quality is the same as the rendering with 8 quadratic caustic spheres. However, from the visual observation we can see that the rendering with 32 quadratic caustic spheres has same visual quality with 16 quadratic caustic spheres. This shows that quantitative image quality assessment still has room for improvement.

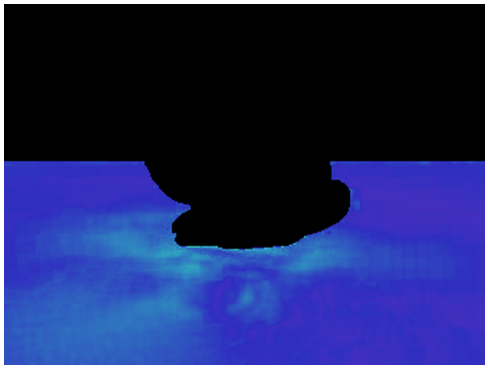
As for the case with light occlusion, we also show the comparison between our results and the results generated using mental ray in Figure 3.17 and we can see that our results are visually similar to the mental ray results. For example, due to the presence of the occluder above and slightly to the left of the bunny, we can see in both our and mental ray results, the caustics at the right hand side of the bunny disappear (please compare with the result



(a) Quadratic radii ($s = 16$)



(b) mental ray result



(c) Image difference

Figure 3.17: Occlusion comparison between the image generated using our method (a) and reference image generated using mental ray (b). We scaled the pixel differences in order to provide a better view.

without occlusion in Figure 3.14) as the light from the top left is blocked by the occluder. We show more rendering comparisons between the images rendered using our technique with the images rendered using mental ray in Figure 3.18. Note that we can observe some differences in the refraction effect of the caustic object because we implemented an approximate refraction algorithm proposed by Wyman [105]. As for the rendering performance, we were able to render the caustics in real-time while mental ray took about four minutes to render one image.

3.5.3 Performance Comparison

We performed experiments using the combinations of all possible options of the rendering techniques described in Section 3.4 and combinations of various numbers of directional

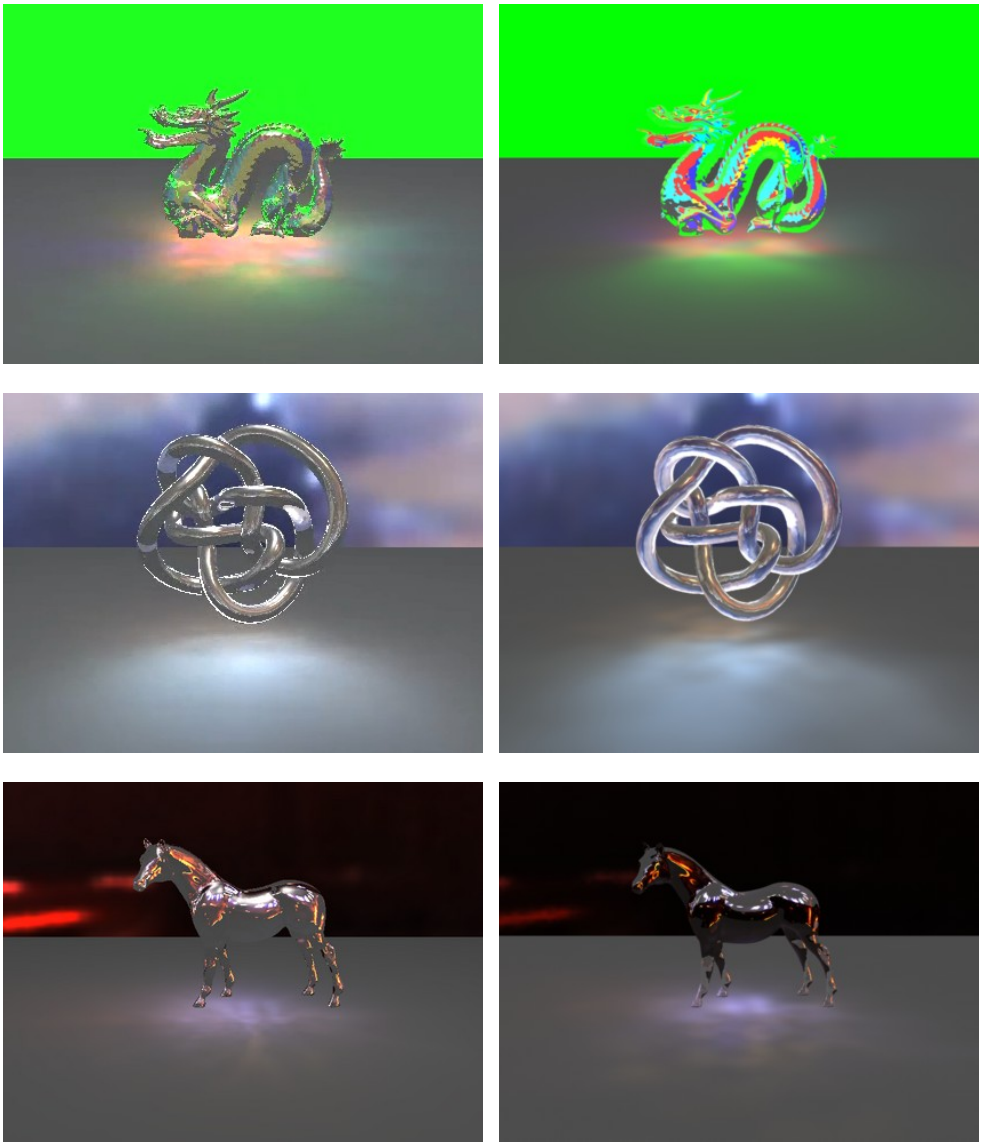


Figure 3.18: Rendering comparisons between the images generated using our technique (left column) and mental ray (right column).

lights $q = \{1, 16, 24, 32, 48\}$ and caustic spheres $s = \{8, 16, 32\}$ to determine the best rendering performance (in average frames per second, fps). We show the performance comparisons of rendering cast caustics in Table 3.3. In general, we achieve the best rendering performance by using the combination of multiple passes rendering, multiple 3D textures storage for the caustic spheres, and the compiled technique. As for the volumetric caustic rendering, the performance ratios between the alternatives have similar tendency as the performance ratios in the cast caustic rendering. For the volumetric caustic rendering

Table 3.3: Performance comparisons of various caustic rendering techniques for cast caustic rendering (in average frames per second, fps)(Note: #Dirs = number of directions, D=direct, C=compiled).

#Dirs.	#Spheres	Multiple Passes				Single Pass	
		Multiple Textures		Single Texture		Single Texture	
		D	C	D	C	D	C
1	8	142.3	173.1	140.3	165.4	120.0	176.7
	16	142.7	156.5	135.4	151.4	119.6	148.9
	32	142.6	129.9	139.6	124.8	120.0	120.9
16	8	38.2	60.0	31.2	60.2	21.4	52.8
	16	37.4	38.8	31.2	39.7	21.2	35.1
	32	37.2	24.0	31.2	24.1	21.2	20.5
24	8	27.6	45.0	22.9	45.1	14.9	41.0
	16	27.7	30.0	22.7	29.9	14.9	25.0
	32	27.7	17.4	22.8	17.5	14.9	14.2
32	8	22.0	37.6	17.8	37.4	11.5	32.7
	16	22.2	23.5	17.8	23.7	11.6	19.7
	32	22.2	13.5	17.8	13.5	11.4	11.0
48	8	15.5	27.0	12.4	27.5	7.9	23.2
	16	15.5	16.4	12.4	16.7	7.9	13.1
	32	15.5	9.6	12.5	9.3	7.8	7.5

with multiple passes rendering, multiple 3D textures storage for the caustic spheres, and the compiled technique, we were able to achieve 14.6 frames per second.

We also show plot frames per second measurement of cast caustic rendering for $q = \{8, 16, 24, 32, 40, 48, 56, 64\}$ with $s = \{16\}$ in Figure 3.19, with the rendering technique option of compiled multiple textures. Even though for each experiment we add the same number of parallel lights, the rendering performance does not drop linearly. Instead, it slowly drops to around 10 frames per second for the experiment with GPU Nvidia GTX 285 and around 40 frames per second for the GPU Nvidia GTX 690.

3.5.4 Limitations of Our Technique

Our technique can produce approximate caustics under environment illumination in real-time, however there are some limitations. First, our technique requires a large amount of memory to store the caustic spheres. Second, since we perform precomputation, the

caustic objects cannot be deformed during the rendering. However, many caustic objects in real-world such as glass objects are actually non-deformable. Therefore, we can apply our method to render caustics of such objects.

3.6 Summary

We have presented a GPU-based technique for real-time rendering of caustics and volumetric caustics in dynamic scenes under multiple directional lights. Our technique can be applied to render approximate caustics under environment illumination taking into account light occlusion from the surrounding objects.

Our proposed technique precomputes caustic patterns of several predefined light directions and we interpolate them in the rendering. In order to render caustic patterns under environment illumination, we perform environment cube map segmentation and obtain several directional lights which represent the environment illumination. Our work can be extended to handle volumetric caustics. We compare our rendering results with the mental ray rendering results and we show that our rendering results are close to the mental ray results.

One of the main aspects in our proposed method is the storage of caustic patterns. We store the precomputed caustic patterns in 3D texture memory in GPU. The sampling is effi-

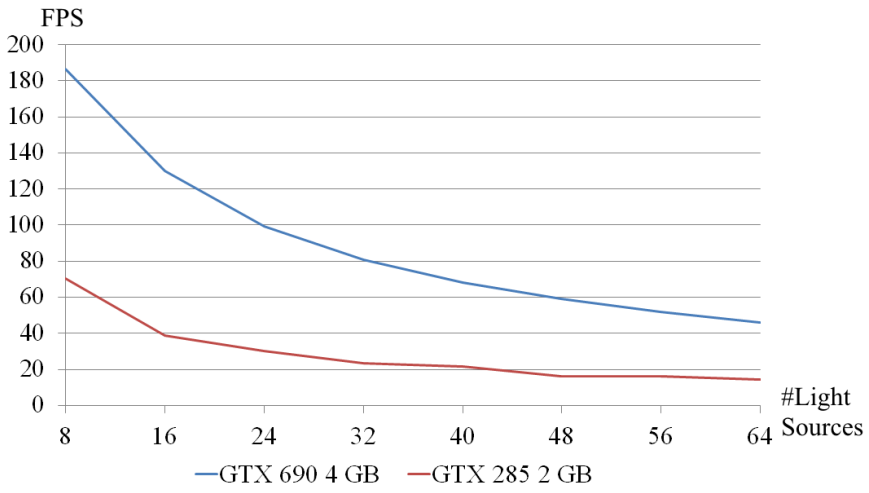


Figure 3.19: Plot of Frames Per Second (FPS) measurement with various number of light sources.

cient as we sample the 3D texture in shader and we obtain the trilinearly interpolated result. The storage can be improved by compressing the caustic patterns such as by using spherical harmonics. As the representation of precomputed caustic patterns is a set of constant values, instead of storing them in 3D textures, they can be stored in an alternate storage such as Uniform Buffer Object (UBO) or Shader Storage Buffer Object (SSBO) that is available in the recent GPU architectures. Moreover, by using CUDA and OpenGL Interoperability feature, we can precompute and/or compress the caustic patterns on-the-fly and pass them in the said buffer object to OpenGL for rendering.

As our work involves precomputation of caustic patterns, it currently only supports non-deformable caustic object. One possible approach to handle deformable caustic object is by precomputing caustic patterns of several deformation states and interpolating them in the rendering. Another possible approach is analysing the deformation and updating the photon map accordingly.

Finally, our technique can be extended to support caustic rendering under several local area lights, since each area light can also be approximated with fewer number of lights by using our segmentation technique (Section 3.3.1). However, care must be taken with respect to the relative position and orientation between the area lights and the caustic object.

Chapter 4

Spectral Caustic Rendering

In our work, we aim to accelerate spectral caustic rendering while maintaining its rendering quality. Specifically, we render spectral caustics effects on a surface due to the light splitting event (over the spectral domain) when the light encounters a caustic object. The caustic object is assumed to be homogeneous and non-scattering (for the subsurface scattering rendering work, readers are advised to check related literatures such as the work by Frisvad et al. for heavily scattering materials [145] and the work by Gutierrez et al. for thinly scattering materials such as underwater sea [146]). Our work supports scenes that have one or more caustic objects that have the same index of refraction profile. One of the main applications of our work is to assist caustic object design and visualization. The designer can design the caustic object (e.g. a gemstone, crystal vase, glass bracelet, crystal pendant, glass chandelier, etc.) and use our method to render it to observe the refraction and caustics effects, which is similar to the application of Guy and Soler's work [147].

We propose a two-step acceleration scheme for spectral caustic rendering. The first step of the acceleration scheme takes into account caustic object characteristic, i.e. the index of refraction over the wavelength $\eta(\lambda)$. Our basic idea is that the refraction angles of some wavelengths can be similar (within a certain threshold). Thus, we cluster those wavelengths into a single cluster and represent them as a single light ray in the rendering. In the second step, we take into account the Spectral Power Distribution (SPD) of the light source, the

surface reflectance in the scene, and the eye's sensitivities toward light stimulus of each visible wavelength [148] in order to determine the importance level of each wavelength cluster obtained from the first step. Moreover, we also take into account a geometrical factor, i.e. the surface area of each material where caustic patterns might be generated. We then adaptively allocate resources for rendering each wavelength cluster based on its importance level. By doing this, we cut down the rendering time of wavelength clusters that contribute less to the final rendered image.

We perform the rendering using the Stochastic Progressive Photon Mapping (SPPM) technique [7]. The main reason for choosing SPPM is its iterative refinement nature which allows us to adaptively adjust the number of iterations for rendering each wavelength cluster. Another important reason is that SPPM does not require large memory footprint compared to the traditional photon mapping technique [133].

We would like to mention that our acceleration scheme itself is not a rendering method, but an acceleration method that is applied to a spectral caustic rendering algorithm based on several considerations (index of refraction, light power, etc.). Hence, after applying our method to the SPPM (which is actually a biased and consistent method [6]), the rendering method will still be a biased method.

From the experimental results, our scheme can accelerate the rendering significantly while preserving its rendering quality. We compare the rendering results using our acceleration scheme against the Brute Force rendering result in Section 4.2.

4.1 Proposed Spectral Caustic Rendering

The basic idea of our spectral caustic rendering is as follows. We first cluster the visible wavelengths, based on their refractive behaviours with respect to the caustic object, into several clusters (first acceleration step, Section 4.1.1). Then, for each cluster, we determine its refinement amount, based on the perceptual level and geometrical factor of surrounding surfaces, and decide the number of refinement iterations during the rendering (second

acceleration step, Section 4.1.2). We then perform spectral caustic rendering using the Stochastic Progressive Photon Mapping (SPPM) technique [7] (please refer to the beginning of this chapter for the reasons of choosing SPPM). SPPM rendering algorithm contains many iterations compared to the traditional photon mapping rendering algorithm. In each iteration, a set of photons are emitted from the light source and stored on surfaces. Afterward, we reshoot camera rays through every pixel. The camera ray passing through each pixel is computed by sampling a random position on a pixel. Based on the shot camera ray, we sample the visible surface in the scene. For each visible point on the surface, we perform photon gathering and update its statistical irradiance information (e.g. number of gathered photons, accumulated irradiance). In SPPM, we assume all local areas hit by all camera rays passing through the same pixel share the same statistical irradiance information. We then discard the emitted photons, and in the next iteration we redo the photon emission and camera ray emission, and update the statistical irradiance information.

Integrating our proposed approach to SPPM is straightforward. Before the rendering starts, we do preprocessing which consists of computing the clusters (first acceleration step), and number of refinements (second acceleration step). For each cluster, we perform a separate SPPM rendering. The photons and camera rays consist of information of each wavelength (such as power) of the current processed cluster. Since we only consider caustic objects during the clustering, in each rendering iteration, each photon will store the same amount of information and will perform a similar amount of workload. This factor is beneficial to GPU which favours coherent workload between threads. When a photon hits a surface, we multiply the photon power with the reflectance of the surface for all the wavelengths in the cluster. On the other hand, when a photon hits a caustic object, we refract the photon based on the average index of refraction of the wavelengths in the current cluster. The number of iterations for the SPPM rendering of the current cluster is the result from the second acceleration step. After we have finished rendering all clusters, we then accumulate them into a single image.

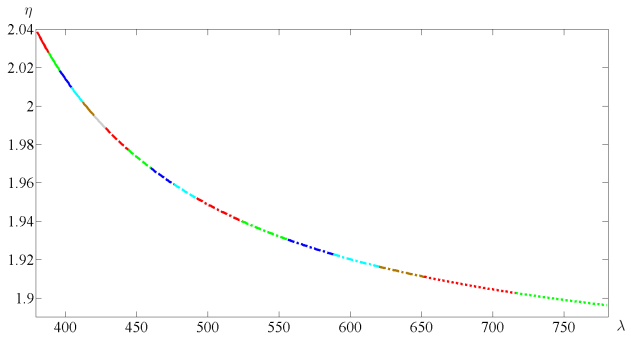
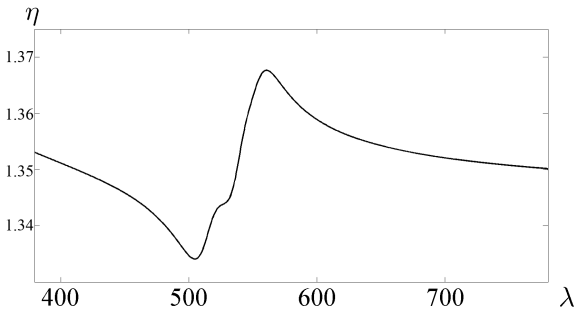


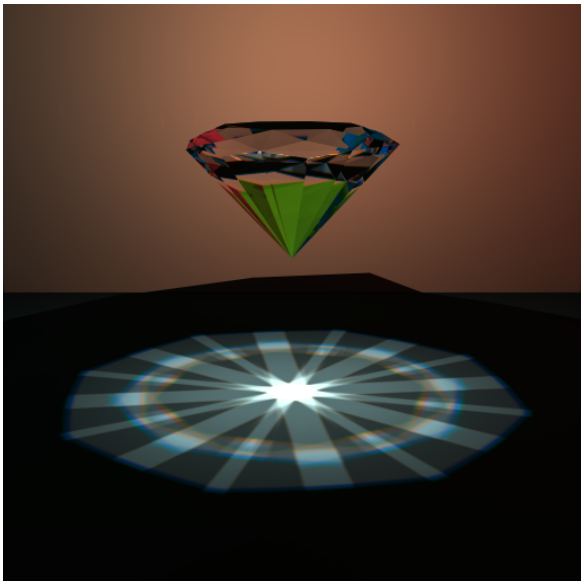
Figure 4.1: Plot of IOR of a real-world material N-SF66, with the x-axis represents wavelengths λ (in nm) and the y-axis represents IOR η. We also show the clustering results, with each cluster denoted by a colour line. Note that in the areas where the changes of IOR are rapid, the clusters have less number of wavelengths (smaller clusters). For visualization purpose, we use a higher threshold value in this plot, i.e. 0.1, and the total number of clusters is 17.

4.1.1 First Acceleration Step

The input to our wavelength clustering is the Index of Refraction (IOR) curve of the caustic object. This information can be obtained from a real-world material or can be designed by users. In general, the IORs of real-world materials are monotonously decreasing with respect to wavelengths. Figure 4.1 shows an example of IOR values (y-axis) of a real-world material N-SF66 [149] with respect to wavelengths (x-axis). This information is obtained from the Schott Data Sheet [149]. We also take into account materials with non-monotonously decreasing IOR, such as real-world materials which have absorption properties [150] (we show an example of IOR and its rendering result in Figure 4.2) or artificial materials designed by users (Figures 4.3a and 4.3b). For these materials, the curve can be of any shapes. Thus, we first sort the wavelengths with decreasing IORs before we perform clustering. This is important since some wavelengths (that are not next to each other) might refract the incoming light to the same direction. For instance, $\eta = 2$ in Figure 4.3a intersects the IOR curve five times. Hence, by sorting the wavelengths based on their IORs before clustering, we can cluster non-neighbouring wavelengths with similar refraction behaviours together and as a result we can obtain less number of clusters. For example, if we cluster the artificial IOR curves used in our experiments (shown in Figure 4.3) without sorting, we ob-



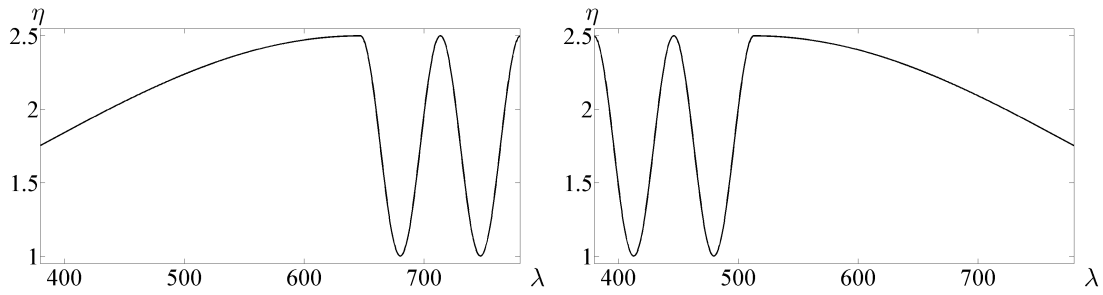
(a) IOR



(b) Rendering result

Figure 4.2: IOR and a rendering result of a diamond caustic object with Rose Bengal 10% solution material.

tain 247 clusters for the two curves. On the other hand, if we first sort the wavelengths, we obtain less number of clusters, 202 clusters for Figure 4.3a and 204 clusters for Figure 4.3b.



(a) Arbitrary 1

(b) Arbitrary 2

Figure 4.3: User's designed IOR curves used in our experiments. (a) Clustered into 202 clusters (b) Clustered into 204 clusters.

The clustering is done by firstly computing the absolute amount of change of refraction angle with respect to the change of wavelength $|d\theta/d\lambda|$ for all possible incoming angles for each 1 nm of visible wavelengths. Next, each wavelength is considered as an individual cluster, and then we group neighbouring wavelengths if their sums of change of refraction angles are below a threshold h.

Change of refraction angle It can be computed as follows. We start with Snell's law in Equation 4.1.

$$\eta_2(\lambda)\sin(\theta_2) = \eta_1(\lambda)\sin(\theta_1), \tag{4.1}$$

where η_1 is the IOR of air, η_2 the IOR of caustic object, θ_1 the incoming angle and θ_2 the refraction angle. Both η_1 and η_2 are functions of wavelength λ since we take into account varying IOR across the wavelengths. The equation is then rearranged since we are interested in computing the refraction angle θ_2. Hence, the refraction angle θ_2 is a function of the incoming angle θ_1 and wavelength λ. We assume that the IOR of air to be constant over the wavelength, i.e. 1.0. The rearranged equation is shown in Equation 4.2.

$$\theta_2(\theta_1, \lambda) = \sin^{-1}\left(\frac{\sin(\theta_1)}{\eta_2(\lambda)}\right). \tag{4.2}$$

We can then compute the derivative of Equation 4.2 with respect to the wavelength by using the chain rule and we obtain the result as shown in Equation 4.3.

$$\frac{d\theta_2(\lambda,\theta_1)}{d\lambda} = \frac{-\sin(\theta_1)\eta_2^{'}(\lambda)}{\sqrt{\eta_2(\lambda)^2(\eta_2(\lambda)^2 - \sin(\theta_1)^2)}}, \tag{4.3}$$

where $\eta_2^{'}(\lambda)$ is the first derivative of the caustic object's IOR with respect to wavelength. Since we have sorted the wavelengths beforehand (with decreasing IOR), $\frac{d\theta_2(\lambda,\theta_1)}{d\lambda}$ will always result in greater than zero values.

We show an example plot $d\theta_2(\lambda,\theta_1)/d\lambda$ of a real-world material N-SF66 [149] in Figure 4.4. The total amount of changes in refraction angle a wavelength range $\Theta(\lambda)$ is computed as the sum of changes for incoming angles (from 0° to 89°), as shown in Equation 4.4.

$$\Theta(\lambda) = \int_{0^\circ}^{89^\circ}\int_{\lambda}^{\lambda+1\text{ nm}} \frac{d\theta_2(\lambda,\theta_1)}{d\lambda} d\lambda d\theta_1. \tag{4.4}$$

Referring to the example in Figure 4.4, we compute the volume below the surface.

Volume computation To compute $\Theta(\lambda)$, we firstly subdivide the surface into patches with

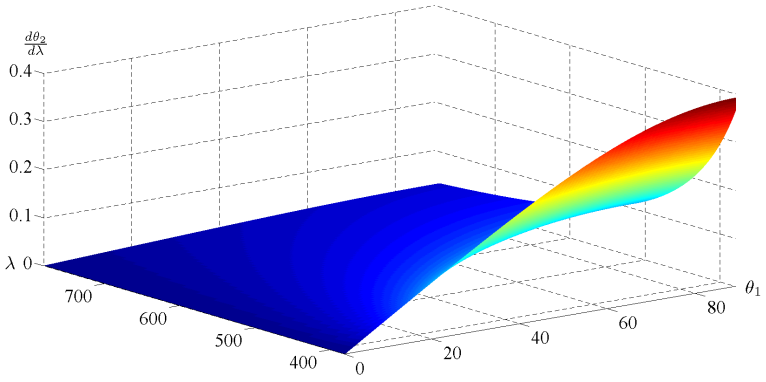


Figure 4.4: Example plot of $|d\theta_2(\lambda, \theta_1)/d\lambda|$ (in degree/nm) for a real-world material N-SF66. The θ_1-axis represents the incoming angle (in degree) and the λ-axis represents wavelength (in nm).

each patch covers 1 nm by 1 degree and then we compute the values inside a patch by using bicubic interpolation. We use bicubic interpolation because we intend to achieve a smooth interpolation of the surface as we observe the IOR curves are smooth to begin with. Hence, it is not really suitable to use a simple one such as bilinear interpolation, which can produce peaks in the corners of each cell in the grid. The $\Theta(\lambda)$ can be computed by performing an integration over the bicubic patches from 0 degree to 89 degrees. In detail, the computation is as follows.

For the sake of clarity, we firstly assume $\psi(\lambda, \theta_1)$ to be a point in one of the patches of $d\theta_2(\lambda, \theta_1)/d\lambda$ surface. To interpolate the patch, we define the bicubic interpolation (in λ and θ_1 directions).

$$\psi(\lambda, \theta_1) = \sum_{j=0}^{3} \sum_{i=0}^{3} a_{i,j} \lambda^i \theta_1^j, \tag{4.5}$$

where each a_{ij} is the coefficient of each term in the equation. To compute the integration, each of the coefficient a_{ij} has to be computed first. In order to achieve this, we have to define the $\psi(\lambda, \theta_1)$ values on the four corners (λ, θ_1), $(\lambda{+}1 \text{ nm}, \theta_1)$, $(\lambda, \theta_1{+}1°)$, $(\lambda{+}1 \text{ nm}, \theta_1{+}1°)$, including its first and second derivatives (Equation 4.6). Since the dimension of every patch in λ and θ_1 axes are the same, in order to simplify the computation, we assume one corner of every patch to be at $(0 \text{ nm}, 0°)$ (for clearer reading, we omit the units in Equation 4.6).

$$
\begin{aligned}
\psi(0,0) &= a_{0,0}, \\
\psi(1,0) &= a_{0,0} + a_{1,0} + a_{2,0} + a_{3,0}, \\
\psi(0,1) &= a_{0,0} + a_{0,1} + a_{0,2} + a_{0,3}, \\
\psi(1,1) &= \sum_{j=0}^{3}\sum_{i=0}^{3} a_{i,j}, \\
\psi_{\lambda}(0,0) &= a_{1,0}, \\
\psi_{\lambda}(1,0) &= a_{1,0} + 2a_{2,0} + 3a_{3,0}, \\
\psi_{\lambda}(0,1) &= a_{1,0} + a_{1,1} + a_{1,2} + a_{1,3}, \\
\psi_{\lambda}(1,1) &= \sum_{j=0}^{3}\sum_{i=1}^{3} i\, a_{i,j}, \\
\psi_{\theta_1}(0,0) &= a_{0,1}, \\
\psi_{\theta_1}(1,0) &= a_{0,1} + a_{1,1} + a_{2,1} + a_{3,1}, \\
\psi_{\theta_1}(0,1) &= a_{0,1} + 2a_{0,2} + 3a_{0,3}, \\
\psi_{\theta_1}(1,1) &= \sum_{j=1}^{3}\sum_{i=0}^{3} j\, a_{i,j}, \\
\psi_{\lambda\theta_1}(0,0) &= a_{1,1}, \\
\psi_{\lambda\theta_1}(1,0) &= a_{1,1} + 2a_{2,1} + 3a_{3,1}, \\
\psi_{\lambda\theta_1}(0,1) &= a_{1,1} + +2a_{1,2} + 3a_{1,3}, \\
\psi_{\lambda\theta_1}(1,1) &= \sum_{j=1}^{3}\sum_{i=1}^{3} i\, j\, a_{i,j},
\end{aligned}
\tag{4.6}
$$

where ψ_{λ}, ψ_{θ_1}, are the first derivative of the patch corner with respect to λ and θ_1 directions respectively, $\psi_{\lambda\theta_1}$ is the second derivative of the patch corner with respect to both λ and θ_1 directions. It is necessary that Equation 4.3 is differentiable in order to perform the bicubic interpolation, and we show that Equation 4.3 is differentiable in Appendix A.

If we assume

$$\mathbf{\Psi} = \left| \begin{array}{ccccc} \psi(0,0) & \psi(\lambda,0) & \dots & \psi_{\lambda\theta_1}(0,\theta_1) & \psi_{\lambda\theta_1}(\lambda,\theta_1) \end{array} \right|^T, \tag{4.7}$$

$$\mathbf{a} = \left| \begin{array}{ccccc} a_{0,0} & a_{1,0} & \dots & a_{2,3} & a_{3,3} \end{array} \right|^T, \tag{4.8}$$

$\mathbf{M}$ is a matrix that contains numerical constants in Equation 4.6. Equation 4.6 can be reformulated as a system of linear equations :

$$\mathbf{\Psi} = \mathbf{M}\mathbf{a}. \tag{4.9}$$

Hence, each constant $a_{i,j}$ in $\mathbf{a}$ can be obtained by solving Equation 4.9 (i.e. $\mathbf{a} = \mathbf{M}^{-1}\mathbf{\Psi}$). $\mathbf{M}^{-1}$ can be precomputed as it is independent of $\psi(\lambda,\theta_1)$. We show the computed $\mathbf{M}^{-1}$ in Appendix A. After obtaining the constants, then we can compute the volume below each patch by using double integration in Equation 4.10.

$$v = \int_0^1 \int_0^1 \psi(\lambda,\theta_1)\, d\lambda d\theta_1 = \int_0^1 \int_0^1 \sum_{j=0}^{3}\sum_{i=0}^{3} a_{i,j}\, \lambda^i\, \theta_1^j\, d\lambda d\theta_1 \tag{4.10}$$

The computation of Equation 4.10 can be optimized by simple mathematics manipulation. Firstly, the sigma summations are moved to the outside:

$$v = \sum_{j=0}^{3}\sum_{i=0}^{3} \int_0^1 \int_0^1 a_{i,j}\, \lambda^i\, \theta_1^j\, d\lambda d\theta_1 \tag{4.11}$$

We can see that from Equation 4.11, $a_{i,j}$ is constant inside the double integral and it can be moved outside of the double integral. Moreover, the product of integration λ^i and θ_1^j is independent of $a_{i,j}$ (which in turn is independent of $\eta(\lambda)$). Hence, the product of integration

λ^i and θ_1^j can be precomputed. The final integration formula then becomes :

$$v = \sum_{j=0}^{3}\sum_{i=0}^{3} a_{i,j}\, b_{i,j} = \mathbf{a}^T\mathbf{b} \tag{4.12}$$

with

$$b_{i,j} = \int_0^1 \int_0^1 \lambda^i\, \theta_1^j\, d\lambda d\theta_1, \tag{4.13}$$

$$\mathbf{b} = \begin{vmatrix} b_{0,0} & b_{1,0} & \dots & b_{2,3} & b_{3,3} \end{vmatrix}^T. \tag{4.14}$$

We show the precomputed values of the vector $\mathbf{b}$ in Appendix A.

Since we discretize the surface into patches, the computation of $\Theta(\lambda)$ becomes the computation of volume covered by one column of p patches (i.e. $0^\circ - 89^\circ$).

Clustering We consider every 1 nm wavelength between 380 nm and 780 nm, that is, in total we have 401 wavelengths. We perform the clustering by iteratively merging two neighbouring wavelength clusters whose sum of their $\Theta(\lambda)$ is below a threshold h. The clustering process is shown in Algorithm 4.1. The threshold h is determined based on the maximum allowable change of refraction angle α with respect to the change of wavelength. Using Equation 4.4, the maximum allowable $\Theta(\lambda)$ for merging clusters is $h = \int\int \alpha\ d\lambda\ d\theta_1 = \alpha 89$. In practice, we set α to 0.05°. We determine this value from various experiments, and we show one of them in Figure 4.6. The scene is illuminated with CIE Standard Illuminant D65 and the IOR of the caustic object is Arbitrary 1 (Figure 4.3a). We use Arbitrary 1 IOR as an example due to its non-monotonous property that is suitable for testing our first acceleration step. Figure 4.1 shows the clustering results of a real-world material N-SF66.

One might argue to use a simpler clustering, i.e. cluster based on equal steps along the IOR axis. However, we would like to mention that similar IORs do not guarantee similar refraction direction, as refraction direction is not linear with respect to IOR. Assuming the incoming angle is 89 degree, the refracted angle for several IORs are :

- IOR 2.00 -> 50.95 degree
- IOR 1.99 -> 51.30 degree

- IOR 1.57 -> 81.49 degree
- IOR 1.56 -> 84.44 degree

As we can see here, even though the similarity between 2.00 and 1.99 IOR is the same as between 1.57 and 1.56 (the difference in IOR is 0.01), however, the refracted angles are different significantly (0.35 degree and 3.05 degree respectively). Note that Elek et al. [77] also uses a similar approach as ours.

4.1.2 Second Acceleration Step

Generally, we perceive every visible wavelength arriving at our retina based on the emitted Spectral Power Distribution (SPD) of the light source, the surface reflectance of materials in the scene, and our eyes' sensitivity for each visible wavelength. By taking into account these information, we propose an importance level $u(\lambda)$ that controls the number of rendering refinement for each wavelength cluster. That is, for clusters with lower $u(\lambda)$ values, we refine less times compared to clusters with higher $u(\lambda)$ values. We show the formula in the following Equation 4.15

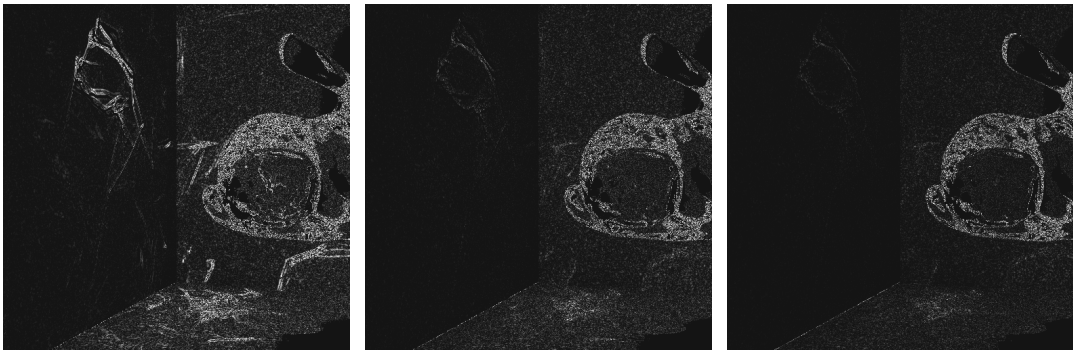
$$u(\lambda) = l(\lambda)y(\lambda)\sum_i m_i(\lambda)L_i\hat{g}_i, \tag{4.15}$$

where $l(\lambda)$ is the light's SPD with respect to wavelength λ (we can use a real-world light source [8] or user's designed), $y(\lambda)$ is the luminosity function, $m_i(\lambda)$ is the i-th material's surface reflectance with respect to wavelength λ (value in the range [0..1] (unit less) as it describes the portion of light that is reflected; we can use a real-world material such as the reflectance of Macbeth Colour Checker [8] or that is designed by a user). For non-diffuse surfaces which have varying reflectance amount over the possible reflected directions, we handle these surfaces by taking their maximum reflectance for m_i. By doing this, we might overestimate but will not underestimate the reflectance.

Algorithm 4.1: Clustering Algorithm

```
for each of the 401 wavelengths (380 nm to 780 nm) do
    if it's Θ(λ) ≥ h then
        Mark it as a final cluster;
    else
        Mark it as a free cluster;
    end
end
while there are free clusters do
    Find a pair of neighbouring free clusters whose sum of their Θ(λ) is the smallest;
    if the sum ≤ h then
        Merge the two clusters, assign the sum as the Θ(λ), and mark the new cluster
        as a free cluster;
    else
        Mark all the free clusters as final clusters;
    end
end
```

Figure 4.5: The overall scene configuration of the results shown on the first row of Figures 4.9 and 4.13. The light source is a point light source located near the face of the bunny.



(a) $0.5°$ (4 clusters) vs. $0.1°$ (17 clusters)

(b) $0.1°$ (17 clusters) vs. $0.05°$ (34 clusters)

(c) $0.05°$ (34 clusters) and $0.01°$ (137 clusters)

(d) $0.5°$ (58 clusters) vs. $0.1°$ (152 clusters)

(e) $0.1°$ (152 clusters) vs. $0.05°$ (204 clusters)

(f) $0.05°$ (204 clusters) and $0.01°$ (346 clusters)

(g) $0.5°$ (55 clusters) vs. $0.1°$ (147 clusters)

(h) $0.1°$ (147 clusters) vs. $0.05°$ (202 clusters)

(i) $0.05°$ (202 clusters) vs $0.01°$ (345 clusters)

Figure 4.6: Image differences between rendering results with varying threshold value, with brighter pixels indicating larger differences. Each cluster is rendered with 2000 iterations. For each pixel, we compute the Euclidean distance and we scale it by 10.0. The first, second, and third columns show image differences between $0.5°$ and $0.1°$, $0.1°$ and $0.05°$, $0.05°$ and $0.01°$ respectively. We use N-SF66 (Figure 4.1), Arbitrary 2 (Figure 4.3b), and Arbitrary 1 (Figure 4.3a) IOR curves on the caustic objects of the first, second, and third row respectively. As for the light source, we use CIE Standard illuminant F10 for the first row, and D65 the second and third rows. The scene configuration of the first row is shown in Figure 4.5. The light source of the scenes on the second row is a point light source directly above the caustic object and on the third row is a point light source directly below the caustic object.

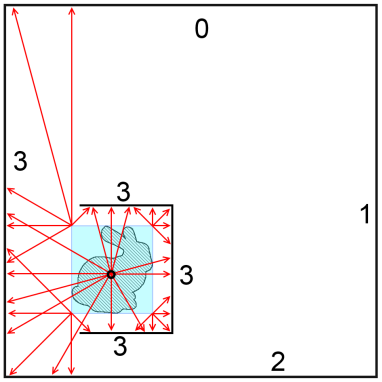


Figure 4.7: We sample the material types of the surrounding surfaces from the bounding box (blue box) corners and the centre of the caustic object. The numbers denote the material type. As shown in the figure, sampling of material #3 is higher as the smaller box enclosing the caustic object has material #3. Some parts of material #0 and #2 are sampled, and none of material #1 is sampled.

L_i is the weighting based on the lightness of the surface material m_i. The lightness is the same as the L^* of the CIE L*A*B* colour space. $\hat{g}_i$ is the weight based on the surface area of material m_i visible from the caustic object.

Perceptual factor affects the refinement as it is redundant to use the same amount of resources for rendering surfaces that have different levels of perceptual. We incorporate two perceptual factors in the $u(\lambda)$ computation :

- Luminosity function $y(\lambda)$ that describes the human eye sensitivity in the spectral domain (i.e. human eyes have different amounts of brightness sensation over the visible wavelength range). The luminosity function is the second CIE standard colour matching function ($y(\lambda)$).

- Lightness L_i, the non-linear perceptual property of the material.

Moreover, we also consider the geometrical distribution of the surface material. More weight is given to materials whose surfaces receive more caustic patterns. To compute the weight $\hat{g}_i$, we use spherical uniform sampling to sample the surrounding of caustic object. The weight $\hat{g}_i$ is computed as $\hat{g}_i = g_i / \sum_j g_j$ (normalized to range [0..1]) where g_i is the number of sample points on the surfaces with material m_i. The sampling is illustrated in Figure 4.7. We show our experiment on geometrical factor in Section 4.2.2.

A part of Equation 4.15 has theoretical ground. For instance, multiplication of light power and eyes luminosity $l(\lambda)y(\lambda)$. The luminosity function is from an established experiment measurement as explained before. The summation term is heuristic, as we weight the contribution of each material for a particular wavelength based on its lightness and its surface area hit by light photons. The greater the lightness is a material, the more conspicuous the material is. Therefore, we should give the material more weight. A similar idea is applied to the geometrical area of the material as we explained in the preceding paragraph

The $u(\lambda)$ of all visible wavelengths are then normalized such that the highest $u(\lambda)$ has the value of 1.0. Figure 4.8 shows some examples of $u(\lambda)$. For comparison, we show the $u(\lambda)$ proposed by Radziszewski et al. [43] which is scene independent (always fixed) in Figure 4.8a

Based on $u(\lambda)$, we have several options to set the importance level of a wavelength cluster. For the first option, we use the maximum $u(\lambda)$ of the wavelengths in the cluster. The drawback of this approach is that a cluster dominated by wavelengths with low $u(\lambda)$ but has few wavelengths with very high $u(\lambda)$ will have too many rendering iterations. Another option is to use the average $u(\lambda)$ of the wavelengths in the cluster. Although a cluster dominated by low $u(\lambda)$ but has a few wavelengths with very high $u(\lambda)$ might not have enough rendering iterations, in practice, we adopt the former approach in order not to underestimate the refinement iteration amount. The number of iterations for a wavelength cluster is then determined by multiplying its importance level with maximum number of iterations.

4.2 Experiments

We perform our experiments on a PC with an Intel i7-3820 3.60 GHz CPU and a Nvidia GTX 690 4 GB. The rendering implementation is done using OptiX, a GPU-based ray tracing engine [151].

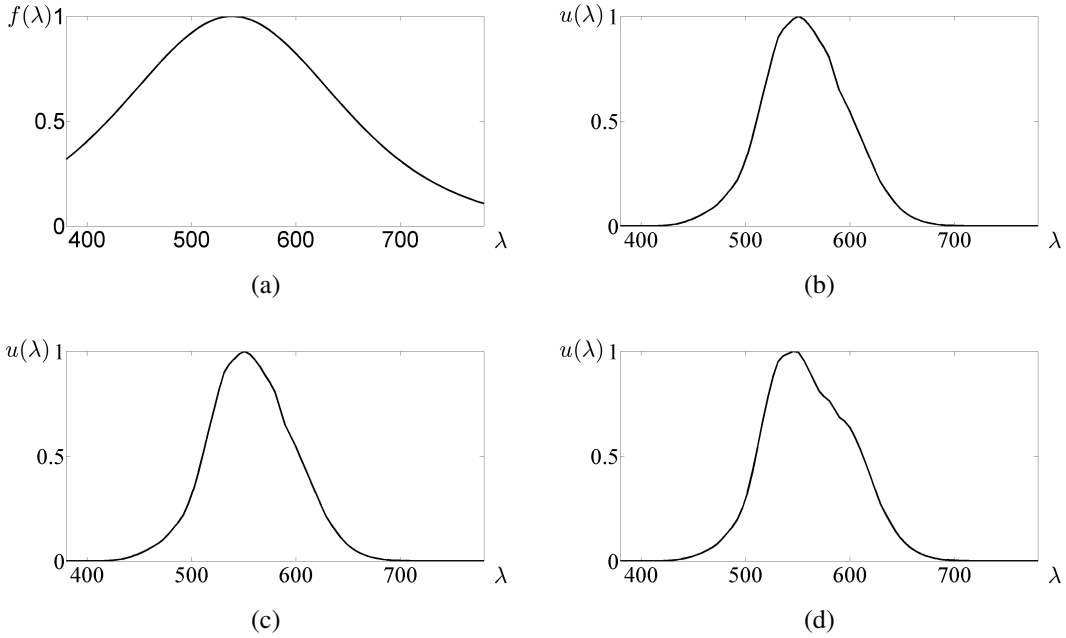


Figure 4.8: Plot of $u(\lambda)$ (y-axis) over the visible wavelengths λ (x-axis) for various scenes. (a) $u(\lambda)$ proposed by Radziszewski et al. It is fixed, independent of scenes. (b) Our $u(\lambda)$ for the scene in Figure 4.13b. (c) Our $u(\lambda)$ for the scene in Figure 4.13e. (d) Our $u(\lambda)$ for the scene in Figures 4.13h.

4.2.1 Brute Force Rendering

For the sake of fair comparisons between the Brute Force method and our method, we firstly perform experiments to find appropriate parameters for the Brute Force experiments. In this case, we try to determine maximum number of iterations and wavelength nanometre step. We typically set maximum number of iterations to 2000 as we can achieve good rendering quality while keeping the rendering time manageable, especially in performing Brute Force rendering (i.e. we render 2000 iterations for each 1 nm of visible wavelengths). As we show in Figure 4.9, the image difference between 2000 and 3000 iterations is pretty low.

In the Brute Force rendering whose results are shown in Figure 4.9, we use 1 nm step as we try to be as conservative as possible in order not to miss any signal variation that can contribute to the final rendering. In order to determine a good nanometre step of the wavelength in the Brute Force computation (and at the same time to provide fair comparison with our proposed acceleration scheme), we perform the following experiments. We render

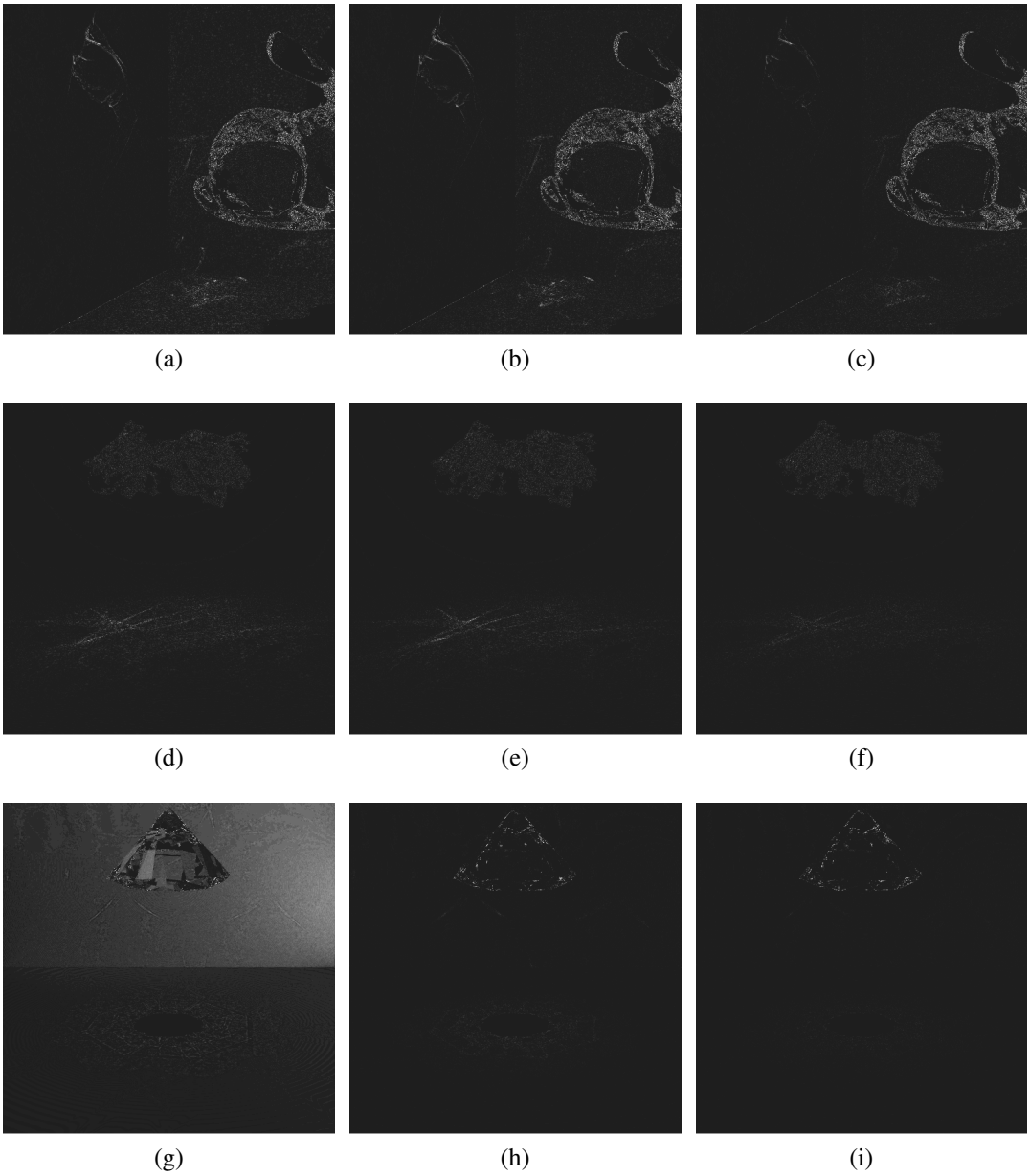
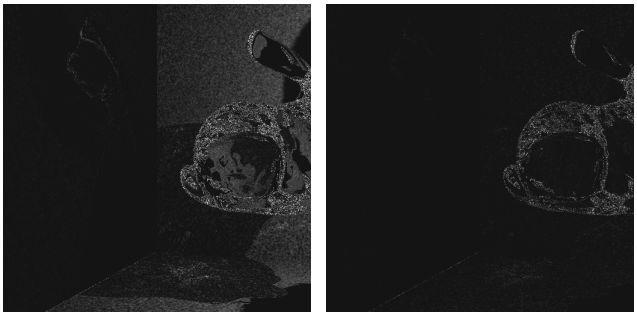


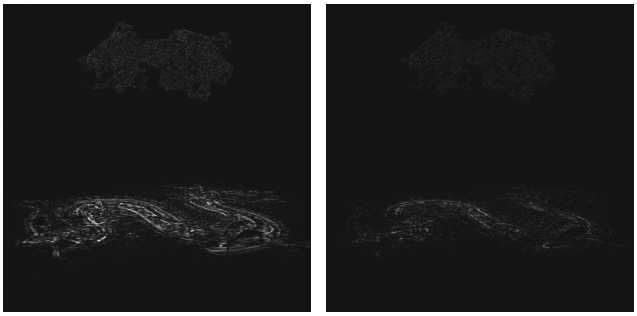
Figure 4.9: Image differences between Brute Force rendering results with varying iterations (in 1 nm step), with brighter pixels indicating larger differences. For each pixel, we compute the Euclidean distance and we scale it by 10.0. The first, second, and third columns show image difference between 500 and 1000 iterations, 1000 and 2000 iterations, 2000 and 3000 iterations respectively. The scene configuration of each row corresponds to the scene configuration of each row of Figure 4.6 or 4.13.

scenes with 401 clusters (each covers 1 nm), 200 clusters (each covers around 2 nm), and 100 clusters (each covers around 4 nm). We show the image difference results (between 100 and 200 clusters, 200 and 401 clusters) in Figure 4.10 for two scenes. As shown in



(a) Image difference between 100 and 200 clusters

(b) Image difference between 200 and 401 clusters



(c) Image difference between 100 and 200 clusters

(d) Image difference between 200 and 401 clusters

Figure 4.10: Image differences between Brute Force rendering results with varying nanometre steps, with brighter pixels indicating larger differences. For each pixel, we compute the Euclidean distance and we scale it by 10.0. (a,b) The same scene as the scene in the first row of Figure 4.9, with N-SF66 IOR (Figure 4.1) and F10 light source. (c,d) The same scene as the scene in the second row of Figure 4.9, with Arbitrary 2 IOR (Figure 4.3b) and D65 light source.

the results, the differences are smaller between 200 and 401 clusters. Moreover, due to the peaks in F10 SPD, the sampling in 100 clusters (every around 4 nm) is not sufficient as it affects the illumination of the whole scene (Figure 4.10a). There are still noticeable differences between 200 and 401 clusters in Figure 4.10d. Hence, in the following comparison experiments, we use 401 clusters (1 nm sampling step) for the Brute Force rendering.

4.2.2 Geometrical Factor Experiment

We show the comparison between considering and not considering the geometrical factor (i.e. for not considering the geometrical factor, the g_i is always 1.0) in Figure 4.11. The scene setting is similar to the scene shown in Figure 4.5, but with additional smaller blue

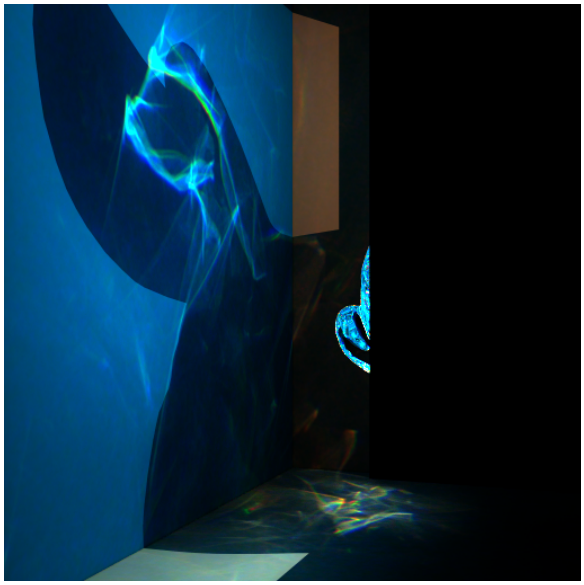


Figure 4.11: Geometrical factor experiment result. We show the rendering result of experiment with geometrical factor as the caustic patterns can be rendered in the same quality as the rendering result without geometrical factor. The rendering with geometrical factor took 15.61 minutes and the rendering without geometrical factor took 16.18 minutes.

walls (they have the same material as the left wall) enclosing the bunny. The only open side is the left side facing the blue wall, such that the caustic patterns can land on the left wall and on the floor. The scene arrangement is illustrated in Figure 4.7. As seen in the results, by considering the geometrical factor, the rendering speed is slightly improved compared to not considering the geometrical factor.

4.2.3 Comparisons

Experimental results of our methods are shown in Figure 4.13 with the Index of Refraction (IOR) curves used in our experiments shown in Figures 4.1 and 4.3. We also show image difference between the Brute Force results and results using our methods and Radziszewski et al.'s in Figure 4.14. All images are rendered with a resolution of 512 x 512. We show the comparisons between three spectral rendering methods.

The first method (results are shown on the first column) is the Brute Force rendering that performs SPPM rendering for each 1 nm of visible wavelengths (with iterations for each 1 nm wavelength). The second method (results are shown on the second column), is the

combination of our first and second acceleration steps. The third method (results are shown on the third column) uses our first acceleration step to cluster the wavelengths and then uses Radziszewski et al.'s probability distribution function $u(\lambda)$ to determine the number of iterations for each wavelength cluster [43]. The purpose of this comparison is to show the effectiveness of our second acceleration step.

Compared to the rendering time by using the Brute Force method, the combination of our first and second acceleration steps can significantly accelerate the rendering (especially for the scene on the first row with a real-world material N-SF66 in which we can cluster the visible wavelengths to just 34 clusters; this shows the effectiveness of our first acceleration step) while preserving the image quality. Even though the rendering using our second acceleration step show a bit more differences compared to the rendering using Radziszewski et al.'s probability distribution function in image differences in Figure 4.14, however, the differences are hard to discern in the original rendering results. Moreover, by using our second acceleration step, we can perform the rendering faster compared to using Radziszewski et al.'s $u(\lambda)$.

Overall, our acceleration scheme achieves higher acceleration on caustics object that have a smaller range of index of refraction (such as natural material, N-SF66) since we can obtain less number of clusters (34 clusters for N-SF66).

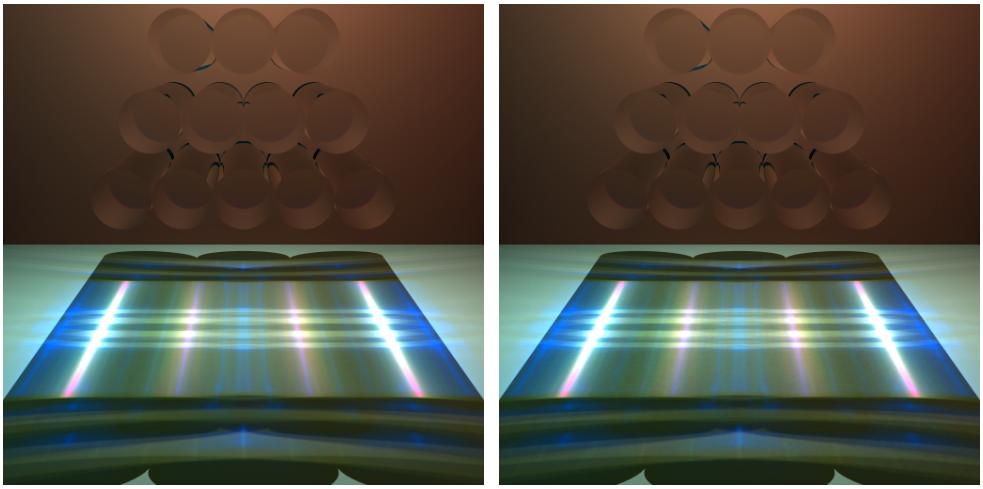
4.2.4 Discussions

The main concern of our proposed method is that the clustering in our first acceleration step only takes into account the first refraction. The refracted clustered light might diverge further on its subsequent reflections and/or refractions. For instance, in the second intersection event, some wavelengths in the cluster should be reflected due to total internal reflection and some others should be refracted (exit the caustic object). However, in our acceleration scheme, we do not split the cluster further. We perform experiments to investigate this issue. We render caustics caused by a stack of cylinders (which has Arbitrary 2 IOR) and

D65 light source. Due to the placement of caustic objects, we expect the light to undergo several intersections. We show the rendering result in Figure 4.12, where only very subtle differences in caustic patterns are observed. The same phenomenon is also observed for the diamond caustic object as shown in Figures 4.13g and 4.13h. The light source is positioned between the floor and the diamond. As a result, the caustic patterns we see on the floor are generated from the lights that undergo several total internal reflections inside the diamond caustic object.

From our experiments, including the ones with materials that have a wide range of IOR, we can see that it is sufficient to consider only the first refraction in the wavelength clustering step. It is sufficient because the light ray energy decreases if it undergoes multiple reflections and/or refractions and thus it will not affect the final caustics much. Hence, in practice, especially spectral caustic rendering of real-world caustic objects that have a smaller range of IOR, our wavelength clustering is sufficient to produce good results.

As our method depends on the input threshold for clustering, it will produce a biased result if the input threshold value is not appropriate (e.g. if the input threshold value is too high, it will result in a very aggressive clustering). Similarly, for the second acceleration step, if the maximum iteration is set too low, it will result in a very early termination, which may produce a very biased result. There are some possibilities in improving our method to be approaching unbiased characteristics. For instance, for the first acceleration step, we could iteratively refine the threshold in the clustering computation (i.e. analyse the directional and positional changes of the lights exiting the caustic object as we are reducing the threshold, until the changes are reasonably low). For the second acceleration step, in each iteration we could randomly choose which wavelength cluster to render based on the importance level of each wavelength cluster. However, this method requires extensive bookkeeping of each wavelength cluster information (such as the updated shared accumulated flux of SPPM for each region in the scene) as we may process the wavelength cluster in the future iterations.



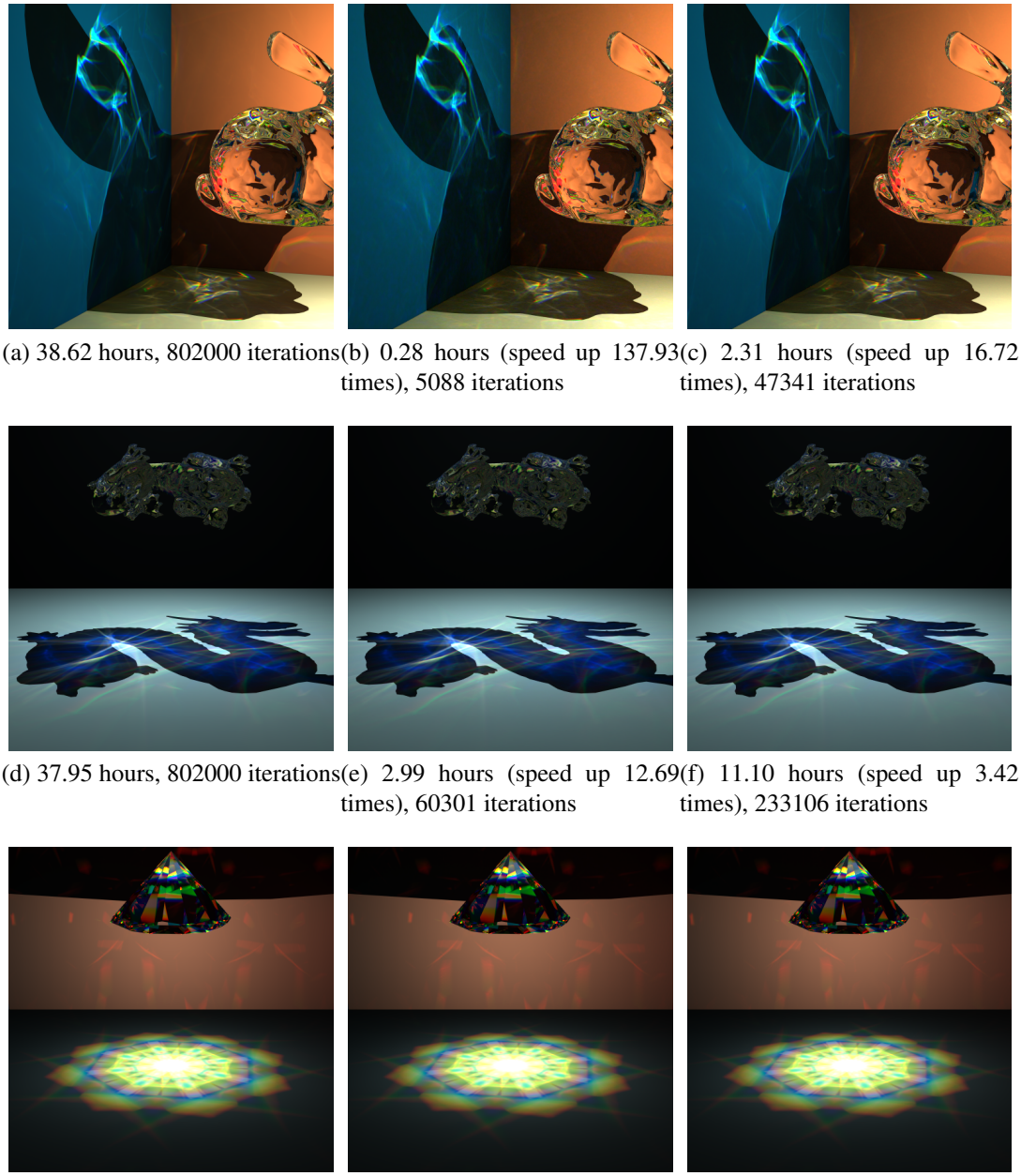
(a) Brute Force approach (b) Our approach

(c) Image difference

Figure 4.12: Rendering comparison of caustics that are produced by multiple refractions. We also show the image difference in (c).

4.3 Summary

We have presented a two-step scheme for accelerating spectral caustic rendering. Our experimental results show that our proposed scheme can achieve rendering quality close to the Brute Force rendering. Moreover, the range of our rendering speed acceleration (comparing to the rendering speed using Brute Force) magnitude is from tens to hundreds. We also compare our second acceleration step with the scene independent probability distribution function $u(\lambda)$ (for computing refinement iterations) proposed by Radziszewski et al. [43]. The experimental results show that our method can achieve higher acceleration while main-

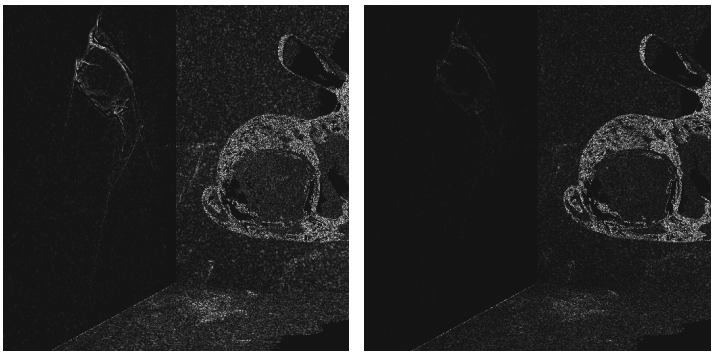


(a) 38.62 hours, 802000 iterations (b) 0.28 hours (speed up 137.93 times), 5088 iterations (c) 2.31 hours (speed up 16.72 times), 47341 iterations

(d) 37.95 hours, 802000 iterations (e) 2.99 hours (speed up 12.69 times), 60301 iterations (f) 11.10 hours (speed up 3.42 times), 233106 iterations

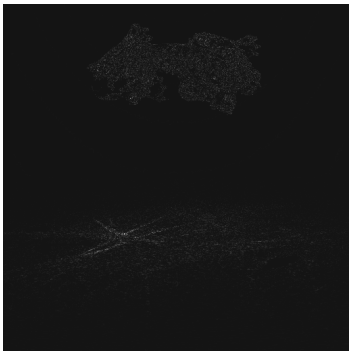
(g) 33.57 hours, 802000 iterations (h) 3.46 hours (speed up 9.70 times), 78714 iterations (i) 9.90 hours (speed up 3.39 times), 231239 iterations

Figure 4.13: Our experimental results. The results in each column are generated by using different methods. The first column is generated by using the Brute Force method (2000 iterations, with 1 nm step), the second column is generated by using our method, and the third column is generated by using our method but for the second acceleration step we use the probability distribution function proposed by Radziszewski et al. We use N-SF66 (Figure 4.1), Arbitrary 2 (Figure 4.3b), Arbitrary 1 (Figure 4.3a IOR curves on the caustic objects of the first, second, and third rows respectively.



(a)

(b)



(c)

(d)

(e)

(f)

Figure 4.14: Image differences of the results with brighter pixels indicating larger differences. For each pixel, we compute the Euclidean distance and we scale it by 10.0. The first column shows the image differences between the Brute Force rendering results and the second column rendering results in Figure 4.13. The second column shows the image differences between the Brute Force rendering results and the third column rendering results in Figure 4.13.

taining the quality of the rendering results comparable to the Brute Force results.

Our work consider only one type of caustic object in the scene (i.e. one IOR graph) and first refraction event. As such, it is possible to extend our work to do clustering by considering more than one caustic object types and/or refraction events. For example, we perform separate clustering for each type of caustic objects, and we combine the clustering results by analysing the intersections between their wavelength clusters (or sets).

It is also possible to do clustering by using another approach, i.e. wavelength clustering of each photon taking into account its incoming angle onto the caustic object surface. This approach requires additional bookkeeping in the process. For example, each photon will store a different number of wavelength clusters. Thus, the integration computation for each photon will involve different number of iterations. In GPU implementation, it is a good practice for each thread to do the same amount of workload and as such, it needs an in-depth investigation of the efficiency of this new clustering approach. We intend to try this alternative clustering approach and compare it with our current approach in the future.

In the second acceleration step, we use uniform sampling to sample material types surrounding the caustic object. However, the sampling may sample surfaces that are not hit by photons. It can be improved by using a real-time approximate caustic renderer to do the sampling for more accurate results.

For a very special case such as Newton's prism experiment with a receiver plane that is very tangential to its refraction direction, the generated rainbow pattern may stretch widely. Even if we render using the Brute Force approach, discontinuity in the rainbow pattern can be observed. One possible approach to remove such discontinuity is by combining our work with Elek et al.'s spectral ray differential [77].

In the future, we are interested in integrating our acceleration scheme into other rendering algorithms. For instance, we could apply our acceleration scheme to bidirectional path tracing. In this case, we cluster the rays of several wavelengths using our first acceleration step, and then vary the number of sample per pixels (or path lengths) based on our second acceleration step.

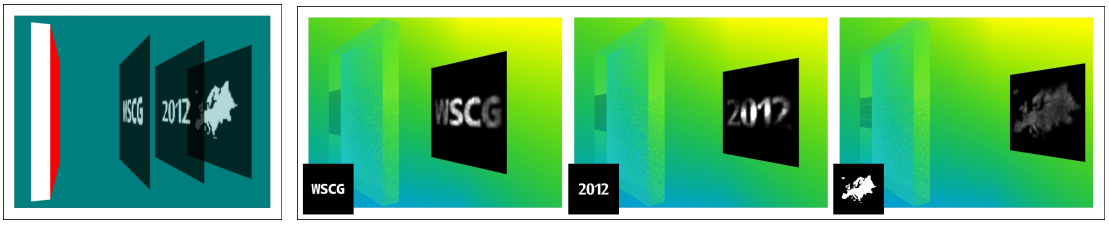
Chapter 5

Caustic Object Reconstruction Based on Multiple Caustic Patterns

Recently, there is a growing interest in inverse problem research in Computer Graphics due to the possibility of controlling the creation of visual effects. By using the inverse techniques, the design process becomes easier as artists can just specify the intended effects directly instead of performing the iterative trial-and-error process. However, inverse problem is generally difficult as in most cases there is no unique bijective relationship between the output and the input (i.e. given an output, there are many input possibilities that can generate such output).

In the inverse caustic problem, given an input caustic pattern (shape, intensity, and location from the caustic object) and a light source, we have to compute the geometry of a caustic object that can produce a caustic pattern similar to the input caustic pattern. Inverse caustic problem is hard to solve because the input caustic pattern only contains the irradiance magnitude and it does not have incident light direction information (and also the reflected and/or refracted light paths). Up to now, the inverse caustic problem is not widely studied and the existing work only considers a single input caustic pattern.

In this chapter, we propose a new inverse caustic problem, that is computing the caustic object given multiple input caustic patterns formed on a caustic receiver (diffuse and non-



(a) Scene setup. (b) Computed caustic object and approximately reconstructed caustic patterns.

Figure 5.1: (a) Scene setup. We compute a caustic object (the leftmost box), specifically the surface geometry of the side (shown in red colour) facing the caustic patterns, given three caustic patterns (WSCG, 2012, and Europe) to be formed on a caustic receiver at three distances from the caustic object, with a directional light source orthogonal to the caustic object illuminates from the left. (b) mental ray renderings of caustics produced by our computed caustic object (final output). Input caustic patterns are shown in the insets at each image. The computational time is 9.0 hours.

transparent surface) at various distances from the caustic object. We show an example in Figure 5.1.

Our basic idea for solving this problem is as follows. We subdivide one side of the caustic object and also the caustic patterns into regular cells. The light beam refracted by each caustic object cell will pass through one caustic cell of each caustic pattern. We try to compute the orientation of each caustic object cell such that the combination of the refracted light beams of all caustic object cells can approximately reconstruct the input caustic patterns.

We use a stochastic approach in our technique and we represent each input caustic pattern as a 2D probability mass function (pmf) by considering the brightness of a caustic cell as the probability of a light beam might pass through it (i.e. the brighter the caustic cell is, the higher probability or the more likely a light beam is considered to pass through it). Hence, for each cell of the caustic object, we use the pmfs of the caustic patterns to determine to which direction the caustic object cell refracts a light beam. From the determined refracted light beam direction, we can compute the orientation of the caustic object cell.

Due to differences of the input caustic patterns (in terms of shapes and intensities), it is hard to compute the caustic object that can satisfy all the input caustic patterns. Thus,

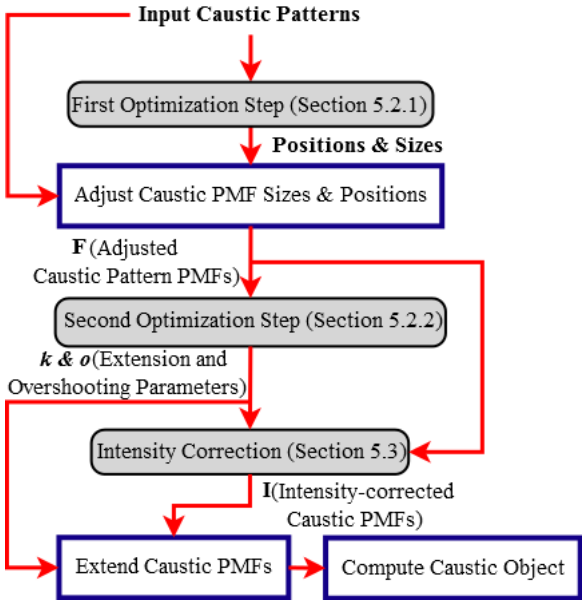


Figure 5.2: The overall pipeline diagram of our proposed method. Note the branches as the same data are used in multiple processes in the pipeline.

we relax the input requirements by slightly adjusting the sizes, positions, and shapes of the non-zero intensity regions of the input caustic patterns. Moreover, we also allow a small amount of light beam which has passed through several caustic patterns to miss or overshoot the rest of the caustic patterns. We compute these adjustments by using optimization techniques. We also propose an additional step for improving the intensity of the reconstructed caustic patterns. The flow of our proposed method for improving shape and intensity of reconstructed caustic pattern is shown in Figure 5.2.

As for the ground truth, currently there is no existing real-world caustic object that can generate different non-random/non-abstract caustic pattern shapes at different distances. We are the first one to propose this problem, as such it is not easy to generate a ground truth in the first place. Unlike the rendering case which has many established rendering engines or a brute force method that we can use to generate ground truth. For now, our evaluation is mainly comparison to the original input caustic patterns that we try to reconstruct. Moreover, we would also like to mention that existing inverse caustic works also do not evaluate their results based on a caustic object geometry as the ground truth [124, 126, 127, 128, 129, 130, 131]. Hence, we validate our results by performing rendering simulation

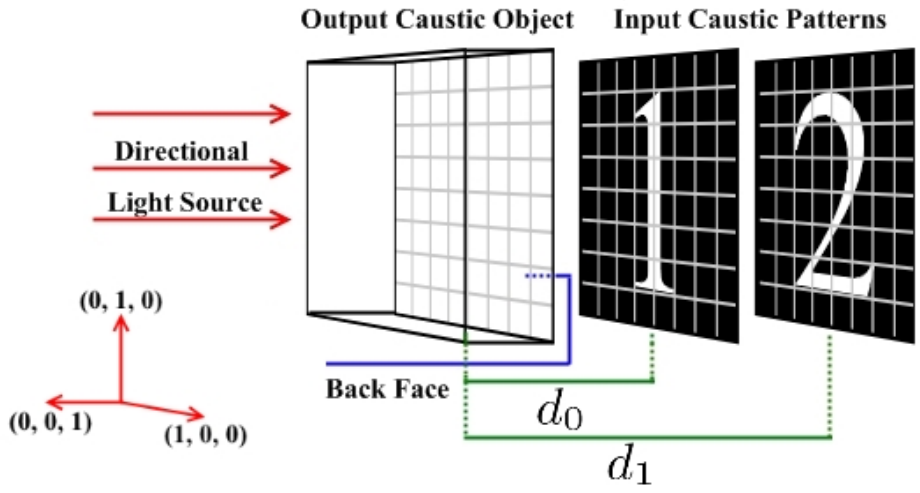


Figure 5.3: Scene setup. Our algorithm computes the normal/orientation of each caustic object cell such that each input caustic pattern can be reconstructed when it is located at specified user-input distance from the caustic object. For example, caustic pattern '1' is formed when the caustic receiver is at distance d_0 from the caustic object and similarly caustic pattern '2' at d_1.

using mental ray [4], a robust industry standard rendering engine.

5.1 Basic Idea of Our Method

Scene Setup The scene setup is shown in Figure 5.3. The scene consists of three components, a caustic object (of a box shape), a square caustic receiver (a diffuse planar surface where the caustic patterns are formed), and a directional light source. The caustic receiver is placed at the **back** of the caustic object (which faces (0, 0, -1) direction). The incoming directional light impinges the other side of the caustic object (front face of the caustic object which faces (0, 0, 1) direction). The direction of the incoming directional light source is (0, 0, -1), which is orthogonal to the front face of the caustic object. The caustic receiver orientation is identical to the front face of the caustic object, i.e. (0, 0, 1).

We assume the caustic receiver and the caustic object to have the same spatial dimension (i.e. same width and height) and orientation. Therefore, the extent of the region of interest of each caustic pattern is bounded by the shape of the caustic receiver

The caustic patterns and the back face of the caustic object are subdivided into a regular

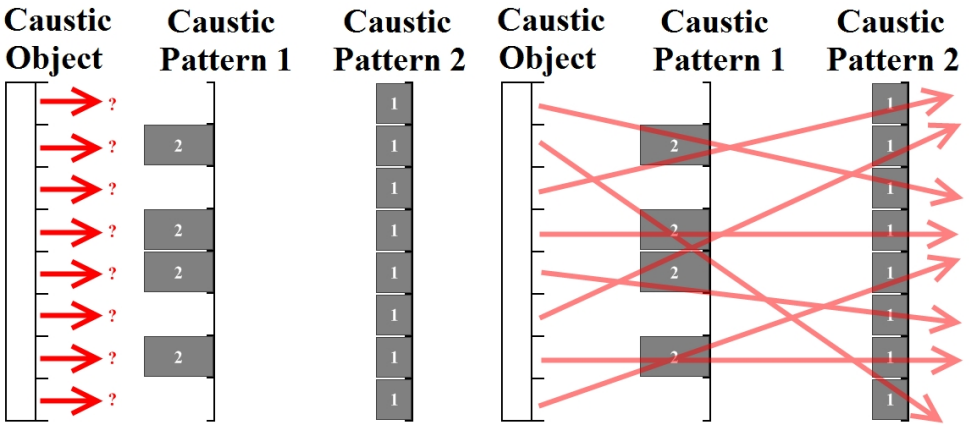


Figure 5.4: Illustration of problem formulation with two input caustic patterns. (a) Given two caustic patterns at two different distances from the caustic object (with the intensity of each caustic cell is denoted by the size of the cell), compute light refraction direction (red arrow) of each caustic object cell such that the refracted light collectively can generate caustic patterns similar to the input caustic patterns. (b) One of the possible light refraction combinations that can satisfy the input caustic patterns.

grid of cells. Each cell of a caustic pattern stores the total caustic intensity on that particular cell. We call cells of caustic patterns which have non-zero caustic intensity as **caustic cells** and cells with zero caustic intensity as **empty cells**. The collection of caustic cells of a caustic pattern is collectively called as a **caustic region**. For each cell of the caustic object (**caustic object cell**), we compute its orientation such that it can produce a refracted light beam to a specific direction. We assume the caustic object to be non-scattering and homogeneous (constant index of refraction for every point inside the caustic object. In other words, light does not undergo path bending inside the caustic object). Hence, the caustic object representation is a surface mesh instead of a volumetric mesh. We also assume each caustic object cell refracts the same amount of light power. In the rest of this chapter, we refer to each refracted light beam as light and we represent each light beam in Figures 5.4 and 5.5 as an arrow.

Problem Formulation We compute the back face of a caustic object C given a set of p input greyscale caustic patterns such that each of the j-th ($j = 0, 1, 2, \ldots, p-1$) input caustic pattern will be formed or reconstructed on the caustic receiver when the receiver is

located at the user-input distance d_j from the caustic object. Specifically, we compute the orientation of each caustic object cell at the back face of the caustic object such that the caustic object cell refracts the incoming light into a direction that passes through parts of the caustic regions. Collectively, the light refracted from all caustic object cells is expected to reconstruct the input caustic patterns. As mentioned before, the input caustic patterns only provide an estimate of the amount of refracted light arriving at caustic receiver cells, not the light directions. Hence, the main challenge is to compute refracted light paths that can approximately reconstruct all the given caustic patterns. This problem is illustrated in Figure 5.4. The orientation of each cell can then be determined based on the path of its refracted light.

Solution As explained above, the task is to compute refracted light direction combinations such that they can approximately reconstruct the input caustic patterns. In this case, more light is expected to pass through brighter caustic cells compared to darker caustic cells. Hence, to solve this, we simulate the direction of the refracted light of each caustic object cell by using a stochastic approach. The idea is to use the caustic intensity in each caustic cell as the probability that we will refract a light to that caustic cell (i.e. the brighter the input caustic pattern is, the more likely it is chosen as a refracted light target).

We represent the set of caustic patterns as a set of normalized 2D probability mass functions $\mathbf{P} = \{f_{P_0}, f_{P_1}, f_{P_2}, \dots f_{P_{p-1}}\}$ (each in $\mathbf{P}$ is the pmf of the user-input caustic pattern on the caustic receiver when the receiver is located at the user-input distance from the caustic object). The pmf of each caustic pattern is defined by using the greyscale value (intensity) of the caustic pattern in which the probability at each caustic cell is the greyscale value of that particular cell in the caustic pattern. Each pmf f_{P_j} is normalized by dividing the probability of each of its cell with the total probability of all cells of f_{P_j}.

We assign a random variable X_i for each i-th cell of the caustic object. For each X_i, the probability value of each possible refracted light direction $\mathbf{x}$ is computed by multiplying the probability value of each caustic cell passed by the light refracted to direction $\mathbf{x}$ (see

Figure 5.5), as shown in Equation 5.1.

$$f_{X_i}(\mathbf{x}) = \prod_{j=0}^{p-1} f_{P_j}(\rho_j^i(\mathbf{x})), \tag{5.1}$$

where $\rho_j^i(\mathbf{x})$ is a mapping function. The mapping function is basically a ray casting function in which the light is shot from the i-th caustic object cell to the caustic pattern j with the direction of $\mathbf{x}$ and return the intersected cell (of caustic pattern j). The range of $\mathbf{x}$ values is the range of possible refracted light direction from the i-th caustic object cell. In the implementation, we set the camera in the middle of the i-th cell and we render the caustic patterns. The multiplication in Equation 5.1 is achieved by using alpha blending. Hence, the coverage of the camera corresponds to the range of $\mathbf{x}$ values. Then, for each caustic object cell or each X_i we assign a refracted light direction by using the Acceptance-Rejection method [152] with the distribution based on the sampled joint probability mass function of all caustic patterns (Equation 5.1).

Once we obtain the refracted light direction for each caustic object cell, we compute its normal or orientation based on the user-input index of refraction of the caustic object, incoming light direction (orthogonal to the caustic object), and the obtained refracted light direction by solving the Snell's Equation. Afterwards, we perform rendering simulation using the mental ray to assess the approximate reconstructed caustic patterns (as shown in Figure 5.10a). Note that our technique can also be applied to point light sources and directional light sources non-orthogonal to the caustic object. In the rest of this chapter, the terms caustic patterns and pmfs are interchangeable as we use the term pmfs when we emphasize on the mathematical representation of the caustic patterns.

5.2 Improving the Reconstructed Caustic Pattern Shapes

The solution in Section 5.1 may not be able to reconstruct the caustic patterns very well if the input consists of multiple patterns. If we only have a single caustic pattern (shown

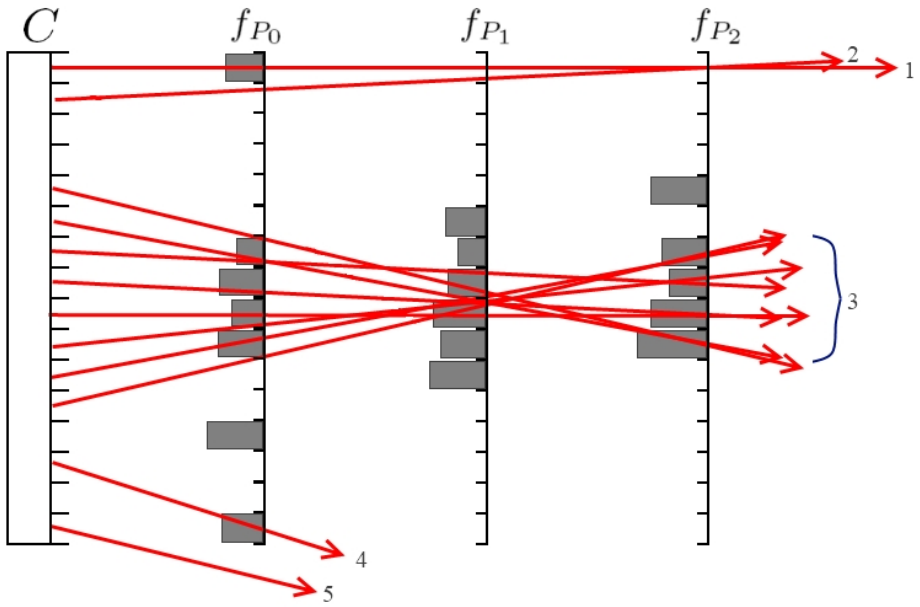
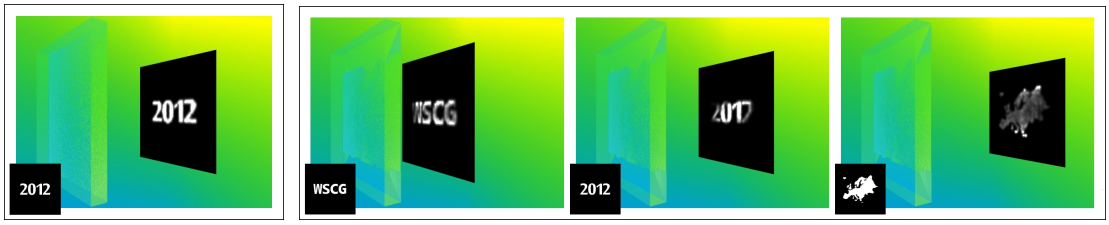


Figure 5.5: Each numbered arrow denotes the refracted light from the cells of the caustic object C and the grey blocks denote the probability of caustic cells. Light $\#1$ and $\#2$ have joint pmf of zero since their paths pass through at least one empty cell. Each light in $\#3$ has the probability greater than zero since the light pass through non-zero cells. Light $\#4$ is allowed even though it misses some of the caustic patterns. We will explain this further in Section 5.2.2. Light $\#5$ is not valid because it does not intersect any caustic patterns

in Figure 5.6a), then we can reconstruct it very well. However, if we add two additional caustic patterns, then some parts of the input caustic patterns are missing (Figure 5.6b).

Reconstruction problem As explained in Section 5.1 (and shown in Figure 5.5), some refraction directions have zero joint pmf when they pass through at least one empty caustic cell. As a result, if all possible refraction directions $\mathbf{x}$ from all caustic object cells pass through a caustic cell of the caustic pattern a but they also pass through empty cells of all other caustic patterns b ($0 \leq b \leq p-1, b \neq a$), then the aforementioned caustic cell cannot be reconstructed (we call such cell as a **missing caustic cell**). As seen in Figure 5.5, the topmost and bottommost caustic cells in f_{P_1} are missing caustic cells since the refracted light that pass through these caustic cells also pass through empty cells in the other caustic patterns.

Proposed solution Based on the given input caustic patterns and their configurations (positions and sizes), it might not be possible to compute the caustic object that can well



(a) (b)

Figure 5.6: (a) Only a single caustic pattern and it can be reconstructed very well. (b) Two additional caustic patterns cause some parts of the input caustic patterns to be missing.

reconstruct the original input caustic patterns. Thus, we propose a method to relax the input requirement by allowing slight changes to the positions, sizes, and shapes of the caustic regions. Our proposed method consists of two steps. In the first step, we optimize the size and position of the caustic regions by slightly adjusting the size and position given by the user (Section 5.2.1). In the second step, the boundaries of each caustic region are adaptively extended such that they enable the reconstruction of the missing caustic cells on the other caustic patterns (Section 5.2.2). We also compute the amount of light that is allowed to overshoot or to miss some caustic patterns.

In both optimization steps, we use Simulated Annealing [10]. The main reason we use Simulated Annealing is because the problem cannot be solved analytically. However, there are also some possible optimization techniques such as Particle Swarm, Ant Colony, and Genetic Algorithm. However, those techniques require keeping the record of multiple possible solutions at once, hence it is not efficient for our case (as seen in Equation 3, the cost computation requires reconstruction of the caustic patterns multiple times using the adjusted input caustic patterns). Moreover, in some of the related work [128, 126, 127], Simulated Annealing is also used. After applying our proposed solution, the reconstructed caustic patterns from the test case in Figure 5.6b are improved as seen in Figure 5.1b.

Cost computation In every iteration of both optimizations, we use the root mean square to compute the cost or degree of possibility that the adjusted input caustic pattern can be approximately reconstructed (in order to guide the simulated annealing). The root mean square is computed as the difference between the normalized reconstructed caustic pat-

terns $\mathbf{Z} = \{f_{Z_0}, f_{Z_1}, \dots f_{Z_{p-1}}\}$ and the normalized adjusted input caustic patterns $\mathbf{D} = \{f_{D_0}, f_{D_1}, f_{D_2}, \dots f_{D_{p-1}}\}$. Only in the cost computation here, the normalization of caustic patterns in $\mathbf{Z}$ and $\mathbf{D}$ are computed by dividing the value of each caustic cell $\mathbf{t}$ with the maximum caustic cell value of the caustic pattern $\mathbf{t}$ belongs to. We compute the cost in this way such that the maximum cost (which is in the worst case scenario, for example the input caustic patterns are not reconstructed at all) is 1.0. The cost computation is shown in Equation 5.3.

$$Cost = \frac{1}{p}\sum_{j=0}^{p-1}\sqrt{\frac{1}{n(\mathbf{W}(j))}\sum_{i=0}^{\beta}(f_{D_j}(\mathbf{t}_i) - f_{Z_j}(\mathbf{t}_i))^2}, \tag{5.2}$$

where

$$\mathbf{W}(j) = \{\mathbf{t} | f_{D_j}(\mathbf{t}) + f_{Z_j}(\mathbf{t}) > 0\}, \tag{5.3}$$

and $\mathbf{t}$ is the caustic cell, p is the number of caustic patterns, $n(\mathbf{W}(j))$ is the number of elements of a set of caustic cells contributing to the cost computation, and β is the total number of cells of each caustic pattern (in our experiments, $\beta = 64 \times 64 = 4096$). For more accurate computation of $\mathbf{Z}$, we approximately reconstruct the caustic patterns 32 times and accumulate their caustic cell values, and finally we divide the value of each caustic cell by 32 in order to get $\mathbf{Z}$.

We reconstruct the caustic patterns by computing the refracted direction as explained in Section 5.1 and then for each caustic cell we accumulate the amount of refracted light that intersects it. We use the modified pmfs (which are adjusted in each optimization step) to compute the joint pmfs (Equation 5.1).

5.2.1 Adjusting the Size and Position

In this first optimization step, we relax the input caustic pattern by iteratively adjusting the size and/or position of the input caustic regions. In this step, we have two possible options, namely adjusting the position and size of the input caustic patterns directly or adjusting the light convergence.

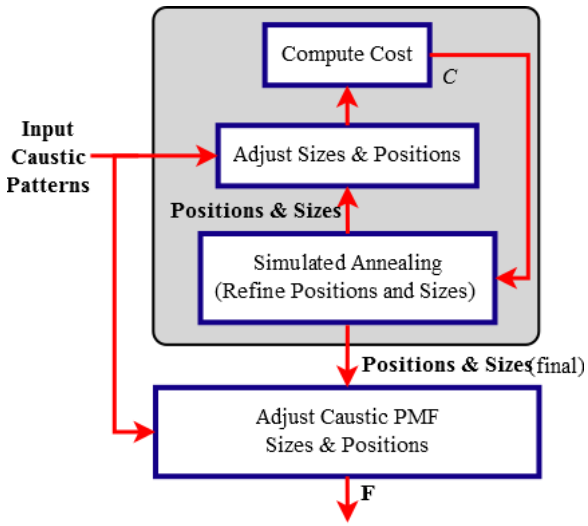


Figure 5.7: The flow for the size and position adjustment optimization step.

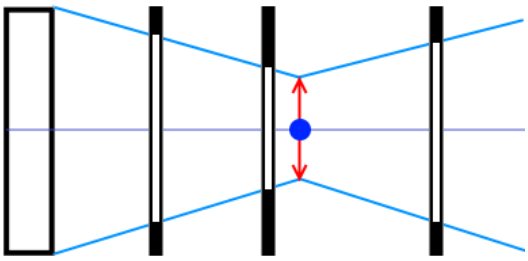


Figure 5.8: We optimize for the good convergent size (vertical arrows).

In the first option, adjusting the position is basically translating the caustic regions in 3D space (translation in x, y, z). This means we also adjust the input distance (translation in z) between the caustic object and the caustic pattern (caustic receiver). When we do the scaling, we scale each caustic region independently. Hence, the total number of variables that we process in this optimization is $4p$ variables.

In the second option, we change the convergent size, as shown in Figure 5.8. Given a convergent size (denoted by vertical arrows in Figure 5.8), the input caustic patterns shapes are bounded by the intersection between the light volume (area bounded by the blue lines in Figure 5.8, which we assume to be the overall refracted light path in the configuration) and the caustic receiver. Hence, by using this option we only need to iteratively adjust one variable value, i.e. the convergent size, and we can directly obtain the size of the caustic patterns. The main rationale for this proposed second option is that in a lens system the

refracted light tends to diverge and/or converge with respect to the focal point. Our caustic object in this case behaves similar to a lens.

Both the first and second options have their own advantages and disadvantages. It is more flexible in the first option since we have more options compared to the second option. However, the second option is more stable (i.e. we obtain almost the same results for every experiment with the same inputs) compared to the first option.

We show the flow of the first optimization step in Figure 5.7. In every iteration, we adjust the sizes and positions of the input caustic regions (i.e. compute the randomized changes in sizes and positions and accumulate them to the sizes and positions of the previous iteration) and compute the cost by using Equation 5.2 in order to guide the optimization iterations. The adjusted input caustic patterns are used as the input to the next optimization step (Section 5.2.2) and they are also the target caustic patterns for every iteration of the second step. We ran the experiments for 10 cycles of 10 iterations (total 100 iterations).

In order to determine which of these two options are better, we used them separately and we then asked 15 people to judge which option can reproduce better caustic patterns. The survey consists of four set of caustic patterns. The participant then voted for the better result for each set of caustic patterns. Afterwards, for each option of each set, we compute its average voted percentage. The one that has higher average percentage is considered to be superior. From the survey result, three caustic pattern sets of the second option were voted with slight higher scores (i.e. 60%, 53.33 %, and 53.25%). From the score, we can see that the two options are interchangeable since they do not differ much. Nevertheless, since the second option obtains higher average scores for three caustic pattern sets, we suggest the second option to be used. For more detail of the survey, please refer to Appendix B. We show one of the comparisons between the first and second options in Figure 5.9.

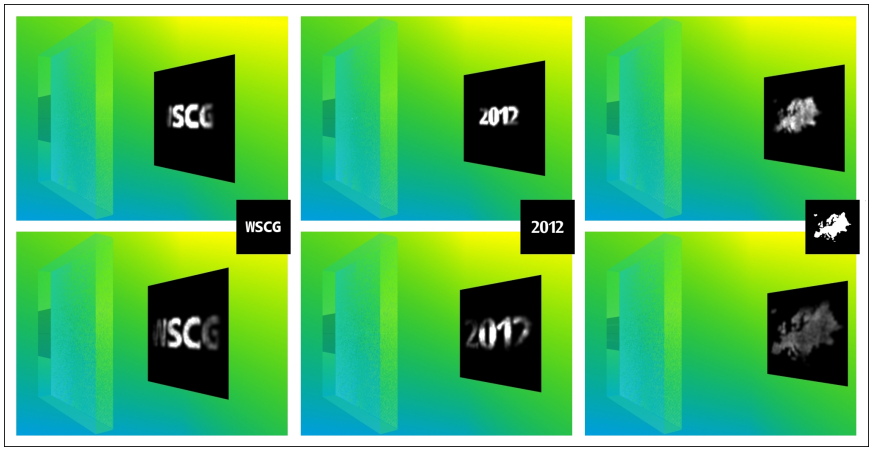


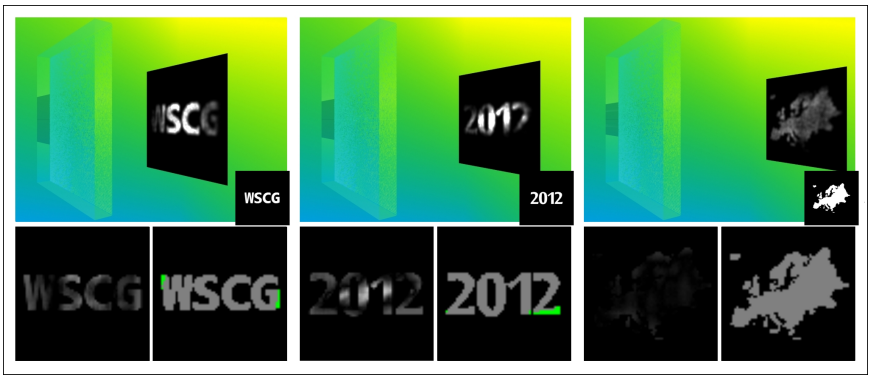
Figure 5.9: Comparison between the first option (top row) and second option (second row) of our proposed first optimization. We can see from the first and the third caustic patterns that the second option is superior.

5.2.2 Extending Caustic Regions and Overshooting Refracted Light

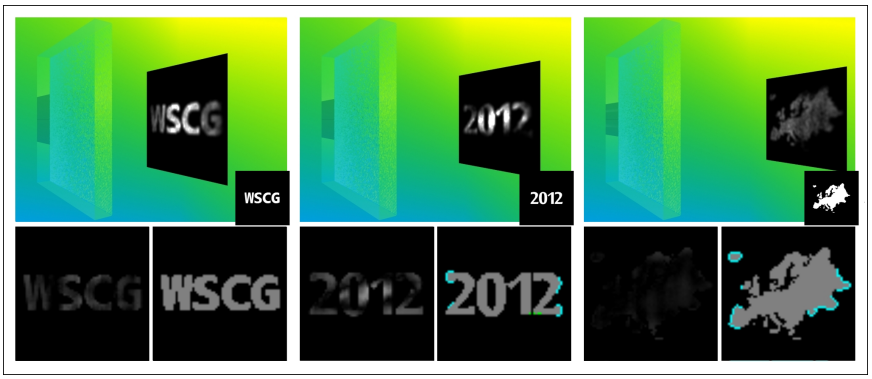
After performing the first optimization step, there might be some missing caustic cells left. Missing caustic cells are the caustic cells that cannot be reconstructed. To reconstruct some of these missing caustic cells, we slightly extend the shape of all input caustic regions. For example, in Figure 5.5, the middle-top and middle-bottom caustic cells of f_{P_1} cannot be reconstructed since all possible refracted light that passes through these cells in f_{P_1} have to pass through empty cells in either f_{P_0} or f_{P_2}. Hence, to solve this, we can extend the middle caustic regions in f_{P_0} and f_{P_2} up and down by one cell.

Some caustic cells especially around caustic patterns borders are hard to reconstruct as most light passing through these cells can miss other caustic patterns in behind. Thus, we relax this requirement by enabling some of the refracted light that passes through caustic cells of one or several caustic patterns to miss the rest of the caustic patterns (or region of interests of the rest of the caustic patterns). This is beneficial especially for the caustic cells on the border of the caustic patterns. For example, in Figure 5.5, if we enable light $\#4$ to pass the bottommost caustic cell of f_{P_0} and miss the rest of caustic patterns (we call this **overshoot**), then the bottommost caustic cell of f_{P_0} can be reconstructed. However, we still do not allow the refracted light of one caustic object cell to miss all of the caustic patterns

(a) Results without optimization. Cost : 6.10×10^{-1}



(b) Results after 1st optimization (Section 5.2.1). Cost : 5.90×10^{-1}



(c) Results after 1st and 2nd optimization (Section 5.2.2). Cost : 5.32×10^{-1}

Figure 5.10: Mental ray rendering results of the optimization steps. Input caustic patterns are shown at the bottom right of each screenshot. We show the missing caustic cell maps at the bottom right of each image (green cells show missing caustic cells, grey cells show caustic cells that can be reconstructed, and cyan cells show extended caustic cells). We also show the caustic irradiance difference maps (differences between the target and the reconstructed caustic patterns). For the sake of visual clarity, we scale up the difference values by 5000. The computational time is 2.72 hours.

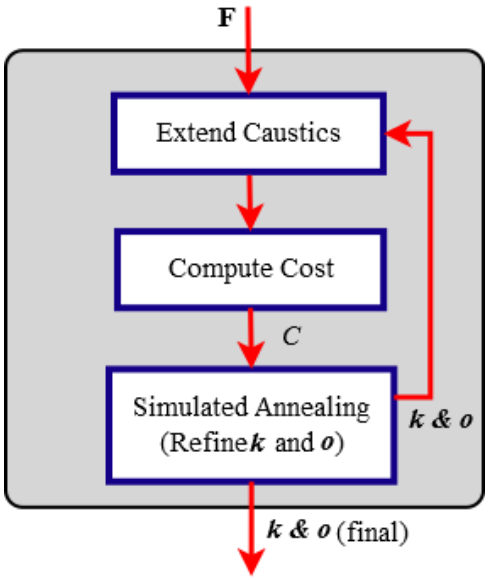
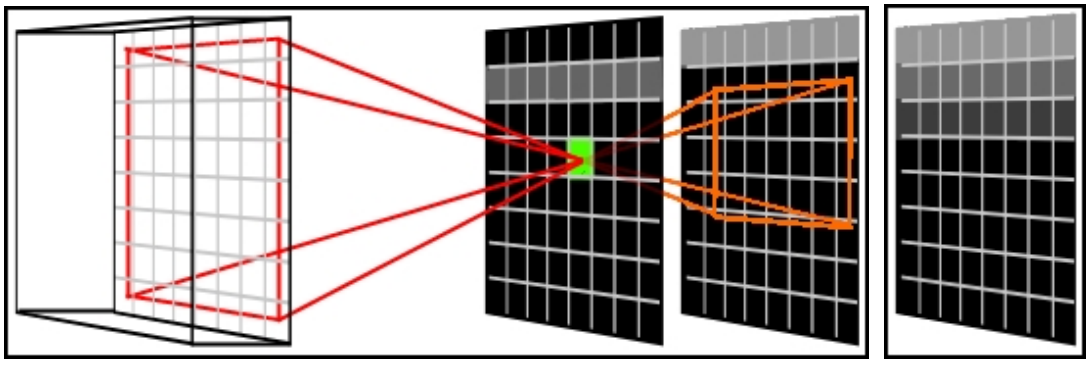


Figure 5.11: The flow for the caustic region extension and refracted light overshooting adjustment optimization step.

(as in light $\#5$).

Fully extending the caustic regions can deform the original caustics too much, and likewise if we allow too much light to overshoot the caustic patterns then the approximate reconstructed caustic patterns will have very low intensity. Hence, in this step, we apply simulated annealing to determine the appropriate caustic regions extensions amount $\mathbf{k} = \{k_0, k_1, \ldots k_{p-1}\}$ and light overshoot amount $\mathbf{o} = \{o_0, o_1, \ldots, o_{p-1}\}$ with $\mathbf{k}$ and $\mathbf{o}$ $\in [0, 1]$ (i.e. a k and an o value for each caustic pattern). Similar to the previous step, we iteratively adjust the $\mathbf{k}$ and $\mathbf{o}$ values in order to get the lowest cost. We ran the experiments for 10 cycles of 10 iterations (total : 100 iterations). We show the main flow of the second optimization step in Figure 5.11. The computation of extension and overshooting is explained in detail as follows.

Extending Caustic Regions We define s_j to be the extension amount of caustic pattern j. To enable the missing caustic cells of a caustic pattern a to be reconstructed, we have to firstly compute at most how many s_b^a unit cells away the caustic region boundaries of the other caustic patterns b ($0 \leq b \leq p-1, b \neq a$) have to be extended. Afterwards, for every caustic pattern j, we extend its caustic region with the amount of $s_j \cdot k_j$. In order to



(a) (b)

Figure 5.12: (a) A simple example of missing caustic cell projection (is explained in Section 5.2.2). Grey cells are the caustic cells and the green cell is the missing caustic cell. (b) We extend the second caustic pattern (two cells away) with gradually decreasing intensity.

enable smooth extension, we extend the caustic regions with linearly decreasing intensity (or probability value).

To do this, for every missing caustic cell of the caustic pattern a, we project it from every caustic object cell to the empty cells of other caustic patterns b ($0 \leq b \leq p-1, b \neq a$). We perform this projection for the missing caustic cells of all caustic patterns. This projection example is shown in Figure 5.12a in which we project the missing caustic cell (shown in green colour). Afterwards, for each caustic pattern b, we obtain the maximum distance (s_b^a) between its caustic region boundaries and its empty cells that receive the projections of the missing caustic cells (of other caustic patterns). Because the number of input caustic patterns might be greater than two, there will be more than one extension amount for each caustic pattern b, depending on the projection from which other caustic pattern a. For example, if there are three caustic patterns, the extension amount for the third caustic pattern because of the first caustic pattern (s_2^0) might be different from the extension amount because of the second caustic pattern (s_2^1). Hence, the final s_j of each caustic pattern is $s_j = \max\{s_j^0, s_j^1, \ldots s_j^{p-1}\}$.

In the example in Figure 5.12a, we need to reconstruct the caustic pattern cell denoted with green colour. In order to achieve this, we need to know how much the second caustic pattern (consists of one row of caustic pattern cells at the top) should be extended. We

compute all possible refraction directions from the caustic object that pass through the missing caustic pattern cell (shown as the volume bounded by the red and orange lines in Figure 5.12a). From this projection, we can see that the second caustic pattern should be extended downward for at most five rows, hence $s_j = 5$. If the optimization results in $k_j = 0.4$, then the second caustic pattern is extended downward for $s_j \cdot k_j = 5 \cdot 0.4 = 2$ rows (shown in Figure 5.12b).

Some of the missing caustic cell projections might miss the other caustic patterns b. For example, if the missing caustic cell in Figure 5.12a is one or two cells to the right, then some of the projections will overshoot or will not hit the second caustic pattern. We use this information to control the possibility of the refracted light to overshoot each caustic pattern.

Overshooting Refracted Light To improve the results, we also enable the refracted light to overshoot some of the caustic patterns. Thus, during the missing caustic cells projections, we also compute the ratio (e_b) between the amount of these projections that do not hit caustic pattern b and the total amount of these projections toward caustic pattern b (i.e. sum of the missing caustic cell projections that hit and do not hit caustic pattern b). e_b is essential since it provides the information on the amount of probability that the refracted light miss plane b. Hence, the probability h_b that we will refract the light to miss the caustic pattern b is shown in Equation 5.4.

$$h_b = e_b \cdot o_b \cdot f_{P_b}(\mathbf{t}_{max}), \tag{5.4}$$

where o_b is the coefficient to control the probability of overshooting b-th caustic pattern and $\mathbf{t}_{max}$ is a cell of f_{P_b} with the highest probability value. In the implementation, we define the probability value outside caustic plane b to be h_b.

We show the optimization progression in Figure 5.10.

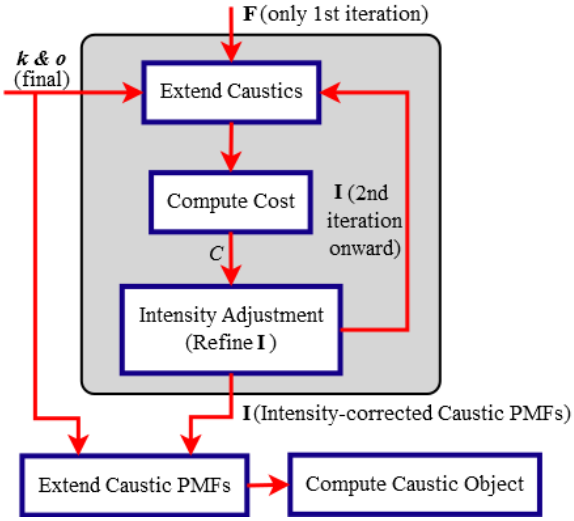


Figure 5.13: The flow for the intensity correction step.

5.3 Intensity Correction

In the previous Section 5.2, we propose two optimizations that can improve the shape of the reconstructed caustic patterns. Another important aspect which needs to be considered is the intensity of the reconstructed caustic patterns. Due to the differences between the input caustic patterns, some bright caustic cells in one pattern might be less likely to be selected as the refraction targets due to the caustic cells of other caustic patterns. This also holds for the inverse case, less bright caustic cells might be very bright in the final result because they are most likely to be selected as the refraction targets due to the caustic cells of other caustic patterns.

To solve this issue, we iteratively refine the pmf of each caustic pattern based on the intensity difference between the intended intensity and the reconstructed intensity. In this intensity correction step, we try to do cell-wise adjustment on the input pmf such that the intensity of input caustic pattern can be well reconstructed.

In the intensity correction, we iteratively refine the pmf results of the first optimization. Before computing the cost, we apply the extension and the overshooting parameters we obtain from the second optimization step. The main flow of the intensity correction step is shown in Figure 5.13.

Let $\mathbf{F} = \{f_{F_0}, f_{F_1}, \ldots f_{F_{p-1}}\}$ be the output pmfs of the first optimization and let $\mathbf{I} = \{f_{I_0}, f_{I_1}, \ldots f_{I_{p-1}}\}$ be the iteratively refined pmfs in this intensity correction step.

As shown in Figure 5.2, we firstly set $\mathbf{I} = \mathbf{F}$ (only for the 1st iteration of intensity correction step). Afterwards, from 2nd iteration onward, we refine $\mathbf{I}$. In every iteration, we compute the average achievable reconstructed caustic pattern f_{Z_j} based on $\mathbf{I}$ by reconstructing the caustic patterns 32 times and accumulate their results (we perform similar computation when we compute the cost in Section 5.2). For each caustic pattern cell $\mathbf{t}_i$, we scale the $f_{I_j}(\mathbf{t}_i)$ by $f_{F_j}(\mathbf{t}_i)/f_{Z_j}(\mathbf{t}_i)$. In our work, we perform this intensity correction until to either 50 or 5 iterations with cost changes below threshold (0.00001). As we show in the cost history in Figure 5.20, some experiments stop before the end of 100 + 100 + 50 iterations as the intensity correction step is unable to a obtain lower cost after five iterations. The final pmf of the intensity correction is the pmf on the intensity correction iteration that has the lowest cost. After we finalize the caustic pattern pmfs $\mathbf{I}$, we apply the extension and overshooting parameters we obtain from the second optimization step and we finally compute the caustic object geometry.

To assess the effectiveness of the intensity correction, we perform another survey that compares the results of two optimization steps with intensity correction and the results of two optimization steps without intensity correction. The survey uses the same methodology and input caustic pattern sets as the survey explained in Section 5.2.1.

In our experiment, the intensity correction is unable to improve the results on two out of four test cases or caustic pattern sets, hence almost the same caustic patterns were shown to the participants. As a results, the average scores for the results with and without intensity correction for those two caustic pattern sets are almost equal (near 50%). For the other two caustic pattern sets, the results with intensity correction obtain higher average score (i.e. 77.78% and 91.67%). Hence, we can conclude that our intensity correction step can improve the results. For more detail of the survey, please refer to Appendix B. We show one of the comparisons between without and with intensity correction in Figure 5.14.

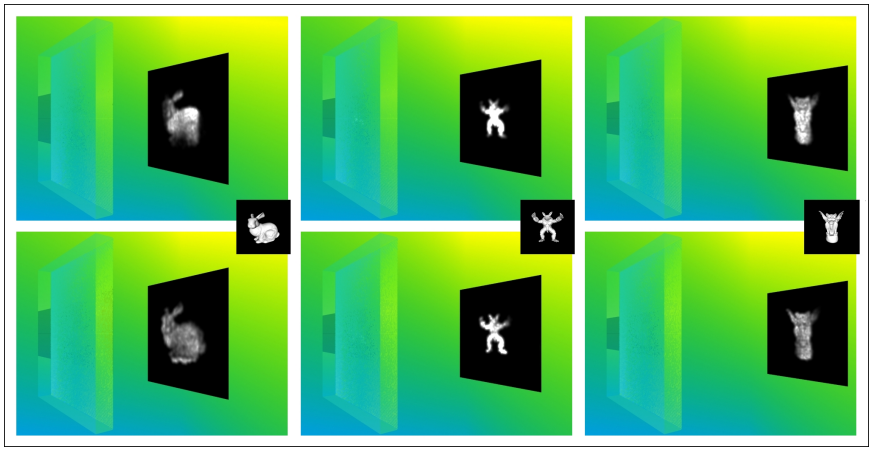


Figure 5.14: Comparison between without (top row) and with intensity correction (bottom row). We can see that by using intensity correction, the caustic pattern is generally improved, especially the first caustic pattern.

5.4 Geometry Construction

Given the refracted light directions for each caustic object cell, the next step in the work is computing the geometry of caustic object. This is done by firstly compute the normal of each caustic object cell given the refracted direction. After the normal of each caustic object cell is obtained, we can compute the geometry of each cell and we assume the height of all caustic object cell centroids to be the same.

5.4.1 Computing Caustic Object Cell Normal

From the obtained refraction direction of each caustic object cell, we compute its normal or orientation. As the vectors of incoming light direction, refraction direction, and surface normal are coplanar, the problem can be simplified by solving it in 2D.

Let $\mathbf{M}$ be the inverse incoming light direction, $\mathbf{R}$ be the refracted light direction, θ_1 be the angle between $\mathbf{M}$ and the inverse normal, θ_2 be the angle between $\mathbf{R}$ and the normal. Assume $\mathbf{R}^{2D}$ to be the refracted light direction in 2D domain and its direction to be (1, 0, 0). Then, direction of $\mathbf{M}^{2D}$ (inverse incoming light direction in 2D domain) can be computed based on the angle between $\mathbf{M}$ and $\mathbf{R}$ (obtained by using dot product), as the angle between $\mathbf{M}^{2D}$ and $\mathbf{R}^{2D}$ is equal to the angle between $\mathbf{M}$ and $\mathbf{R}$. We show the 2D representation in

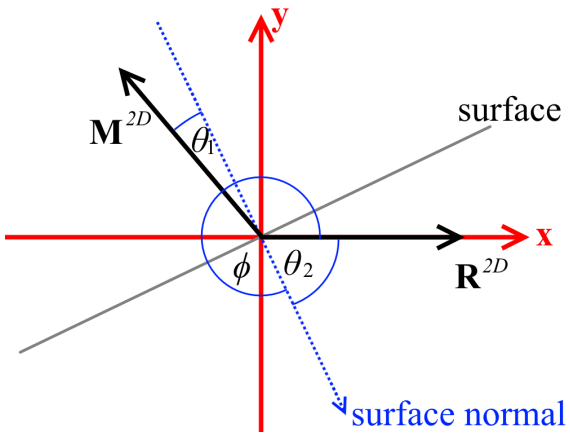


Figure 5.15: Refraction in 2D domain. We assume the refraction direction to be parallel with the positive **x** axis.

Figure 5.15.

Given the Snell's Equation:

$$\begin{aligned}
\eta_1 \sin\theta_1 &= \eta_2 \sin\theta_2, \\
\eta\sqrt{1-\cos^2\theta_1} &= \sqrt{1-\cos^2\theta_2}, \\
\eta\sqrt{1-(-\mathbf{N}^{2D}\cdot\mathbf{M}^{2D})^2} &= \sqrt{1-(\mathbf{N}^{2D}\cdot\mathbf{R}^{2D})^2}, \\
A_1 N_x^{2D} N_x^{2D} + A_2 N_x^{2D} N_y^{2D} + A_3 N_y^{2D} N_y^{2D} &= 1,
\end{aligned} \tag{5.5}$$

where η_1 is the index of refraction of the caustic object, η_2 is the index of the refraction of air, $\mathbf{N}^{2D}$ is the normal in 2D domain, and

$$\eta = \tfrac{\eta_1}{\eta_2} \qquad A_1 = \tfrac{\eta^2 M_x^{2D} M_x^{2D} - R_x^{2D} R_x^{2D}}{\eta^2 - 1}$$
$$A_2 = 2\tfrac{\eta^2 M_x^{2D} M_y^{2D} - R_x^{2D} R_y^{2D}}{\eta^2 - 1} \qquad A_3 = \tfrac{\eta^2 M_y^{2D} M_y^{2D} - R_y^{2D} R_y^{2D}}{\eta^2 - 1}$$

Since the normal vector $\mathbf{N}^{2D}$ is normalized, the solution lies on a unit circle and as a result $N_x^{2D} = \cos\phi$ and $N_y^{2D} = \sin\phi$. Hence, Equation 5.5 can be simplified to

$$(2A_3 - 2)\tan^2\phi + 2A_2\tan\phi + (2A_1 - 2) = 0. \tag{5.6}$$

Equation 5.6 is basically a quadratic equation and the angle ϕ can be obtained by solving

the quadratic equation.

As we showed in Figure 5.15, the surface normal is obtained by rotating vector $\mathbf{R}^{2D}$ with the amount of ϕ. Similarly, to obtain the caustic object cell normal in the original 3D domain, we rotate $\mathbf{R}$ with the amount of ϕ, with the rotation axis obtained from the cross product $\mathbf{R} \times \mathbf{M}$.

5.4.2 Computing Caustic Object Cell Geometry

Based on the computed orientation of each caustic object cell, we can obtain the caustic object geometry by computing the x, y, z coordinates of the four corners of each caustic object cell. The x, y coordinates of each caustic object cell corner can be easily found as the caustic object is uniformly subdivided. To compute the z coordinate of the caustic object cell corners, we firstly assume that the z coordinate of all caustic object cell middle points to be the same (z = 0.0). Next, we use the dot product operation, i.e. dot product between the normal of the caustic object cell and the vector $\mathbf{B}$ from the caustic object cell middle point (we know its x, y coordinates from the uniform subdivision and also its z coordinate which is 0.0) to each caustic object cell corner (we know its x, y coordinates) must be equal to zero. In this case, the only unknown value is z of the caustic object cell corner. The computation is shown in Equations 5.7-5.9.

$$N_x B_x + N_y B_y + N_z B_z = 0, \tag{5.7}$$

where

$$\mathbf{B} = (Bx, By, Bz) = (x_{corner} - x_{middle}, y_{corner} - y_{middle}, z_{corner}), \tag{5.8}$$

thus

$$z_{corner} = \frac{-N_x(x_{corner} - x_{middle}) - N_y(y_{corner} - y_{middle})}{N_z}. \tag{5.9}$$

Since the edges of the neighbouring caustic object cells do not have the same slope or the same z coordinate on both endpoints, there are vertical open spaces between caustic object

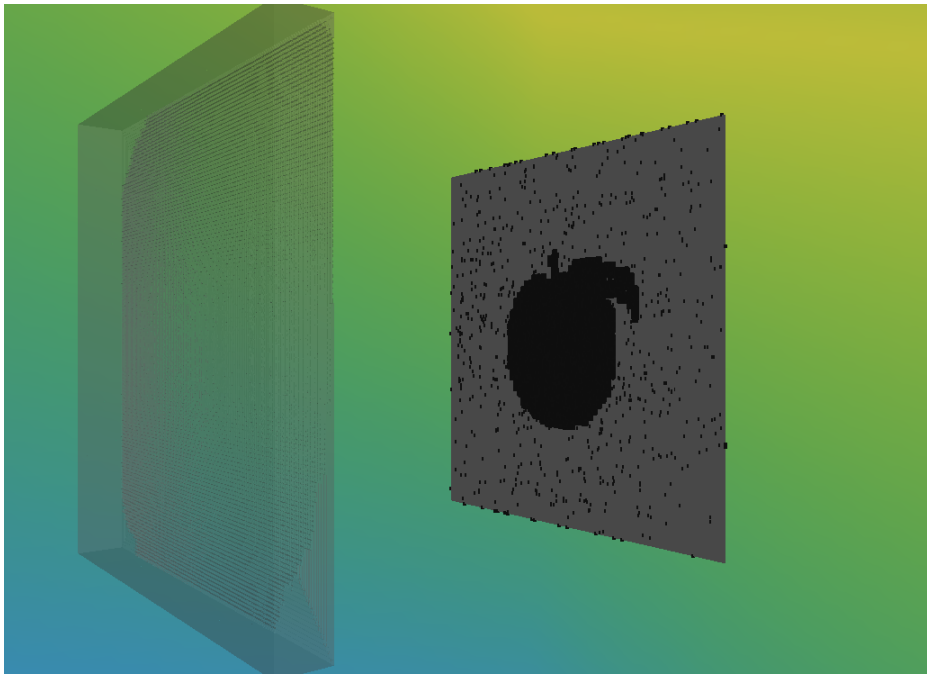


Figure 5.16: Visualization of photons (black dots) for the first caustic pattern of the test case in Figure 5.19a.

cells (shaded with lighter grey in Figure 5.17). Therefore, we generate additional polygons to close those gaps.

Due to uneven surface patches, some refracted light might hit neighbouring caustic object cell. This is evident from our rendering experiment using mental ray where we notice some stray photons on caustic pattern receivers in the photon map visualizer (Figure 5.16). However, the amount of stray photons is negligible as they are not conspicuous in the rendering results.

5.5 Results

We present some results computed using our technique in Figures 5.1, 5.10, 5.19. We also show the cost history of the optimization iterations in Figure 5.20. In all of the test cases, the index of refraction of the caustic objects are 1.5, the resolution of the caustic objects are 128×128 and the resolution of the caustic receiver is 64×64. Resolution refers to the number of cells. If we assume the spatial size to be 1.0×1.0, then the size of each caustic

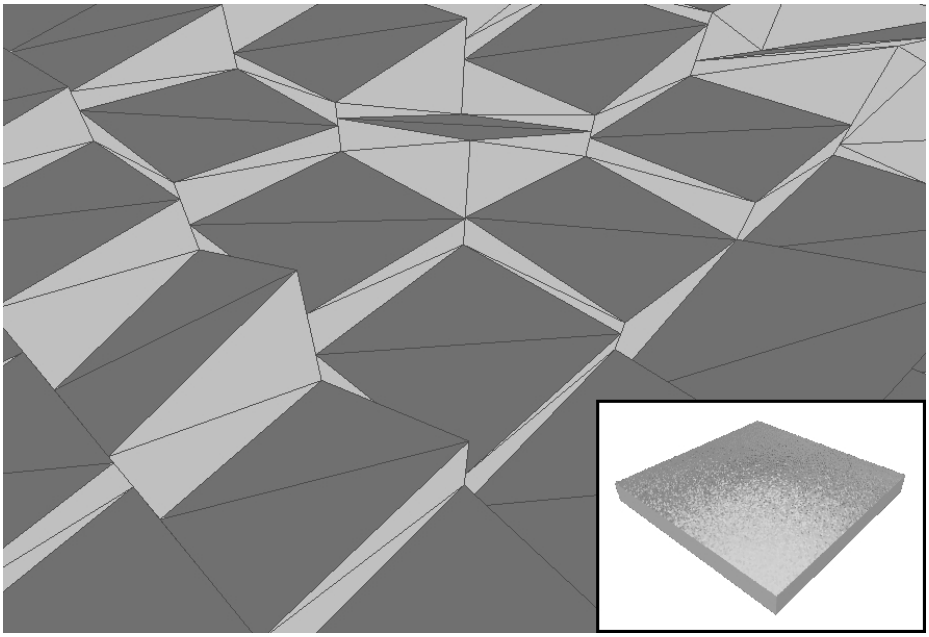


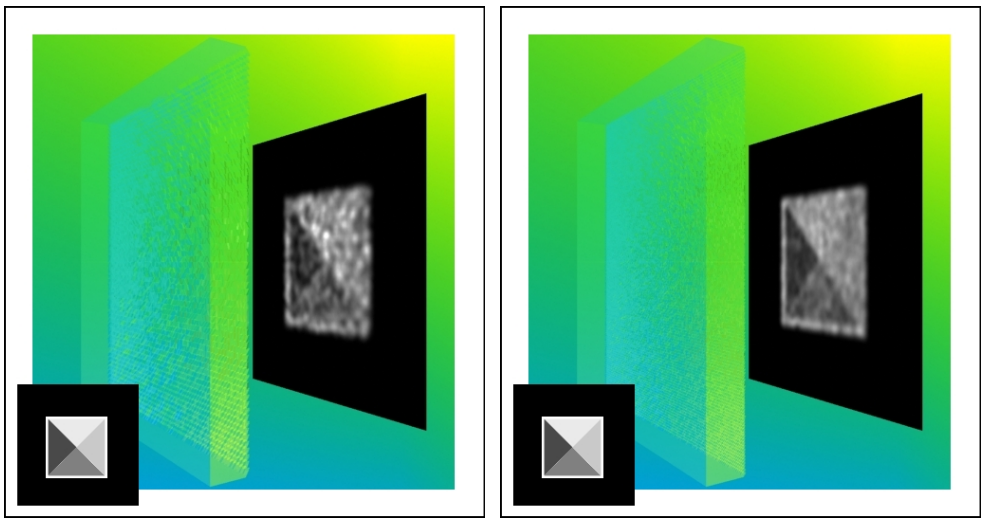
Figure 5.17: Caustic object geometry (inset) of Figure 5.10c with a zoom-in view. In the zoom-in view, each caustic object cell consists of two co-planar triangles shaded with darker grey. We also generate additional vertical polygons (shaded with lighter grey) to close the gaps between caustic object cells.

object cell is $1.0/128 \times 1.0/128$ and the size of each caustic cell is $1.0/64 \times 1.0/64$.

We use higher resolution for caustic object cells since we want to have more variations on the refracted light paths so that we can better reconstruct the contrast (or intensity variations) of the given caustic patterns. We show a comparison example with a case of a single caustic pattern in Figure 5.18, a simple caustic pattern (resolution 64×64) reconstructed with a resolution 64 x 64 caustic object and a resolution 128 x 128 caustic object.

We use the same parameters for the simulation annealing for both optimization steps in all experiments, i.e. 10 cycles of 10 iterations, Boltzmann's constant of 1.0, and temperature reduction factor of 0.5.

The experiments were performed on a PC with the specification Intel Xeon E5-1650 3.20 GHz, NVIDIA GTX 680 2 GB. In the implementations, we calculate the joint pmf by rendering each caustic pattern and then we multiply them by using alpha blending (hence the use of GPU). For the rest of the computations such as Simulated Annealing and Acceptance-Rejection method, we perform them on CPU.

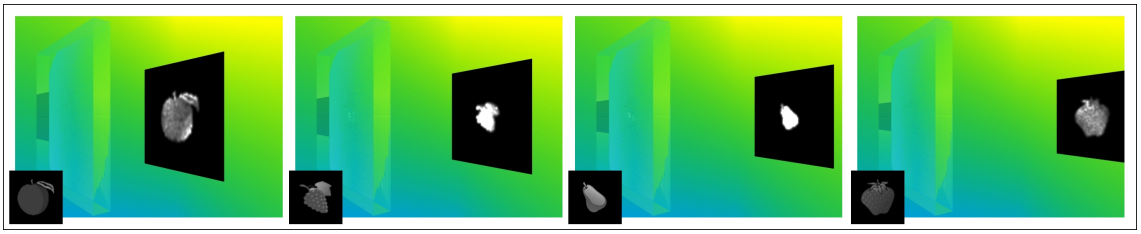


(a) (b)

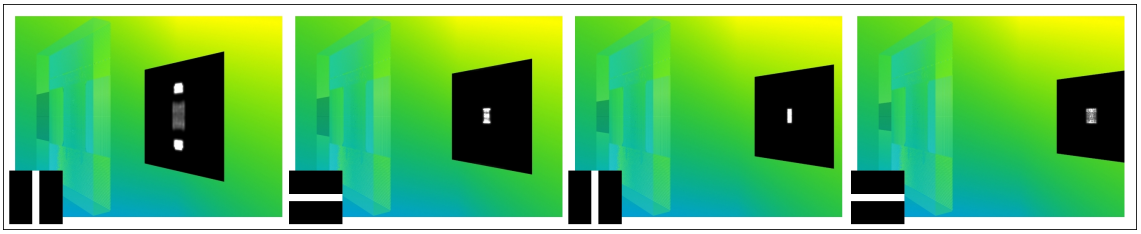
Figure 5.18: A simple test case (one caustic pattern, with resolution 64×64) reconstructed with different caustic object resolutions. (a) Resolution 64 x 64 caustic object. (b) Resolution 128 x 128 caustic object.

From the results, we can observe that the caustic objects generated using our technique can approximately reconstruct various types of input caustic patterns, especially for the WSCG (Figure 5.1b), fruits (Figure 5.19a), and rotating star (Figure 5.19c) test cases. The degree of difficulty in reconstructing the caustic patterns mostly depends on the number of caustic patterns, similarity between shapes, and the number of caustic cells in the input caustic patterns. As shown in Equation 5.5, due to the multiplication in computing the joint pmf, the probability of a particular refraction direction can become zero if the refracted light passes through empty cells. With the increasing number of caustic patterns especially the ones with different shapes, the chances that we have many refraction directions with zero probability also increase. We can see this from the results in Figures 5.1 and 5.10 (three input caustic patterns) where we can reconstruct better compared to the results in Figures 5.19 (four or more caustic patterns).

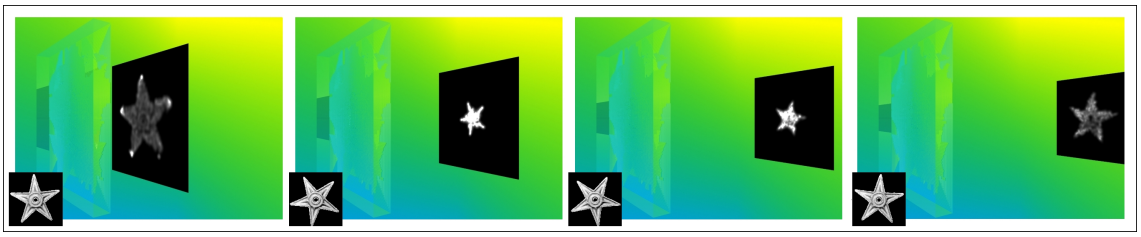
The shapes and orientations of caustic patterns can also affect the reconstruction difficulty. The test cases with similar caustic patterns, can be approximately reconstructed pretty well since similar refracted light paths are sufficient to reconstruct the caustic patterns. This is evident by comparing Figure 5.19a and Figure 5.19b. The caustic patterns in



(a) Fruits (four caustic patterns). Computational time : 5.86 hours.



(b) Four bars (four caustic patterns). Computational time : 18.96 hours.

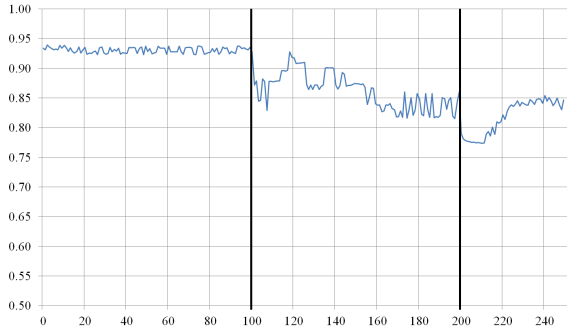


(c) Rotating Star (nine caustic patterns). Computational time : 29.80 hours.

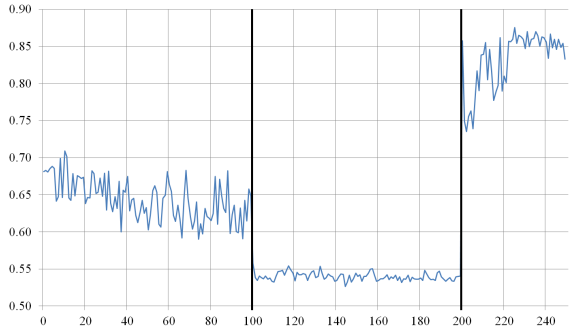
Figure 5.19: More results. Note that the caustic pattern set in (c) contain similar patterns, as they are frames of a simple animation.

Figure 5.19a have near round shapes and approximately the same orientations. In contrast, the test cases in Figure 5.19b have pretty different shapes (and orientations). Hence, there are few light that can pass through the endpoints of the bars compared to the middle regions of the bars. As we see in the cost history plot in Figure 5.20, both these two cases lie on the extreme vertical end of the plots.

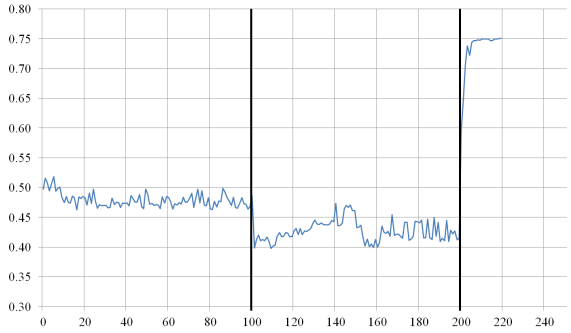
Note the difficult test case shown in Figure 5.19b, four bar caustic patterns with alternating orientations. As we can see, the hardest parts to reconstruct are the ones near the two endpoints of the bars and on the other hand the parts around the centres are the easiest. This is due to the alternating shapes which cause the light paths to always pass through the center of the caustic regions. As we allow the refracted light to miss some of the caustic patterns, we are able to approximately reconstruct the top and bottom parts of the first caustic pattern. However, the consequence is that the last caustic pattern appears much dimmer. From



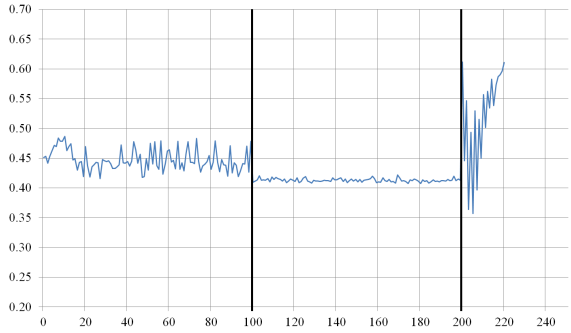
(a) Bars (Figure 5.19b)



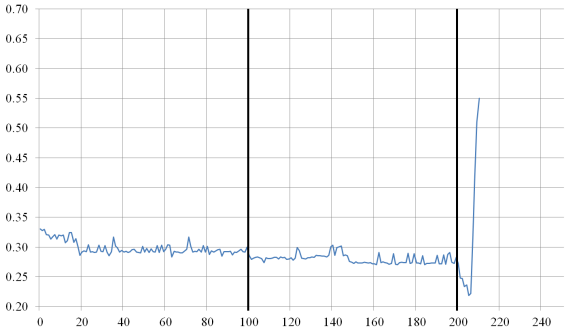
(b) WSCG - 2012 - Europe (Figure 5.1b)



(c) Rotating Star (Figure 5.19c)



(d) Bunny - Armadillo - Gargoyle (Figure 5.14)



(e) Fruits (Figure 5.19a)

Figure 5.20: Cost history of our experiments. Note the partitions in the 100th and 200th iterations which divide the cost history into first optimization step, second optimization step, and intensity correction.

the cost history plot in Figure 5.20, the second optimization step manages to bring the cost further lower. However, for the intensity correction step, it cannot achieve improved result for the test cases of WSCG (Figure 5.1b) and Star (Figure 5.19c).

In many cases, light tends to converge to the middle caustic patterns and as a result the centre regions of these caustic patterns become relatively brighter compared to the other caustic patterns (for example, the Armadillo caustic pattern in Figure 5.10 exhibits this

effect). This is due to two reasons. First, the caustic regions are positioned approximately at the centre of the caustic patterns. Second, because the size of the caustic regions are mostly smaller than the caustic object. Therefore, some caustic object cells have to refract the light in the diagonal directions. This is illustrated in Figure 5.5 in which some of the light grouped to $\#3$ have to be refracted in the diagonal directions.

Note that the relative depth of caustic patterns also affects the quality, as it is very difficult to reconstruct if the caustic patterns are located very near to each other. This is because the refracted light paths will intersect the patterns at similar locations and thus it is difficult to reconstruct caustic patterns with different shapes.

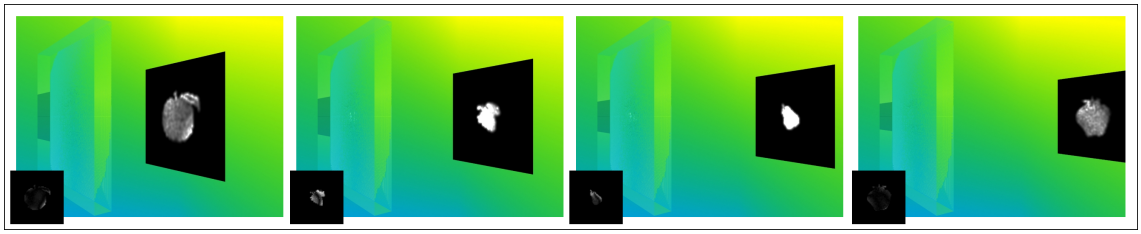
We performed several additional experiments in order to observe the stability of our proposed method. We computed caustic object for the input caustic patterns shown in Figure 5.19a for additional three times, and the results are shown in Figure 5.21. We can see that we are able to obtain similar results for Figures 5.21a, 5.21b, 5.21d, and a slightly different result for Figure 5.21c (the second and the third reconstructed caustic patterns look slightly rounder). From this experiment, we are able to reach fairly similar results in term of visual quality and final cost even though we do not reach the global minima (i.e. all the results have the same final cost).

5.6 Applications

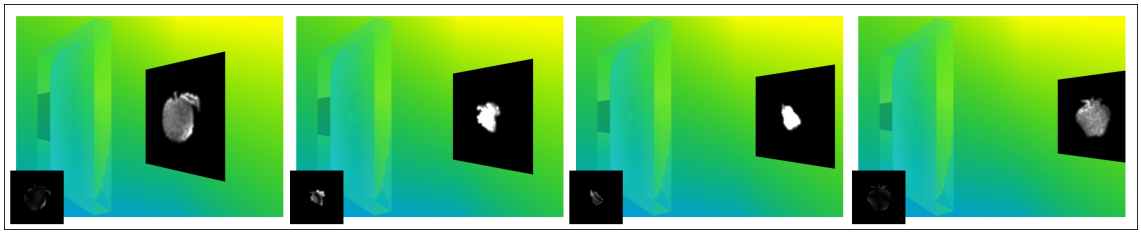
The inverse caustics has several potential applications.

Arts As shown in Figure 5.19, our technique can generate caustic objects that can produce several interesting caustic effects (similar to the intention in the inverse shadow [118]). Therefore, we hope that our work can encourage more exploration in caustic arts.

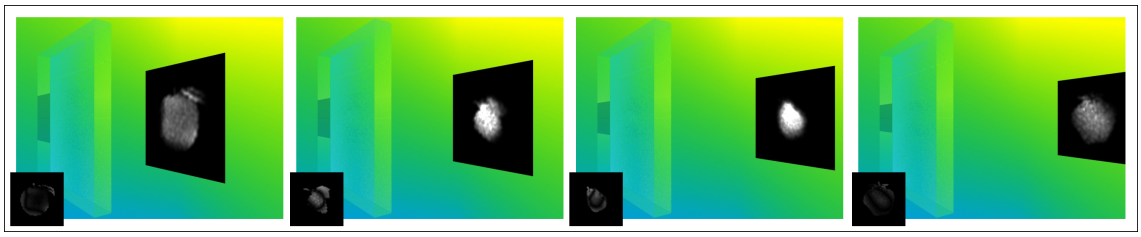
Information Encoding Information (such as serial numbers, passwords) can be encoded as caustic patterns of encrypted 2D images. Only when the requirements (such as light direction and caustic receiver distance) are known, we can recover the original information. We show an example in Figure 5.22 in which we encrypt **WSCG** and **2012** into two QR



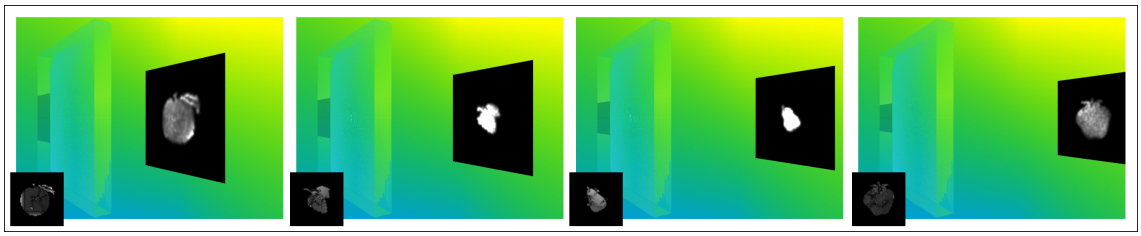
(a) Final Cost : 0.21845



(b) Final Cost : 0.23049



(c) Final Cost : 0.28532



(d) Final Cost : 0.24669

Figure 5.21: Additional experiments for the input caustic patterns shown in Figure 5.19a. We also show the irradiance difference maps at the lower left corner of each result. (a) The same result as the result in Figure 5.19a. (b-d) Additional experimental results.

barcode patterns.

Validation Tests By using the computed caustic object, we can validate some processes such as rendering process (validating the correctness of caustics rendering algorithm) or manufacturing (validating the quality of produced glasses or light sources).

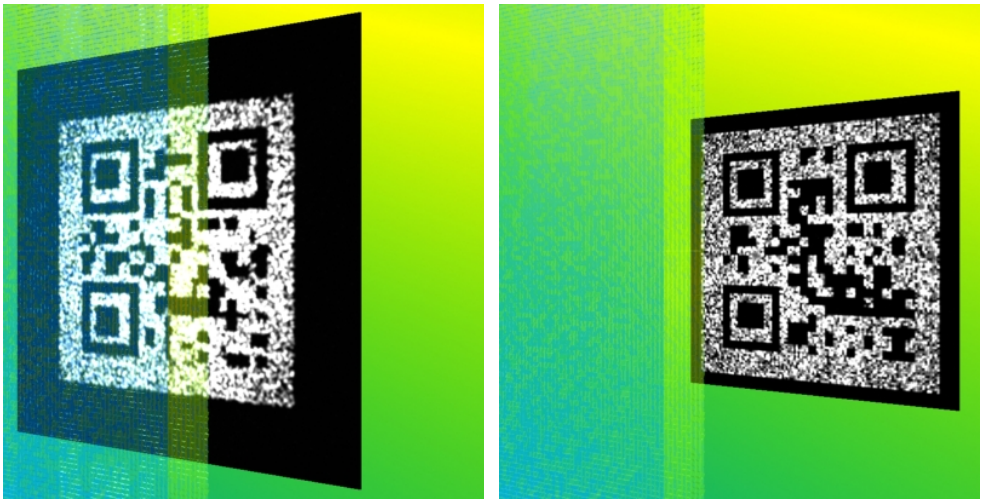


Figure 5.22: An application of information encoding. We encode **WSCG** and **2012** into two QR barcode patterns.

5.7 Summary

We have presented an inverse caustic problem and a novel technique which computes a caustic object given a set of caustic patterns with each pattern is positioned at a user-input distance from the caustic object. Our proposed technique is based on a stochastic approach, and it is augmented with two optimization steps that can alleviate the missing caustic problems. We also added another intensity correction step to improve the intensity of the reconstructed caustic patterns. We have validated our results by performing physically rendering simulation using mental ray, and the caustic object generated using our technique can approximately reconstruct various types of input caustic patterns.

In the future, we would like to improve the quality of the caustic object in terms of smoothness. It is also interesting to consider more complex light situations such as area light sources and dynamic light sources. It is challenging to use area light sources since they emit light to many directions from every point in the area light sources. As for the dynamic light sources, it is interesting to generate unique caustic pattern for each given light source direction (in this case, caustic object and caustic receiver are static). Last but not least, we would like to fabricate a real caustic object based on the computed geometry.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

We have proposed three solutions for the scalability problems in caustic computation, namely rendering caustics under environment illumination (multiple light sources), rendering caustics taking into account light's visible wavelengths, and caustic object construction based on several input caustic patterns. Our main idea in these solutions is adjustment of inputs such that the problems are tractable. For instance, we adjust the inputs by approximating or clustering the inputs (in the caustic rendering under environment illumination and spectral caustics rendering) in order to reduce the computation time that involves iteration on all the individual inputs. Another approach in our idea is adjusting the input parameters (in caustic object reconstruction work).

The main motivation of using this approach is to reduce the problem requirements such that it is still able to achieve the desired results with the compromises (adjustment of inputs) while preserving the quality of the results. For instance, in the caustic rendering under environment illumination and spectral rendering, integration of all the inputs will result in prohibiting rendering costs. Thus, we reduce the cost by grouping similar inputs and treat them as one input. For inverse caustics patterns, it is not possible to reduce the number of input caustic patterns, as it is our goal to reproduce those patterns. Hence, we adjust the

parameters of the input caustic patterns, that is we relax the constraint requirement.

Our proposed methods in the two caustic rendering works can accelerate the rendering pretty well. Our proposed methods consider the scene properties (e.g. light distribution and index of refraction). The computations in some other aspects, however, can still be improved in terms of implementation. For example, caustic spheres precomputation (in caustic rendering under environment illumination work) or caustic pattern computation of each wavelength clusters (in spectral caustic rendering work). Moreover, the degree of clustering or approximation depends on the specific inputs from the user (i.e. number of clustered environment light or threshold for wavelength clustering). Hence, our work can be improved by reducing the said manual user inputs.

By and large, we hope that our work can benefit the computer graphics, multimedia, and art communities. For instance, by using our work, it is possible to render caustics under environment illumination for game or walk through applications. Moreover, in offline rendering (e.g. movie production), the rendering of scenes that have caustic objects, spectral caustic effects can be accelerated yet at the same time the visual quality can be maintained. Our work on caustic object reconstruction opens a new possibility in caustic art and we hope that it can stimulate research in this area.

6.1.1 Caustic Rendering Under Environment Illumination

We have proposed an efficient method for rendering caustics under environment illumination. We precompute the caustic patterns of several directional light sources and we interpolate them in the rendering. In order to make the rendering under environment illumination feasible in real-time, we approximate it as a set of directional light sources by using our environment cube map segmentation technique.

The main strength of our work is the capability of generating real-time caustics under environment illumination. As we show in our experiment, by using the quadratic radii caustic spheres, our caustic pattern results are closer to the results generated by using mental

ray (an industry standard photorealistic rendering software) compared to using uniform radii caustic spheres. Our rendering technique supports dynamic scenes and it also takes into account light occlusion from the surrounding objects during the rendering. As we have also shown in our experiment results, our caustic occlusion effect results are comparable to the results generated by using mental ray.

The main limitation of our work is that it does not support deformable caustic object due to the precomputation nature. However, as we see in real-life, many caustic objects are not deformable such as glasses, crystals, etc. Another limitation is it requires a large amount of memory to store the precomputed caustic patterns.

6.1.2 Spectral Caustic Rendering

We have proposed a two-step scheme which accelerates spectral caustic rendering. Our acceleration technique takes into account various factors: index of refraction of caustic object, light power, material reflectance, human eyes' sensitivity, and geometry of the scene.

Our combined first and second acceleration steps is used in rendering with the Stochastic Progressive Photon Mapping. The experimental results show the strength of our work, i.e. the proposed method can significantly accelerate spectral caustic rendering while maintaining the rendering quality. We also compare an alternative of refinement computation by using importance level proposed by Radziszewski et al. [43], and we show that our proposed importance level is superior in terms of rendering speed.

The limitation of our proposed method is that it considers only a single type of caustic object material (only single IOR graph in the scene is considered during wavelength clustering) and single refraction (i.e. first refraction event when the light hits the caustic objects). However, as we show in our experiment, the effect of considering only single refraction is negligible as our rendering result is visually similar to the rendering result of the Brute Force technique. As shown in our experiment, our method can still handle a scene which has multiple caustic objects albeit all of them must have the same IOR graph.

6.1.3 Caustic Object Reconstruction

We have proposed a method which reconstructs a caustic object based on several input caustic patterns. In our work, we proposed two optimization steps which improve the shapes of the reconstructed caustic patterns. We also proposed an additional intensity correction step to improve the intensity of caustic patterns.

Our first and second optimization steps (which slightly adjust the position, size, and shape of each input caustic pattern) can improve the reconstruction of input caustic patterns with various shape differences. As caustic patterns have shapes and intensities, we also adjust the intensities of the reconstructed caustic patterns. Our results show that our intensity correction step can improve the results further. We have validated our results by performing physically-based rendering simulation using mental ray, and the caustic object generated using our technique can approximately reconstruct various types of input caustic patterns.

The limitations of our proposed methods are as follows. Our method requires a long computation time for optimization. It can be benefited by more optimized implementation. Moreover, for the case of input caustic patterns whose orientations are very different (e.g. alternating bar in Figure 5.19b), it is difficult to achieve a good result.

6.2 Future Work

6.2.1 Caustic Rendering of Deformable Caustic Objects

In our presented work in this thesis, our targets are rigid caustic objects (i.e. not deformable). Thus, for rendering caustics caused by deformable caustic objects (e.g. jelly, etc.), it is necessary to recompute everything for every rendering frame, which is not efficient. Some examples of related caustic rendering work presented in Section 2.3.2 such as work by Wyman et al. [78], Liu et al. [108], Hu and Qin [109], and Shah et al. [110] also require recomputation for every frame if their work is used to render deformable caustic object.

Weiss and Grosch [39] try to reduce the recomputation by keeping tracks of photon paths for each frame. Unfortunately, their approach requires a priori knowledge of the whole animation sequence.

Hence, we are interested in investigating possible ways in handling deformable caustic objects with less precomputation or recomputation. The followings are the possible approaches for solving this problem:

- Precompute the caustic patterns of the caustic object in several deformation states. During the rendering, when we deform the caustic object between the predefined deformation states, we interpolate their respective caustic patterns.

- Analyse the deformation motion and update the photon map accordingly.

6.2.2 Spectral Caustic Rendering in Dynamic Scenes

Our proposed spectral rendering acceleration scheme is mainly applied for static scenes. If we intend to render animation frames, we have to recompute the caustic patterns of every wavelength cluster.

After rendering the full spectrum caustics for a given scene configuration (such as surrounding objects and caustic objects positions and orientations, camera position and orientation, light position and types) by using our acceleration scheme, we obtain some rendering statistics and information that can be used for subsequent rendering if the scene configuration is pretty much the same. The followings are the possible approaches:

- **Scene motion** To handle scenes with moving objects, we can improve upon the work by Weiss and Grosch [39], by analysing the photon paths for each clustered wavelength. It is also possible to extend it further by reusing the statistics of visible points of preceding frames.

- **Deformable caustic object** We can use some approaches presented in Section 6.2.1 to render spectral caustics caused by deformable caustic objects.

- **Relighting** Given the photon distribution of all processed wavelengths, we can also perform relighting by changing the Spectral Power Distribution (SPD) of the light source.

6.2.3 Physical Construction of Caustic Object

The caustic object generated by our inverse caustics has rough edges which is difficult to fabricate with low cost. One way to lower the fabrication cost is by smoothing the surface. The existing solutions try to smooth the surface by swapping two caustic object cells while maintaining their respected refraction targets [128, 127]. However, in our case, it is not exactly feasible since we have multiple caustic patterns. If we swap two caustic object cells while keeping their refraction targets on one of the caustic patterns to be the same, their refracted light might hit other parts of the other caustic patterns.

To solve this issue, we may consider an iterative approach for smoothing the caustic object surface. In this approach, we try to reduce the cost of local cell orientation difference. The changing of cell orientations in this step will most likely affect the reconstructed caustic patterns. Hence, we must introduce additional cost, i.e. the similarity of the reconstructed caustic patterns before and after changing the cell orientations.

For each cell in each iteration, we compute its local cost $Cost_N$ which is the one minus average of normal difference between the cell and its four neighbouring cells. We also compute the global error $Cost_P$ by using Equation 5.3. Both $Cost_N$ and $Cost_P$ are weighted by user defined parameters, i.e. α_N and α_P respectively (as shown in Equation 6.1).

$$Cost_S = \alpha_N Cost_N + \alpha_P Cost_P, \tag{6.1}$$

where $Cost_S$ is the total cost for the particular cell we are processing. After obtaining the relatively smooth surface in terms of normal field or cell orientations, the next task is computing the height field or the height of each caustic object cells such that the amount of rough edges can be reduced further. There are some existing techniques to solve this

problem, such as by solving Eikonal equation [153] or Poisson problem [128]. We are interested in performing some experiments by using these approaches to generate caustic object models that are more viable for physical construction.

Bibliography

[1] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 137–145. ACM, 1984.

[2] Gregory J. Ward, Francis M. Rubinstein, and Robert D. Clear. A ray tracing solution for diffuse interreflection. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, pages 85–92. ACM, 1988.

[3] Henrik W. Jensen. *The photon map in global illumination*. PhD thesis, Technical University of Denmark, 1996.

[4] Nvidia advanced rendering: Nvidia mental ray. `http://www.nvidia-arc.com/mentalray.html`. Date accessed : 2015-04-13.

[5] Gorm Lai and Niels J. Christensen. A compression method for spectral photon map rendering. In *15th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2007, WSCG'2007 - In Co-operation with EUROGRAPHICS, Full Papers WSCG Proceedings*, pages 95–102, Plzen, 2007.

[6] Toshiya Hachisuka. Toshiya Hachisuka at UTokyo. `http://www.ci.i.u-tokyo.ac.jp/~hachisuka/`. Date accessed : 2014-09-18.

[7] Toshiya Hachisuka and Henrik W. Jensen. Stochastic progressive photon mapping. *ACM Transactions on Graphics*, 28(5):141:1–141:8, December 2009.

[8] Useful color data. http://www.cis.rit.edu/research/mcsl2/online/cie.php. Date accessed : 2013-04-06.

[9] Robert Hooke and T. A. Jeeves. “ Direct search” solution of numerical and statistical problems. *Journal of the ACM*, 8(2):212–229, April 1961.

[10] Scott Kirkpatrick, Charles D. Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[11] Solomon Boulos, Dave Edwards, Jesse D. Lacewell, Joe Kniss, Jan Kautz, Peter Shirley, and Ingo Wald. Packet-based whitted and distribution ray tracing. In *Proceedings of Graphics Interface 2007*, GI ’07, page 177–184, New York, NY, USA, 2007. ACM.

[12] James T. Kajiya. The rendering equation. In *SIGGRAPH ’86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150. ACM, 1986.

[13] Paul S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. In *SIGGRAPH ’90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 145–154, New York, NY, USA, 1990. ACM.

[14] Arthur Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pages 37–45, Atlantic City, New Jersey, 1968. ACM.

[15] Turner Whitted. Seminal graphics. chapter An Improved Illumination Model for Shaded Display, pages 119–125. ACM, New York, NY, USA, 1998.

[16] Eric P. Lafortune and Yves D. Willems. Bi-Directional path tracing. *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (COMPUGRAPHICS) ’93*, pages 145—153, 1993.

[17] Eric Veach and Leonidas J. Guibas. Bidirectional estimators for light transport. In *Eurographics Rendering Workshop 1994 Proceedings*, page 145–167, June 1994.

[18] Eric Veach and Leonidas J. Guibas. Metropolis light transport. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 65–76. ACM Press/Addison-Wesley Publishing Co., 1997.

[19] Akira Fujimoto, Takayuki Tanaka, and Kansei Iwata. Arts: Accelerated ray-tracing system. *IEEE Computer Graphics and Applications*, 6(4):16 –26, April 1986.

[20] Vlastimil Havran. *Heuristic Ray Shooting Algorithms*. Ph.D. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, November 2000.

[21] Carsten Wächter and Alexander Keller. Instant ray tracing: The bounding interval hierarchy. *In Rendering Techniques 2006 - Proceedings of the 17th Eurographics Symposium on Rendering*, pages 139—149, 2006.

[22] Henrik W. Jensen and Per H. Christensen. Efficient simulation of light transport in scenes with participating media using photon maps. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 311–320. ACM, 1998.

[23] Ingmar Peter and Georg Pietrek. Importance driven construction of photon maps. In *In Rendering Techniques 1998 - Proceedings of the 9th Eurographics Workshop on Rendering*, pages 269–280. Springer-Verlag, 1998.

[24] Lars Schjøth, Jeppe R. Frisvad, Kenny Erleben, and Jon Sporring. Photon differentials. In *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, pages 179–186, Perth, Australia, 2007. ACM.

[25] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Progressive photon mapping. *ACM Transactions on Graphics*, 27(5):130:1–130:8, December 2008.

[26] Timothy J. Purcell, Craig Donner, Mike Cammarano, Henrik W. Jensen, and Pat Hanrahan. Photon mapping on programmable graphics hardware. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, page 258, New York, NY, USA, 2005. ACM.

[27] Ben Spencer and Mark W. Jones. Hierarchical photon mapping. *IEEE Transactions on Visualization and Computer Graphics*, 15(1):49 –61, February 2009.

[28] Brian C. Budge, John C. Anderson, and Kenneth I. Joy. Caustic forecasting: Unbiased estimation of caustic lighting for global illumination. *Computer Graphics Forum*, 27(7):1963–1970, 2008.

[29] Rui Wang, Rui Wang, Kun Zhou, Minghao Pan, and Hujun Bao. An efficient gpu-based approach for interactive global illumination. *ACM Transactions on Graphics*, 28(3):91:1–91:8, July 2009.

[30] Tsung-Chih Chen, Lieu-Henand Tsai and Yu-Sheng Chen. Grouped photon mapping. *The Visual Computer*, 26(3):217–226, March 2010.

[31] Michael Mara, David Luebke, and Morgan McGuire. Toward practical real-time photon mapping: efficient GPU density estimation. In *Proceedings of the 2013 Symposium on Interactive 3D Graphics and Games*, page 71–78, New York, NY, USA, 2013. ACM.

[32] Johannes Günther, Ingo Wald, and Philipp Slusallek. Realtime caustics using distributed photon mapping. In Alexander Keller and Henrik Wann Jensen, editors, *Rendering Techniques 2004 : Eurographics Symposium on Rendering*, pages 111–121, Norkoeping, Sweden, June 2004. Eurographics Association, Eurographics.

[33] Zhe Fu and Henrik W. Jensen. Noise reduction for progressive photon mapping. In *ACM SIGGRAPH 2012 Talks*, SIGGRAPH '12, page 29:1–29:1, New York, NY, USA, 2012. ACM.

[34] Jiating Chen, Bin Wang, and Jun-Hai Yong. Improved stochastic progressive photon mapping with metropolis sampling. *Computer Graphics Forum*, 30(4):1205–1213, 2011.

[35] Alexander Keller, Leonhard Grünschloß, and Marc Droske. Quasi-monte carlo progressive photon mapping. In Leszek Plaskota and Henryk Woźniakowski, editors, *Monte Carlo and Quasi-Monte Carlo Methods 2010*, volume 23, pages 499–509. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[36] Toshiya Hachisuka, Wojciech Jarosz, and Henrik W. Jensen. A progressive error estimation framework for photon density estimation. *ACM Transactions on Graphics*, 29(6):1, December 2010.

[37] Claude Knaus and Matthias Zwicker. Progressive photon mapping: A probabilistic approach. *ACM Transactions on Graphics*, 30(3):25:1–25:13, May 2011.

[38] Anton S. Kaplanyan and Carsten Dachsbacher. Adaptive progressive photon mapping. *ACM Transactions on Graphics*, 32(2):16:1–16:13, April 2013.

[39] Maayan Weiss and Thorsten Grosch. Stochastic progressive photon mapping for dynamic scenes. *Computer Graphics Forum*, 31(2pt3):719–726, 2012.

[40] Toshiya Hachisuka and Henrik W. Jensen. Robust adaptive photon tracing using photon path visibility. *ACM Transactions on Graphics*, 30(5):114:1–114:11, October 2011.

[41] Wojciech Jarosz, Derek Nowrouzezahrai, Robert Thomas, Peter-Pike Sloan, and Matthias Zwicker. Progressive photon beams. In *Proceedings of the 2011 SIG-*

GRAPH Asia Conference, SIGGRAPH ASIA '11, page 181:1–181:12, New York, NY, USA, 2011. ACM.

[42] Garrett Johnson and Mark D. Fairchild. Full-spectral color calculations in realistic image synthesis. *IEEE Computer Graphics and Applications*, 19(4):47 –53, August 1999.

[43] Michal Radziszewski, Krzysztof Boryczko, and Witold Alda. An improved technique for full spectral rendering. *Journal of WSCG*, 17(1-3):9–16, 2009.

[44] Jinjin Shi, Dengming Zhu, Yingping Zhang, and Zhaoqi Wang. Realistically rendering polluted water. *The Visual Computer*, 28(6-8):647–656, June 2012.

[45] Weiming Dong. Rendering optical effects based on spectra representation in complex scenes. In *Advances in Computer Graphics*, volume 4035. Springer Berlin Heidelberg, Berlin, Heidelberg.

[46] Jay S. Gondek, Gary W. Meyer, and Jonathan G. Newman. Wavelength dependent reflectance functions. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 213–220, New York, NY, USA, 1994. ACM.

[47] Hideaki Hirayama, Kazufumi Kaneda, Hideo Yamashita, Yoshiki Yamaji, and Yoshimi Monden. Visualization of optical phenomena caused by multilayer films with complex refractive indices. In *In Proceedings of the Seventh Pacific Conference on Computer Graphics and Applications (PG '99)*, Pacific Graphics '99, pages 128–137, 320, 1999.

[48] Y. Sun, Frank D. Fracchia, Thomas W. Calvert, and Mark S. Drew. Deriving spectra from colors and rendering light interference. *IEEE Computer Graphics and Applications*, 19(4):61–67, 1999.

[49] Kei Iwasaki, Keichi Matsuzawa, and Tomoyuki Nishita. Real-time rendering of soap bubbles taking into account light interference. In *Proceedings of the Computer Graphics International*, pages 344–348. IEEE Computer Society, 2004.

[50] Roman Ďurikovič and Ryou Kimura. Spectrum-based rendering using programmable graphics hardware. In *Proceedings of the 21st spring conference on Computer graphics*, pages 233–236, Budmerice, Slovakia, 2005. ACM.

[51] Roman Durikovic and Ryou Kimura. GPU rendering of the thin film on paints with full spectrum. In *Information Visualization, 2006. IV 2006. Tenth International Conference on*, 2006.

[52] Yinlong Sun. Rendering biological iridescences with RGB-based renderers. *ACM Transactions on Graphics*, 25(1):100–129, 2006.

[53] Yinlong Sun and Qiqi Wang. Interference shaders of thin films. *Computer Graphics Forum*, 27(6):1607–1631, 2008.

[54] Jos Stam. Diffraction shaders. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 101–110. ACM Press/Addison-Wesley Publishing Co., 1999.

[55] Emmanuel O. Agu. *Diffraction shading models in computer graphics*. PhD thesis, University of Massachusetts Amherst, September 2001.

[56] Masataka Imura, Takaaki Abe, Ichiroh Kanaya, Yoshihiro Yasumuro, Yoshitsugu Manabe, and Kunihiro Chihara. Rendering of 'play of color' using stratified model based on amorphous structure of opal. *Proceedings VIIth Digital Image Computing: Techniques and Applications*, December 2003.

[57] Jiaze Wu, Changwen Zheng, Xiaohui Hu, and Fanjiang Xu. Rendering realistic spectral bokeh due to lens stops and aberrations. *The Visual Computer*, 29(1):41–52, January 2013.

[58] Forrest K. Musgrave. Prisms and rainbows: a dispersion model for computer graphics. In *Proceedings of Graphics Interface*, volume 89, pages 227–234, 1989.

[59] Iman Sadeghi, Adolfo Munoz, Philip Laven, Wojciech Jarosz, Francisco Seron, Diego Gutierrez, and Henrik W. Jensen. Physically-based simulation of rainbows. *ACM Transactions on Graphics*, 31(1):3:1–3:12, February 2012.

[60] Xiaoxiong Xing, Weiming Dong, Xiaopeng Zhang, and Jean-Claude Paul. Spectrally-based single image relighting. In *Proceedings of the Entertainment for education, and 5th international conference on E-learning and games*, Edutainment'10, pages 509–517, Berlin, Heidelberg, 2010. Springer-Verlag.

[61] Steven Bergner, Torsten Möller, Mark S. Drew, and Graham D. Finlayson. Interactive spectral volume rendering. In *Proceedings of the conference on Visualization '02*, pages 101–108, Boston, Massachusetts, 2002. IEEE Computer Society.

[62] Magnus Strengert, Thomas Klein, Ralf Botchen, Simon Stegmaier, Min Chen, and Thomas Ertl. Spectral volume rendering using GPU-based raycasting. *The Visual Computer*, 22(8):550–561, June 2006.

[63] David H. Marimont and Brian A. Wandell. Linear models of surface and illuminant spectra. *Journal of the Optical Society of America A*, 9(11):1905–1913, November 1992.

[64] Jin-Ren Chern and Chung-Ming Wang. A novel progressive refinement algorithm for full spectral rendering. *Real-Time Imaging*, 11(2):117–127, April 2005.

[65] Gilles Rougeron, Bernard Péroche, and Centre Simade Lisse. An adaptive representation of spectral data for reflectance computations. In *Eurographics Rendering Workshop*, pages 127–138. Springer, 1997.

[66] Huiying Xu and Yinlong Sun. Compact representation of spectral BRDFs using fourier transform and spherical harmonic expansion. *Computer Graphics Forum*, 25(4):759–775, 2006.

[67] Eric Zeghers, Samuel Carré, and Kadi Bouatouch. Error-bound wavelength selection for spectral rendering. *The Visual Computer*, 13(9-10):424–434, January 1998.

[68] Jean C. Iehl and Bernard Péroche. Adaptive spectral rendering with a perceptual control. *Computer Graphics Forum*, 19(3):291–299, 2000.

[69] Gary W. Meyer. Wavelength selection for synthetic image generation. *Computer Vision, Graphics, and Image Processing*, 41(1):57–79, January 1988.

[70] Pascal M. Deville, Slimane Merzouk, Dorothée Cazier, and Jean C. Paul. Spectral data modeling for a lighting application. *Computer Graphics Forum*, 13(3):97–106, 1994.

[71] Yinlong Sun, Frank D. Fracchia, and Mark S. Drew. A composite spectral model and its applications. *Color and Imaging Conference*, 2000(1):102–107, 2000.

[72] Yinlong Sun, Frank D. Fracchia, Mark S. Drew, and Thomas W. Calvert. A spectrally based framework for realistic image synthesis. *The Visual Computer*, 17(7):429–444, October 2001.

[73] Glenn F. Evans and Micheal D. McCool. Stratified wavelength clusters for efficient spectral monte carlo rendering. In *Proceedings of the Conference on Graphics interface '99*, page 42–49, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[74] Jean C. Iehl and Bernard Péroche. Towards perceptual control of physically based spectral rendering. *Computers and Graphics (Pergamon)*, 27(5):747–762, 2003.

[75] Brian Smits. An RGB-to-spectrum conversion for reflectances. *Journal of Graphic Tools*, 4(4):11–22, December 1999.

[76] Robert Wallis. Fast computation of tristimulus values by use of gaussian quadrature. *Journal of the Optical Society of America*, 65(1):91–94, January 1975.

[77] Oskar Elek, Pablo Bauszat, Tobias Ritschel, Marcus Magnor, and Hans-Peter Seidel. Spectral ray differentials. *Computer Graphics Forum (Proceedings of EGSR)*, 33(4), 2014.

[78] Chris Wyman and Scott T. Davis. Interactive image-space techniques for approximating caustics. In *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*, pages 153–160, New York, NY, USA, 2006. ACM.

[79] Kate Devlin, Alan Chalmers, Alexander Wilkie, and Werner Purgathofer. STAR: tone reproduction and physically based spectral rendering. In Dieter Fellner and Roberto Scopignio, editors, *State of the Art Reports, Eurographics 2002*, pages 101–123. The Eurographics Association, September 2002.

[80] Henrik W. Jensen. Rendering caustics on non-Lambertian surfaces. In *Proceedings of the conference on Graphics interface '96*, pages 116–121, Toronto, Ontario, Canada, 1996. Canadian Information Processing Society.

[81] Ben Spencer and Mark W. Jones. Into the blue: Better caustics through photon relaxation. *Computer Graphics Forum*, 28(2):319–328, 2009.

[82] Ian C. Doidge, Mark W. Jones, and Benjamin Mora. Mixing monte carlo and progressive rendering for improved global illumination. *The Visual Computer*, 28(6-8):603–612, April 2012.

[83] Iliyan Georgiev, Jaroslav Křivánek, Tomáš Davidovič, and Philipp Slusallek. Light transport simulation with vertex connection and merging. *ACM Transactions on Graphics*, 31(6):192:1–192:10, November 2012.

[84] Toshiya Hachisuka, Jacopo Pantaleoni, and Henrik W. Jensen. A path space extension for robust light transport simulation. *ACM Transactions on Graphics*, 31(6):191:1–191:10, November 2012.

[85] Lionel Baboud and Xavier Decoret. Realistic water volumes in Real-Time. page 25–32, Vienna, Austria, 2006. Eurographics Association.

[86] Cem Yuksel and John Keyser. Fast real-time caustics from height fields. *The Visual Computer*, 25(5):559–564, May 2009.

[87] Chris Trendall and Alan J. Stewart. General calculations using graphics hardware with applications to interactive caustics. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000*, pages 287–298. Springer-Verlag, 2000.

[88] Jens Krüger, Kai Bürger, and Rüdiger Westermann. Interactive screen-space accurate photon tracing on GPUs. In *Rendering Techniques - Eurographics Symposium on Rendering)*, pages 319–329, June 2006.

[89] Xin Sun, Kun Zhou, Stephen Lin, and Baining Guo. Line space gathering for single scattering in large scenes. *ACM Transactions on Graphics*, 29(4):54:1–54:8, July 2010.

[90] Hu Wei, Dong Zhao, Ivo Ihrke, Thorsten Grosch, Guodong Yuan, and Hans-Peter Seidel. Interactive volume caustics in single-scattering media. In *Proceedings of the 2010 symposium on Interactive 3D Graphics and Games*, pages 109–117, Washington, D.C., 2010. ACM.

[91] Mark Watt. Light-water interaction using backward beam tracing. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pages 377–385, Dallas, TX, USA, 1990. ACM.

[92] Tomoyuki Nishita and Eihachiro Nakamae. Method of displaying optical effects within water using accumulation buffer. In *SIGGRAPH '94: Proceedings of the 21st*

Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '94, page 373–379, New York, NY, USA, 1994. ACM.

[93] Kei Iwasaki, Yoshinori Dobashi, and Tomoyuki Nishita. An efficient method for rendering underwater optical effects using graphics hardware. *Computer Graphics Forum*, 21(4):701–711, 2002.

[94] Kei Iwasaki, Yoshinori Dobashi, and Tomoyuki Nishita. A fast rendering method for refractive and reflective caustics due to water surfaces. *Computer Graphics Forum*, 22(3):601–609, 2003.

[95] Manfred Ernst, Tomas Akenine-Möller, and Henrik W. Jensen. Interactive rendering of caustics using interpolated warped volumes. In *GI '05: Proceedings of Graphics Interface 2005*, pages 87–96, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2005. Canadian Human-Computer Communications Society.

[96] Pizzanu Kanongchaiyos. An interactive method for refractive water caustics rendering using color and depth textures. In *Proceedings of the First International Conference on Computer Graphics Theory and Applications*, pages 392–399, Setúbal, Portugal, 2006.

[97] Bo Qin, Lina Cui, and Cui Xie. Fast rendering algorithm of underwater optical effect. In *Proceedings of the 2nd International Symposium on Intelligent Information Technology and Security Informatics, IITSI 2009*, pages 171–174, 2009.

[98] Gabor Liktor and Carsten Dachsbacher. Real-Time volumetric caustics with projected light beams. In *Proceedings of 5th Hungarian Conference on Computer Graphics and Geometry*, pages 12–18, SZTAKI, Budapest, 2010.

[99] Michael Wand and Wolfgang Straßer. Real-time caustics. *Computer Graphics Forum*, 22(3):611–620, 2003.

[100] Kei Iwasaki, Fujiichi Yoshimoto, Yoshinori Dobashi, and Tomoyuki Nishita. A rapid rendering method for caustics arising from refraction by transparent objects. In *Cyberworlds, 2004 International Conference on*, pages 39–44, 2004.

[101] Kei Iwasaki, Fujiichi Yoshimoto, Yoshinori Dobashi, and Tomoyuki Nishita. A fast rendering technique of transparent objects and caustics. In *CASA '05 : Proceedings of Computer Animation and Social Agents 2005*, pages 165–170, 2005.

[102] Kei Iwasaki, Fujiichi Yoshimoto, Yoshinori Dobashi, and Tomoyuki Nishita. A method for fast rendering of caustics from refraction by transparent objects. *IEICE Transactions on Information and Systems*, E88-D(5):904–911, 2005.

[103] Chris Wyman, Charles Hansen, and Peter Shirley. Interactive caustics using local precomputed irradiance. In *Proceedings of the twelfth Pacific Conference on Computer Graphics and Applications (PG '04)*, pages 143–151, Washington, DC, USA, 2004. IEEE Computer Society.

[104] Xinguo Liu, Zhao Dong, Hujun Bao, and Qunsheng Peng. Caustic spot light for rendering caustics. *The Visual Computer*, 24(7-9):485–494, 2008.

[105] Chris Wyman. An approximate image-space approach for interactive refraction. *ACM Transactions on Graphics*, 24(3):1050–1053, July 2005.

[106] Chris Wyman. Interactive image-space refraction of nearby geometry. In *GRAPHITE '05: Proceedings of the 3rd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, pages 205–211, New York, NY, USA, 2005. ACM.

[107] Chris Wyman and Scott T. Davis. Interactive refractions with total internal reflection. In *Proceedings of Graphics Interface 2007*, pages 185–190, Montreal, Canada, 2007. ACM.

[108] Baoquan Liu, Enhua Wu, and Xuehui Liu. Interactively rendering dynamic caustics on GPU. In *Advances in Computer Graphics*, volume Volume 4035/2006 of *Lecture Notes in Computer Science*, pages 136–147. Springer Berlin / Heidelberg, 2006.

[109] Wei Hu and Kaihuai Qin. Interactive approximate rendering of reflections, refractions, and caustics. *IEEE Transactions on Visualization and Computer Graphics*, 13(1):46–57, 2007.

[110] Musawir A. Shah and Jaakko Konttinen. Caustics mapping: an image-space technique for real-time caustics. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):272–280, 2007.

[111] Tamás Umenhoffer, Gustavo Hou, and László Szirmay-Kalos. Caustic triangles on the GPU. In *Computer Graphics International 2008*, pages 222–228, 2008.

[112] Christ Wyman and Carsten Dachsbacher. Improving image-space caustics via variable-sized splatting. *Journal of Graphics Tools*, 13(1):1–17, 2008.

[113] Chris Wyman. Hierarchical caustic maps. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games*, pages 163–171, New York, NY, USA, 2008. ACM.

[114] Chris Wyman and Greg Nichols. Adaptive caustic maps using deferred shading. *Computer Graphics Forum*, 28(2):309–318, 2009.

[115] Ivo Ihrke, Gernot Ziegler, Art Tevs, Christian Theobalt, Marcus Magnor, and Hans-Peter Seidel. Eikonal rendering: Efficient light transport in refractive objects. *ACM Transactions on Graphics*, 26(3), July 2007.

[116] Xin Sun, Kun Zhou, Eric Stollnitz, Jiaoying Shi, and Baining Guo. Interactive relighting of dynamic refractive objects. *ACM Transactions on Graphics*, 27(3):35:1–35:9, August 2008.

[117] Andrea Bottino, L. Cavallero, and Aldo Laurentini. Interactive reconstruction of 3-D objects from silhouettes. In *WSCG*, pages 230–236, 2001.

[118] Niloy J. Mitra and Mark Pauly. Shadow art. *ACM Transactions on Graphics*, 28:156:1–156:7, December 2009.

[119] Gustavo Patow, Xavier Pueyo, and Alvar Vinacua. Reflector design from radiance distributions. *World Scientific*, 10(2):211–235, 2004.

[120] Gustavo Patow, Xavier Pueyo, and Alvar Vinacua. User-guided inverse reflector design. *Computers And Graphics*, 31(3):501–515, June 2007.

[121] Albert Mas, Ignacio Martín, and Gustavo Patow. Fast inverse reflector design (FIRD). *Computer Graphics Forum*, 28(8):2046–2056, December 2009.

[122] Vladimir Oliker. Optical design of freeform two-mirror beam-shaping systems. *Journal of the Optical Society of America A*, 24(12):3741–3752, December 2007.

[123] Steven Doyle, David Corcoran, and Jon Connell. Automated mirror design using an evolution strategy. *Optical Engineering*, 38(2):323–333, February 1999.

[124] Oscar Anson, Francisco J. Seron, and Diego Gutierrez. NURBS-Based inverse reflector design. In *CEIG08: Congreso Español de Informática Gráfica*, pages 65–74, Barcelona, Spain, 2008. Eurographics Association.

[125] Andreas Neubauer. Design of 3D-Reflectors for near field and far field problems. In Lorenz T. Biegler, Thomas F. Coleman, Andrew R. Conn, and Fadil N. Santosa, editors, *Large-Scale Optimization with Applications*, number 92 in The IMA Volumes in Mathematics and its Applications, pages 101–118. Springer New York, January 1997.

[126] Manuel Finckh, Holger Dammertz, and Hendrik P. A. Lensch. Geometry construction from caustic images. In *Proceedings of the 11th European Conference on Com-*

puter Vision: Part V, ECCV'10, pages 464–477, Berlin, Heidelberg, 2010. Springer-Verlag.

[127] Marios Papas, Wojciech Jarosz, Wenzel Jakob, Szymon Rusinkiewicz, Wojciech Matusik, and Tim Weyrich. Goal-based caustics. *Computer Graphics Forum*, 30(2):503–511, 2011.

[128] Tim Weyrich, Pieter Peers, Wojciech Matusik, and Szymon Rusinkiewicz. Fabricating microgeometry for custom surface reflectance. *ACM Transactions on Graphics*, 28:32:1–32:6, July 2009.

[129] Thomas Kiser, Michael Eigensatz, Minh Man Nguyen, Philippe Bompas, and Mark Pauly. Architectural caustics — controlling light with geometry. In Lars Hesselgren, Shrikant Sharma, Johannes Wallner, Niccolo Baldassini, Philippe Bompas, and Jacques Raynaud, editors, *Advances in Architectural Geometry 2012*, pages 91–106. Springer Vienna, January 2013.

[130] Yonghao Yue, Kei Iwasaki, Bing-Yu Chen, Yoshinori Dobashi, and Tomoyuki Nishita. Poisson-based continuous surface generation for goal-based caustics. *ACM Transactions on Graphics*, 33(3):31:1–31:7, June 2014.

[131] Yonghao Yue, Kei Iwasaki, Bing-Yu Chen, Yoshinori Dobashi, and Tomoyuki Nishita. Pixel art with refracted light by rearrangeable sticks. *Computer Graphics Forum*, 31(2):575–582, 2012.

[132] Marios Papas, Thomas Houit, Derek Nowrouzezahrai, Markus Gross, and Wojciech Jarosz. The magic lens: refractive steganography. *ACM Transactions on Graphics*, 31(6):186:1–186:10, November 2012.

[133] Henrik W. Jensen. Global illumination using photon maps. In *Proceedings of the Eurographics Workshop on Rendering Techniques '96*, pages 21–30, London, UK, 1996. Springer-Verlag.

[134] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 527–536, New York, NY, USA, July 2002. ACM.

[135] Lance Williams. Casting curved shadows on curved surfaces. In *SIGGRAPH '78: Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*, pages 270–274, New York, NY, USA, 1978. ACM.

[136] Markus Hadwiger, Patric Ljung, Christof Rezk Salama, and Timo Ropinski. Advanced illumination techniques for GPU volume raycasting. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 Courses*, pages 1–166, New York, NY, USA, 2008. ACM.

[137] John W. (Lord Rayleigh) Strutt. On the light from the sky, its polarization and colour. *Philosophical Magazine*, 41:107–120,274–279, 1871.

[138] Louis G. Henvey and Jesse L. Christof Greenstein. Diffuse reflection in the galaxy. *Astrophysics Journal*, 93:70–83, 1941.

[139] Paul Debevec. A median cut algorithm for light probe sampling. In *ACM SIGGRAPH 2005 Posters*, SIGGRAPH '05, New York, NY, USA, 2005. ACM.

[140] Petrik Clarberg, Wojciech Jarosz, Tomas Akenine-Möller, and Henrik W. Jensen. Wavelet importance sampling: Efficiently evaluating products of complex functions. *ACM Transactions on Graphics*, 24(3):1166–1175, July 2005.

[141] Thomas Annen, Zhao Dong, Tom Mertens, Philippe Bekaert, Hans-Peter Seidel, and Jan Kautz. Real-time, all-frequency shadows in dynamic scenes. *ACM Transactions on Graphics*, 27(3):34:1–34:8, August 2008.

[142] Lance Williams. Pyramidal Parametrics. In *Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '83, pages 1–11, New York, NY, USA, 1983. ACM.

[143] Zhou Wang and Qiang Li. Information Content Weighting for Perceptual Image Quality Assessment. *IEEE Transactions on Image Processing*, 20(5):1185–1198, May 2011.

[144] Wang Zhou. IW-SSIM: Information content weighted structural similarity index for image quality assessment. `https://ece.uwaterloo.ca/~z70wang/research/iwssim/`. Date accessed : 2015-03-11.

[145] Jeppe R. Frisvad, Niels J. Christensen, and Henrik W. Jensen. Computing the scattering properties of participating media using lorenz-mie theory. *ACM Transactions on Graphics*, 26(3), July 2007.

[146] Diego Gutierrez, Francisco J. Seron, Adolfo Munoz, and Oscar Anson. Visualizing underwater ocean optics. *Computer Graphics Forum*, 27(2):547–556, 2008.

[147] Stephane Guy and Cyril Soler. Graphics gems revisited: fast and physically-based rendering of gemstones. *ACM Transactions on Graphics*, 23(3):231–238, August 2004.

[148] W. D. Wright. A re-determination of the trichromatic coefficients of the spectral colours. *Transactions of the Optical Society*, 30(4):141, March 1929.

[149] Downloads | SCHOTT north america. `http://www.us.schott.com/advanced_optics/english/download/index.html`. Date accessed : 2013-04-06.

[150] Andrea Weidlich and Alexander Wilkie. Anomalous dispersion in predictive rendering. *Computer Graphics Forum*, 28(4):1065–1072, 2009.

[151] NVIDIA OptiX ray tracing engine. `http://developer.nvidia.com/optix`. Date accessed : 2013-07-17.

[152] John von Neumann. Various techniques used in connection with random digits. In *Monte Carlo Method*, volume 12 of *National Bureau of Standards Applied Mathematics Series*, pages 36–38. U.S. Government Printing Office, Washington, D.C., 1951.

[153] Jeffrey Ho, Jongwoo Lim, Ming-Hsuan Yang, and David Kriegman. Integrating surface normal vectors using fast marching method. In *Proceedings of the 9th European Conference on Computer Vision - Volume Part III*, ECCV'06, pages 239–250, Berlin, Heidelberg, 2006. Springer-Verlag.

My Caustic Research Publications

[1] Budianto Tandianus, Henry Johan, Hock S. Seah, and Feng Lin. Spectral caustic rendering of a homogeneous caustic object based on wavelength clustering and eye sensitivity. *The Visual Computer*, pages 1–14, October 2014. Chapter 4.

[2] Budianto Tandianus, Henry Johan, and Hock S. Seah. Caustic object construction based on multiple caustic patterns. *Journal of WSCG*, 20(1):37–46, 2012. Chapter 5.

[3] Budianto Tandianus, Henry Johan, and Hock S. Seah. Real-time rendering of approximate caustics under environment illumination. *Entertainment Computing*, 3(4):129 – 141, 2012. Invited publication, one of the top four papers in the 9th International Conference on Entertainment Computing (ICEC). Chapter 3.

[4] Budianto Tandianus, Henry Johan, and Hock S. Seah. Real-time caustics in dynamic scenes with multiple directional lights. In Hyun Yang, Rainer Malaka, Junichi Hoshino, and Jung Han, editors, *Entertainment Computing - ICEC 2010*, volume 6243 of *Lecture Notes in Computer Science*, pages 308–316. Springer Berlin / Heidelberg, 2010. Chapter 3.

My Other Research Publications

[1] Budianto Tandianus, Chau T. Huynh, Jie Qiu, Quan Chen, Feng Lin, and Hock Soon Seah. GPU renderfarm with integrated asset management & production system. Poster in GPU Technology Workshop South East Asia 2014.

[2] Budianto Tandianus, Chau Trung Huynh, Jie Qiu, Quan Chen, Feng Lin, and Hock S. Seah. GPU renderfarm with integrated asset management & production system. Technical Report (was presented in GTC 2014).

[3] Aman Gupta, Satyam Mandavalli, Vincent J. Mooney, Keck-Voon Ling, Aridam Basu, Henry Johan, and Budianto Tandianus. Low power probabilistic floating point multiplier design. In *VLSI (ISVLSI), 2011 IEEE Computer Society Annual Symposium on*, pages 182 –187, July 2011.

Appendix A

Spectral Rendering

A.1 Differentiability of $d\theta_2(\lambda, \theta_1)/d\lambda$

Let $s = \sin(\theta_1)$, $a = \eta_2^{'}(\lambda)$, $b = \eta_2(\lambda)$. If we differentiate Equation 4.3 with respect to the λ direction ($\sin(\theta_1)$ becomes a constant s) :

$$\begin{aligned}\frac{\partial}{\partial\lambda}\Bigg(-s\frac{\eta_2^{'}(\lambda)}{\sqrt{\eta_2(\lambda)^2(\eta_2(\lambda)^2-s^2)}}\Bigg) = \\ -s\Bigg(\frac{\eta_2^{''}(\lambda)}{\eta_2(\lambda)\sqrt{\eta_2(\lambda)^2-s^2}} - \frac{\eta_2^{'}(\lambda)(2\eta_2(\lambda)^2\eta_2^{'}(\lambda)-s^2\eta_2^{'}(\lambda))}{\eta(\lambda)^2(\eta_2(\lambda)^2-s^2)^{3/2}}\Bigg).\end{aligned} \tag{A.1}$$

The equation will become not differentiable when the $\eta_2(\lambda)^2 - s^2$ in both denominators is 0 or less. However, it will not happen as $0^o \leq \theta_1 \leq 89^o$ (which in turn the range of s^2 is $[0 \ldots 0.99]$) and $1.0 \leq \eta_2(\lambda)$. Since $\eta_2(\lambda)$ monotonically decreases, its first and second derivatives $\eta_2^{'}(\lambda)$ and $\eta_2^{''}(\lambda)$ respectively) will be monotonically decreasing, but they will not make the equation to be not differentiable.

If we differentiate Equation 4.3 with respect to the θ_1 direction ($\eta_2^{'}(\lambda)$ and $\eta_2(\lambda)$ become constants a and b respectively) :

$$\frac{\partial}{\partial\theta_1}\Bigg(-\frac{a}{b}\frac{\sin(\theta_1)}{\sqrt{b^2-\sin(\theta_1)^2}}\Bigg) = -\frac{ab\cos(\theta_1)}{(b^2-\sin(\theta_1)^2)^{3/2}}. \tag{A.2}$$

It will not be differentiable if $b^2 - \sin(\theta_1)^2$ is zero or less than zero. Similar to the derivative with respect to λ direction, it will also not happen with the same reason.

If we perform double derivatives (i.e. with respect to both λ and θ_1 directions), we will obtain :

$$\begin{aligned}\frac{\partial^2}{\partial\lambda\partial\theta_1}\left(\frac{-\sin(\theta_1)\eta_2^{'}(\lambda)}{\sqrt{\eta_2(\lambda)^2(\eta_2(\lambda)^2-\sin(\theta_1)^2)}}\right) = &\frac{-\cos(\theta_1)}{(\eta_2(\lambda)^2-\sin(\theta_1)^2)^{(5/2)}} - \\ &\sin(\theta_1)^2 * \eta_2(\lambda) * \eta_2^{''}(\lambda) + \\ &\eta_2(\lambda)^3 * \eta_2^{''}(\lambda) - \sin(\theta_1)^2 * \eta_2^{'}(\lambda)^2 - \\ &2 * \eta_2(\lambda)^2 * \eta_2^{'}(\lambda)^2,\end{aligned} \tag{A.3}$$

which is also differentiable for the ranges $0^o \leq \theta_1 \leq 89^o$ and $1.0 \leq \eta_2(\lambda)$. Hence, Equation 4.3 is differentiable at all points, including the four corners of each patch (as they are within the ranges of $0^o \leq \theta_1 \leq 89^o$ and $1.0 \leq \eta_2(\lambda)$) .

There is one special case where the function is not differentiable, i.e. when the arbitrary IOR is not continuous (i.e. it forms a staircase graph), such as :

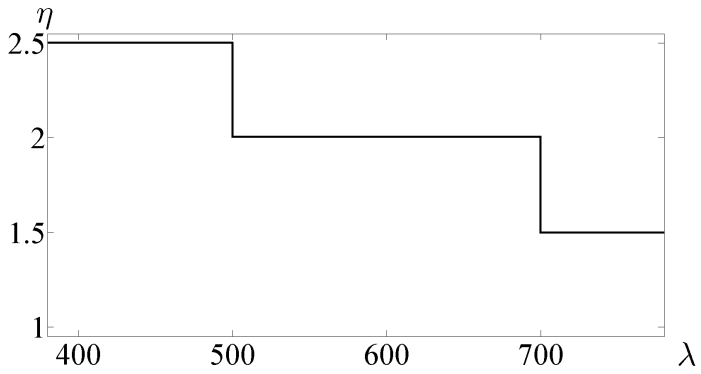


Figure A.1: Index of Refraction with a staircase shape.

In our computation, we use finite difference in computing the differentials (thus for the corners we use forward/backward finite differential accordingly). Hence, even though we encounter the staircase IOR, we will not obtain undefined $\eta_2^{'}(\lambda)$. Instead, we will obtain a higher jump around the discontinuity point that may result in the wavelengths around this discontinuity point being put into its own cluster (which is desirable, even if we obtain

an undefined first derivative, we assume the change is infinitely large and the wavelength should be put in its own wavelength cluster). Moreover, we also would like to mention that we only store one IOR value for each wavelength in our implementation (consequently, we only allow the arbitrary IOR input value to be continuous).

A.2 Constants used in Spectral Rendering

$$\mathbf{M}^{-1} = \begin{vmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 & -2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & -2 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -3 & 3 & 0 & 0 & -2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & -2 & 0 & 0 & 1 & 1 & 0 & 0 \\ -3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -3 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & -1 & 0 \\ 9 & -9 & -9 & 9 & 6 & 3 & -6 & -3 & 6 & -6 & 3 & -3 & 4 & 2 & 2 & 1 \\ -6 & 6 & 6 & -6 & -3 & -3 & 3 & 3 & -4 & 4 & -2 & 2 & -2 & -2 & -1 & -1 \\ 2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ -6 & 6 & 6 & -6 & -4 & -2 & 4 & 2 & -3 & 3 & -3 & 3 & -2 & -1 & -2 & -1 \\ 4 & -4 & -4 & 4 & 2 & 2 & -2 & -2 & 2 & -2 & 2 & -2 & 1 & 1 & 1 & 1 \end{vmatrix}$$

$$\mathbf{b} = \begin{vmatrix} b_{0,0} \\ b_{1,0} \\ b_{2,0} \\ b_{3,0} \\ b_{0,1} \\ b_{1,1} \\ b_{2,1} \\ b_{3,1} \\ b_{0,2} \\ b_{1,2} \\ b_{2,2} \\ b_{3,2} \\ b_{0,3} \\ b_{1,3} \\ b_{2,3} \\ b_{3,3} \end{vmatrix} = \begin{vmatrix} 1.0 \\ 1.0/2.0 \\ 1.0/3.0 \\ 1.0/4.0 \\ 1.0/2.0 \\ 1.0/4.0 \\ 1.0/6.0 \\ 1.0/8.0 \\ 1.0/3.0 \\ 1.0/6.0 \\ 1.0/9.0 \\ 1.0/12.0 \\ 1.0/4.0 \\ 1.0/8.0 \\ 1.0/12.0 \\ 1.0/16.0 \end{vmatrix}$$

Appendix B

Caustic Object Reconstruction Survey

In this Appendix, we present the surveys we conducted for assessing our work in caustic object reconstruction. Each survey consists of 15 respondents. They are asked to choose which reconstructed caustic patterns better resemble the input caustic patterns. Each question consists of three columns: the first column is the input caustic pattern, the second and third columns are two reconstruction results to be assessed. In order to reduce bias, we do not provide the information on which approach was used in the 2nd and 3rd columns and the results of the two approaches are placed randomly in the 2nd and 3rd columns. As additional references to readers, we also include irradiance difference maps in each of the screenshot, that were not presented to the respondents in the surveys. Generally, the respondents voted for the results with smaller differences in the irradiance difference maps.

B.1 First survey

The first survey aims to find the superior option out of two possible options explained in Section 5.2, i.e. changing the size and position of caustic regions directly, or changing the size of caustic regions indirectly (by optimizing the light convergence).

Table B.1: Survey of first optimization options (1st caustic pattern set).

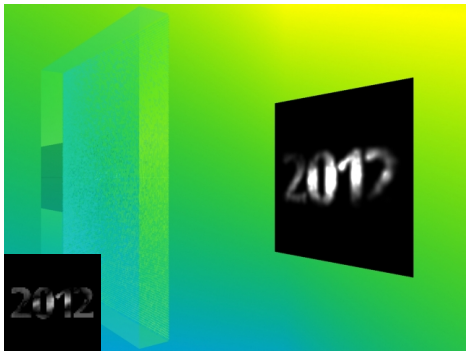
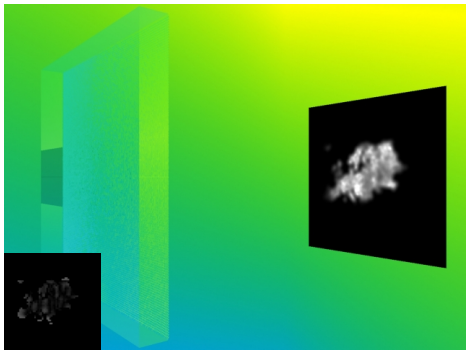
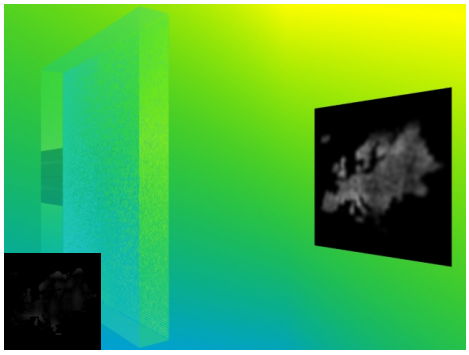
Input	Size and Position Adjustment	Light Convergence Adjustment
		
	6.67%	93.33%
		
	100.00%	0.00%
		
	13.33%	86.67%
Average	40.00%	60.00%

Table B.2: Survey of first optimization options (2nd caustic pattern set).

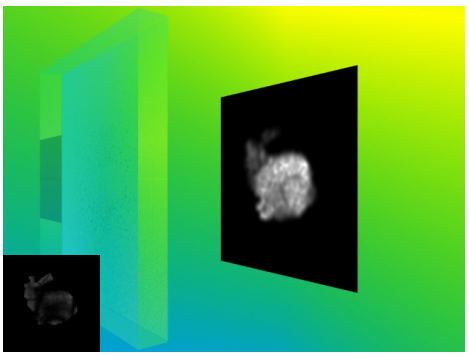
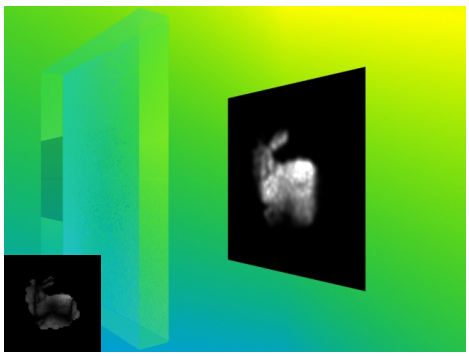
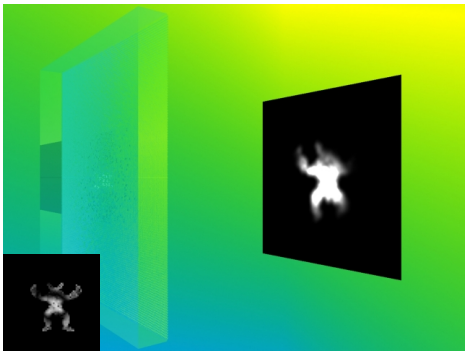
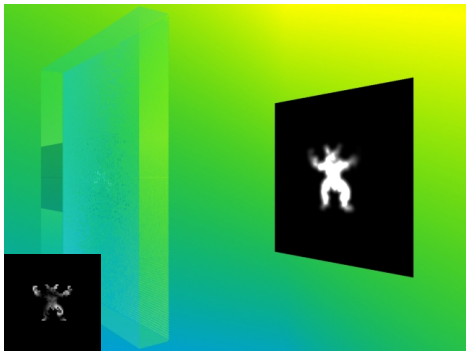
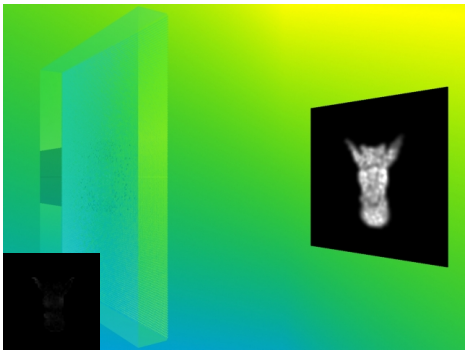
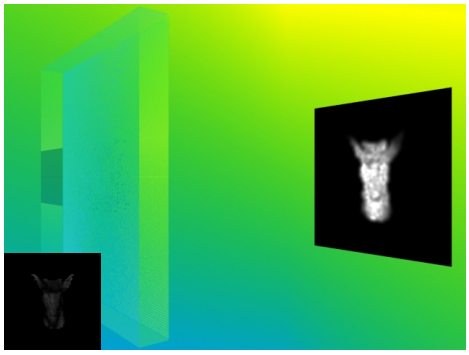
Input	Size and Position Adjustment	Light Convergence Adjustment
	 53.33%	 46.67%
	 0.00%	 100.00%
	 86.67%	 13.33%
Average	46.67%	53.33%

Table B.3: Survey of first optimization options (3rd caustic pattern set).

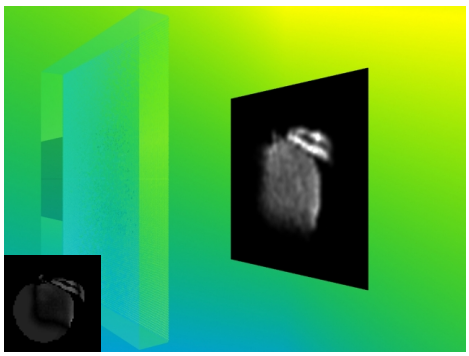
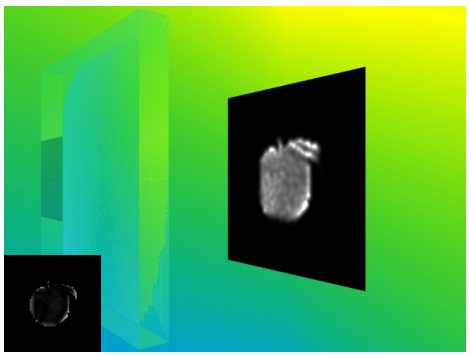
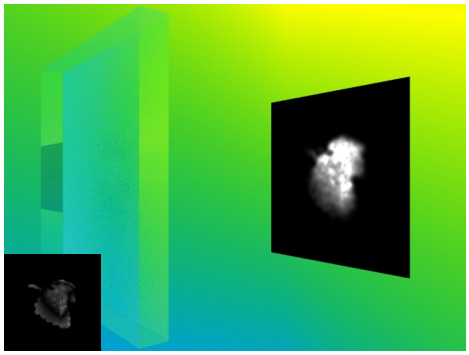
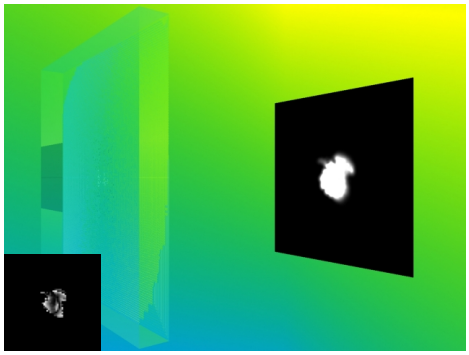
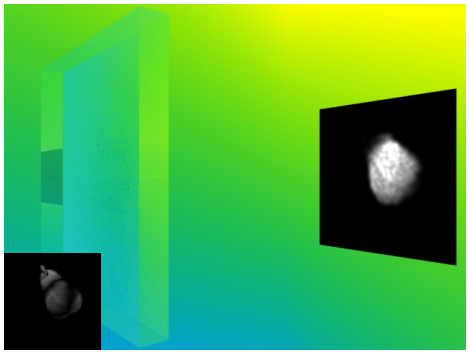
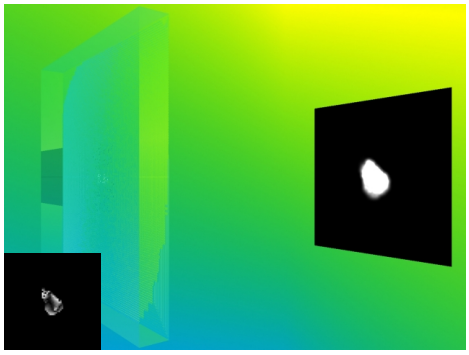
Input	Size and Position Adjustment	Light Convergence Adjustment
	 0.00%	 100.00%
	 100.00%	 0.00%
	 20.00%	 80.00%

Table B.3: Survey of first optimization options (3rd caustic pattern set)

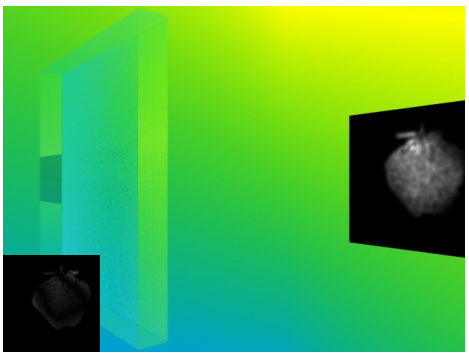
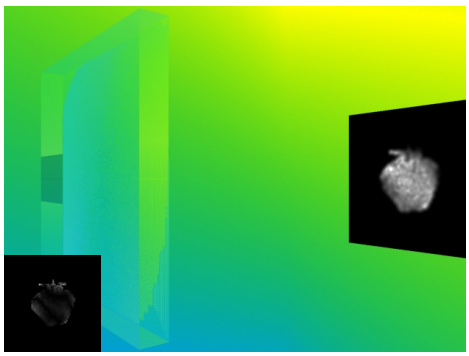
Input	Size and Position Adjustment	Light Convergence Adjustment
		
	66.67%	33.33%
Average	46.67%	53.33%

Table B.4: Survey of first optimization options (4th caustic pattern set).

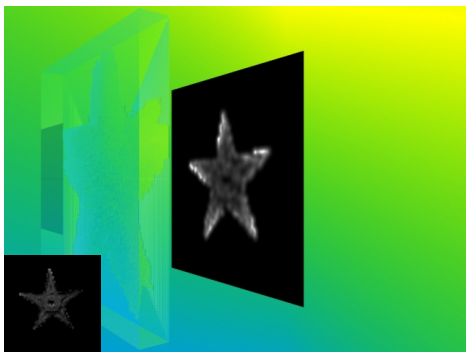
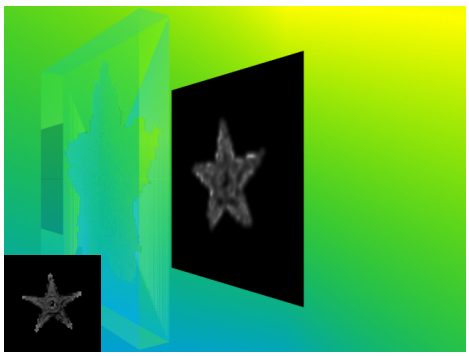
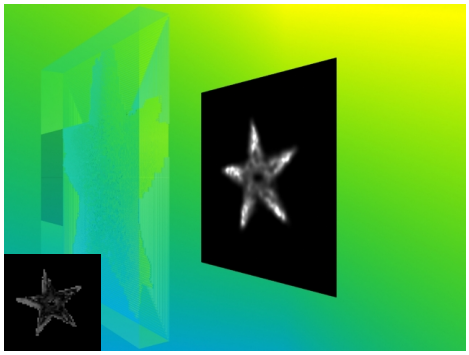
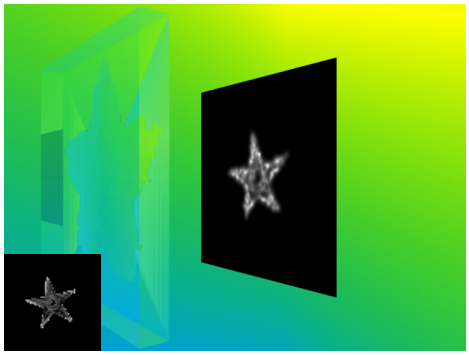
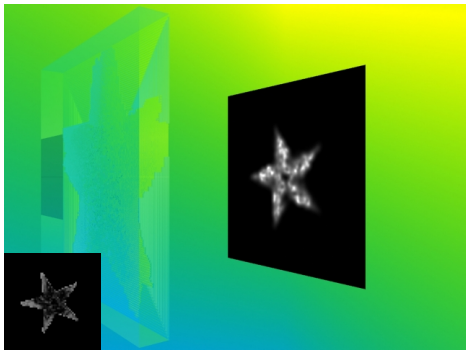
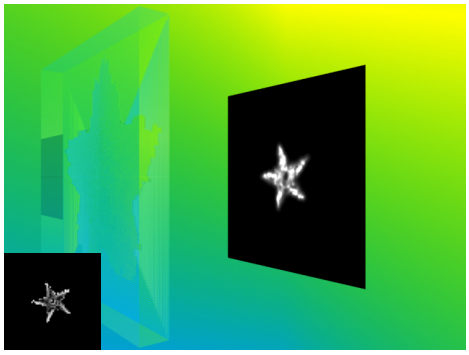
Input	Size and Position Adjustment	Light Convergence Adjustment
	 60.00%	 40.00%
	 53.33%	 46.67%
	 80.00%	 20.00%

Table B.4: Survey of first optimization options (4th caustic pattern set)

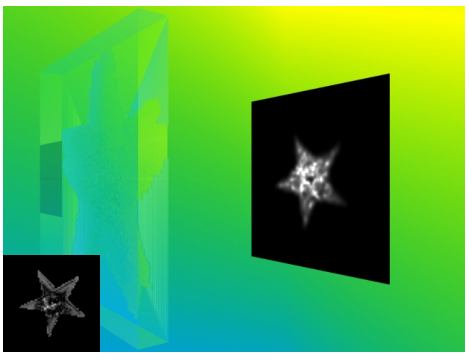
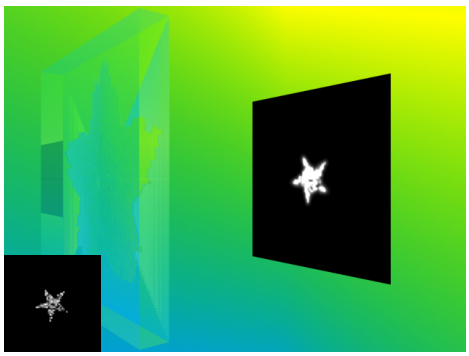
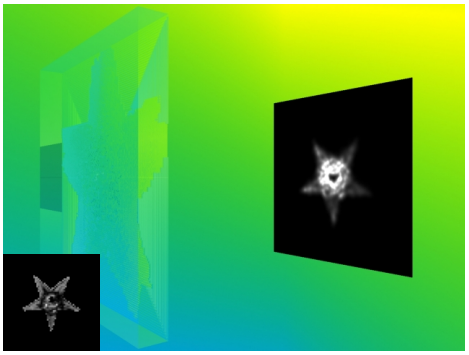
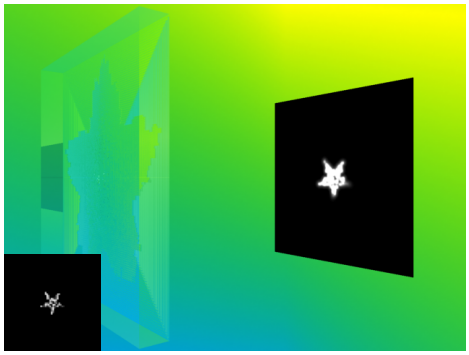
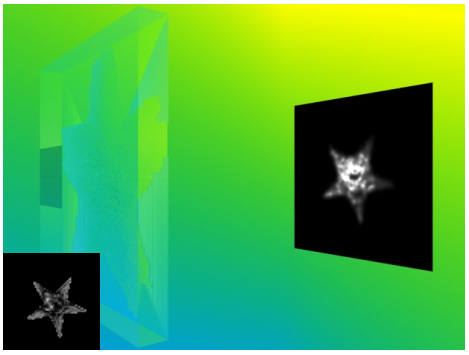
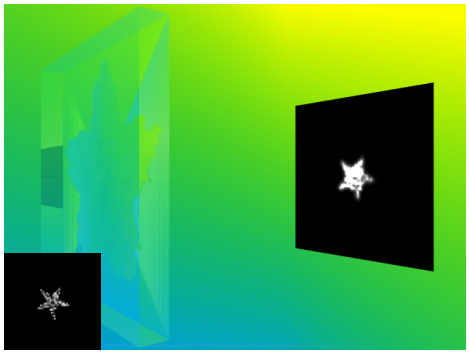
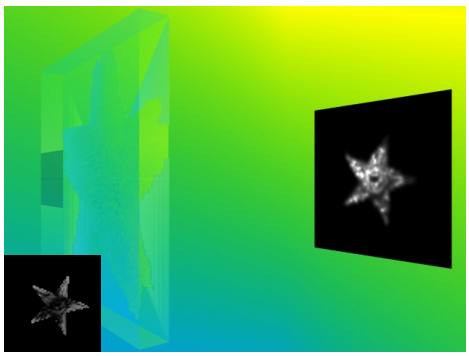
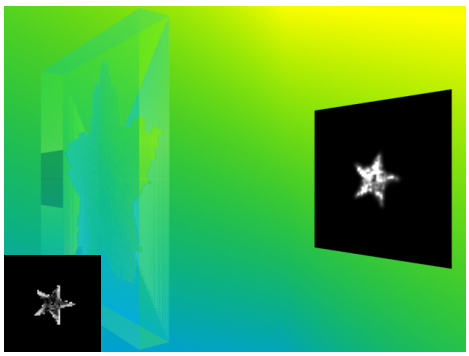
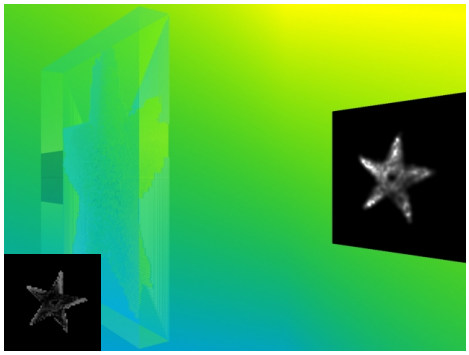
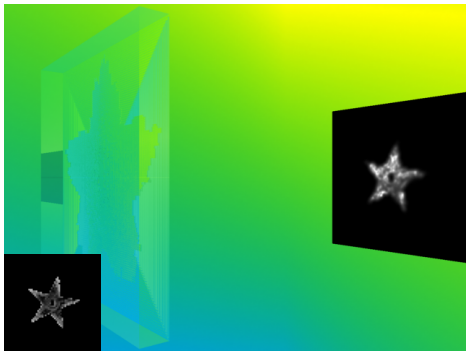
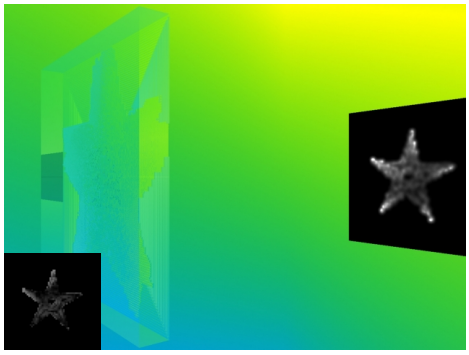
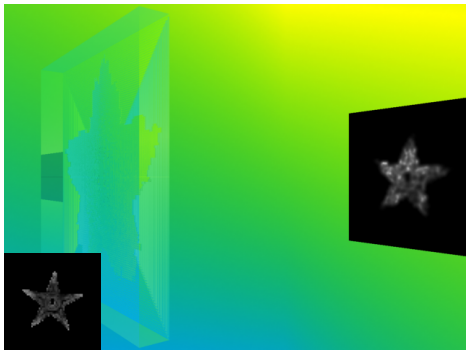
Input	Size and Position Adjustment	Light Convergence Adjustment
	 93.33%	 6.67%
	 86.67%	 13.33%
	 86.67%	 13.33%

Table B.4: Survey of first optimization options (4th caustic pattern set)

Input	Size and Position Adjustment	Light Convergence Adjustment
	 93.33%	 6.67%
	 93.33%	 6.67%
	 100.00%	 0.00%
Average	82.96%	17.04%

B.2 Second survey

The survey aims to prove that the intensity correction step (Section 5.3) can improve caustic pattern reconstruction results. In this survey we compare between the results of with and without intensity correction. In both results, we apply the optimizations explained in Sections 5.2.1 and 5.2.2.

Table B.5: Survey of intensity correction improvement (1st caustic pattern set).

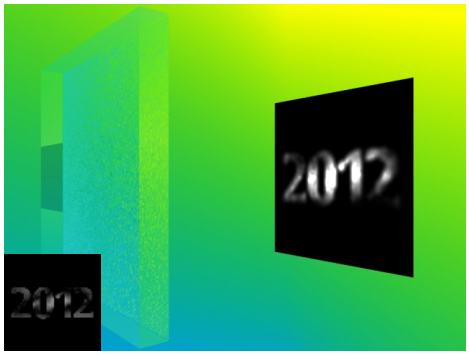
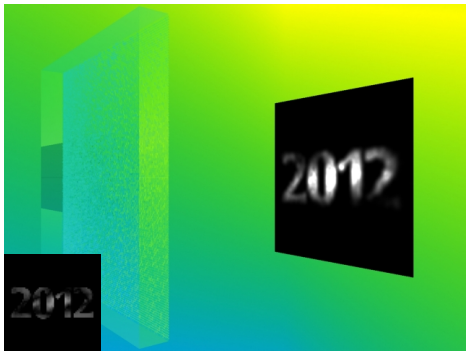
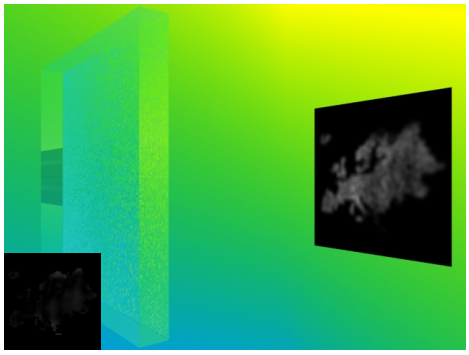
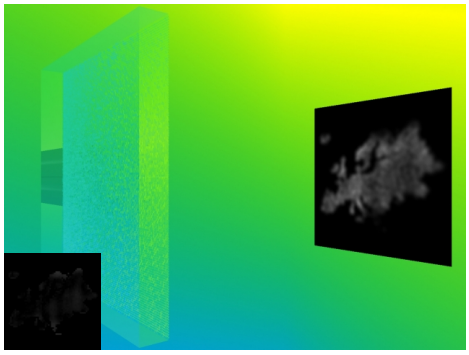
Input	Without Intensity Correction	With Intensity Correction
		
	66.67%	33.33%
		
	26.67%	73.33%
		
	40.00%	60.00%
Average	44.45%	55.55%

Table B.6: Survey of intensity correction improvement (2nd caustic pattern set).

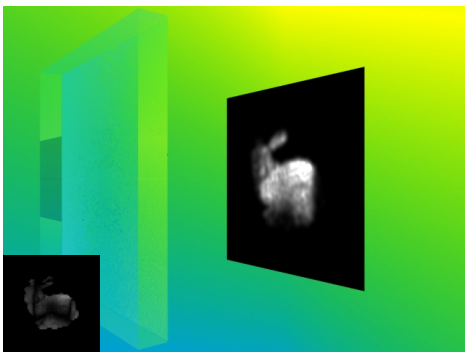
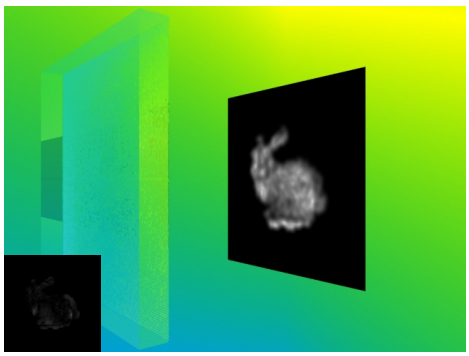
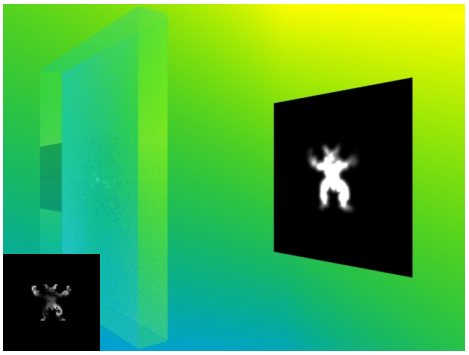
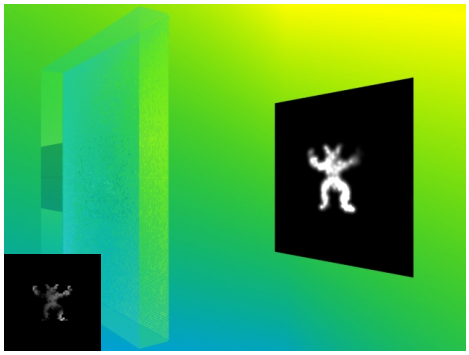
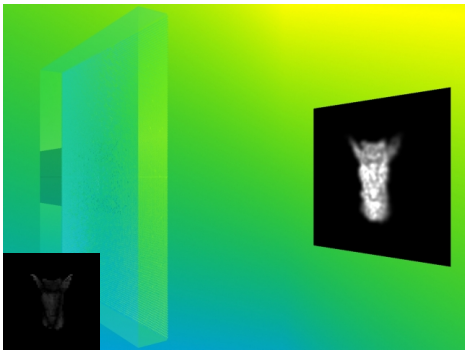
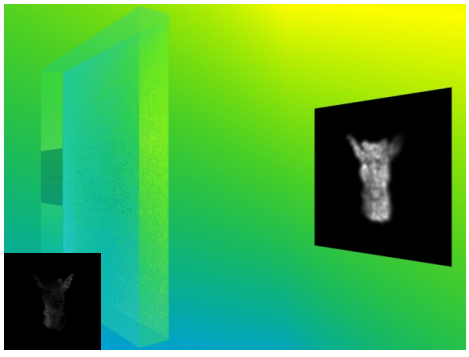
Input	Without Intensity Correction	With Intensity Correction
	 0.00%	 100.00%
	 13.33%	 86.67%
	 53.33%	 46.67%
Average	22.22%	77.78%

Table B.7: Survey of intensity correction improvement (3rd caustic pattern set).

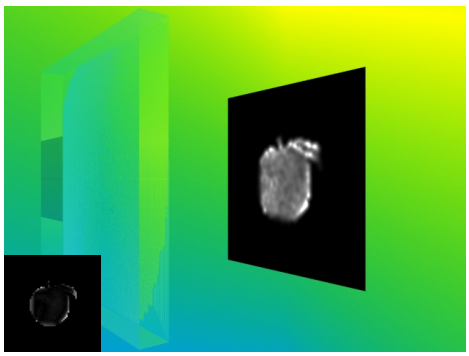
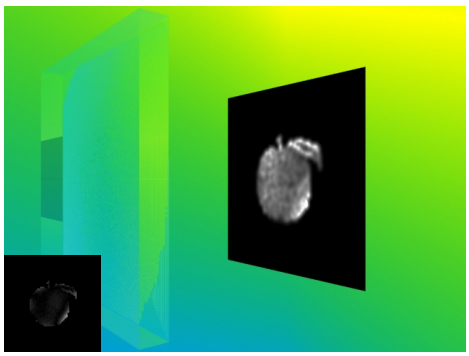
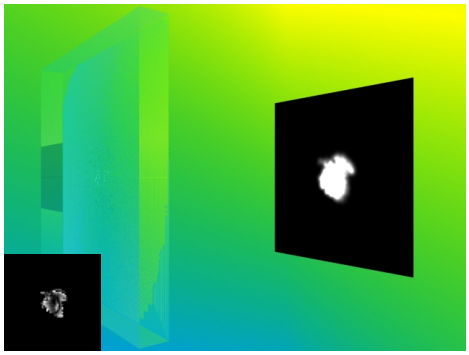
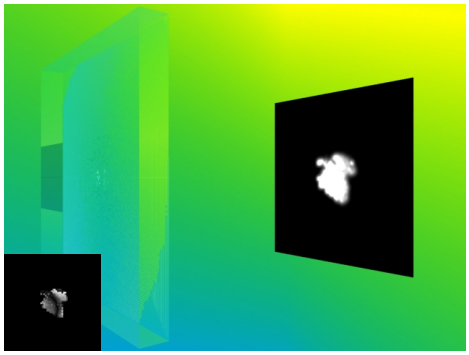
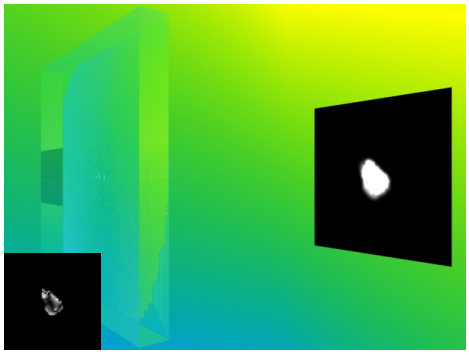
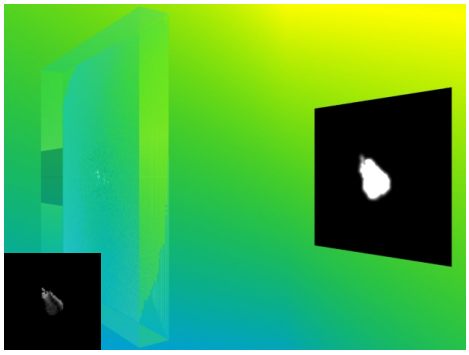
Input	Without Intensity Correction	With Intensity Correction
	 0.00%	 100.00%
	 6.67%	 93.33%
	 0.00%	 100.00%

Table B.7: Survey of intensity correction improvement (3rd caustic pattern set)

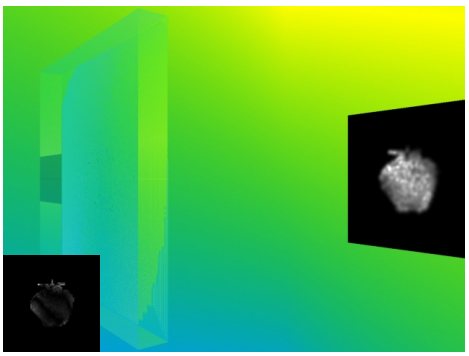
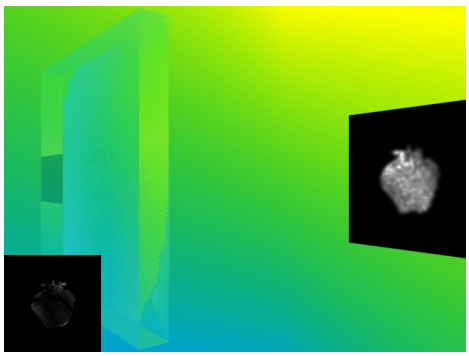
Input	Without Intensity Correction	With Intensity Correction
		
	26.67%	73.33%
Average	8.34%	91.67%

Table B.8: Survey of intensity correction improvement (4th caustic pattern set)

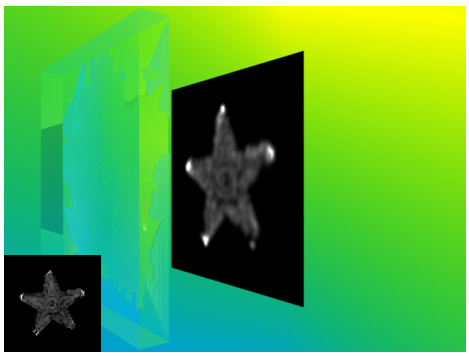
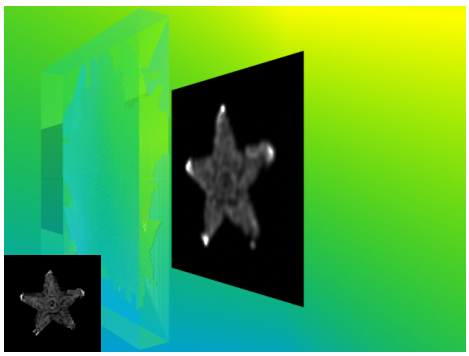
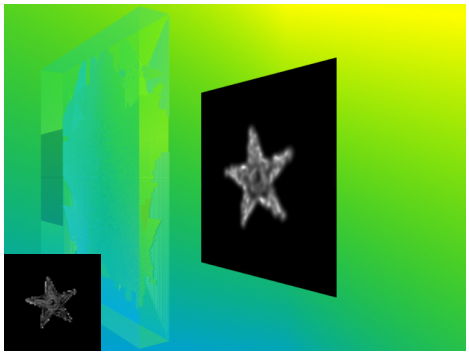
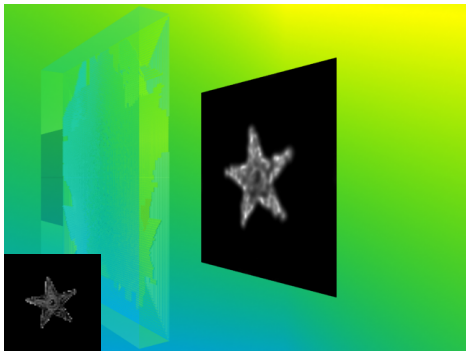
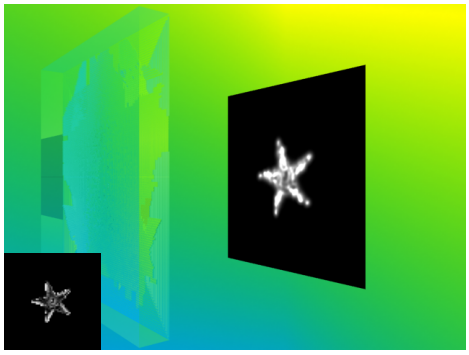
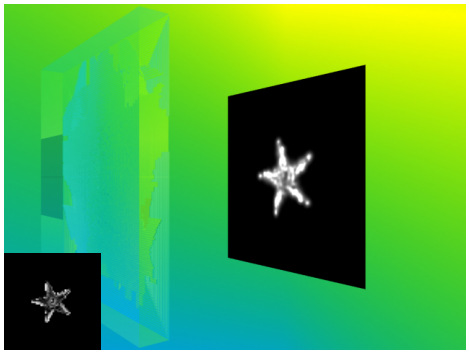
Input	Without Intensity Correction	With Intensity Correction
	 26.67%	 73.33%
	 53.33%	 46.67%
	 33.33%	 66.67%

Table B.8: Survey of intensity correction improvement (4th caustic pattern set)

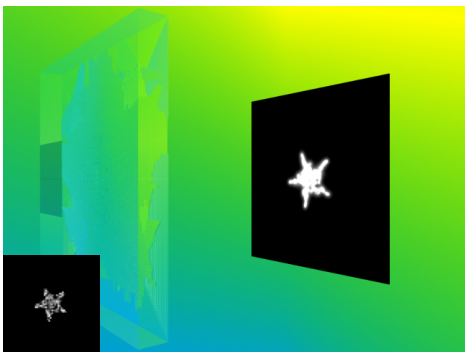
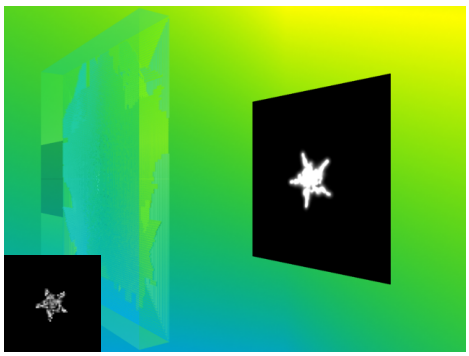
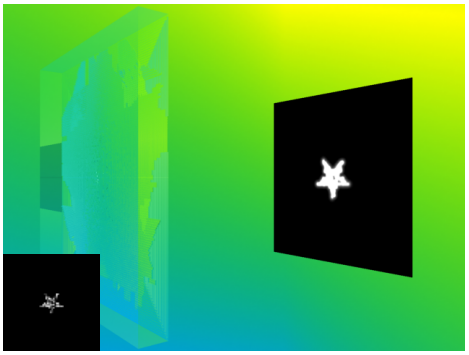
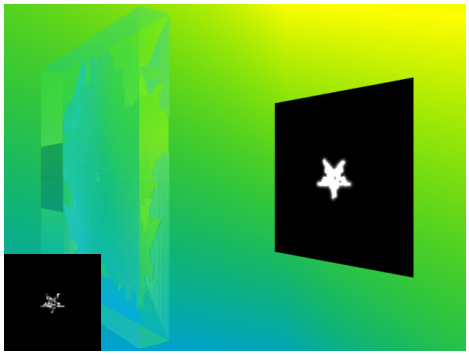
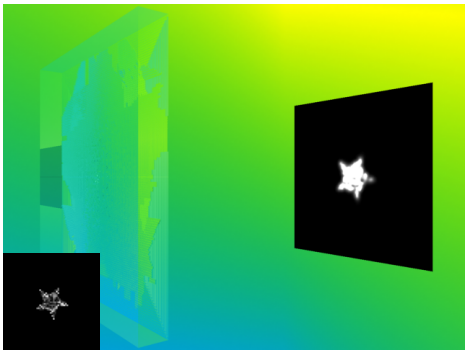
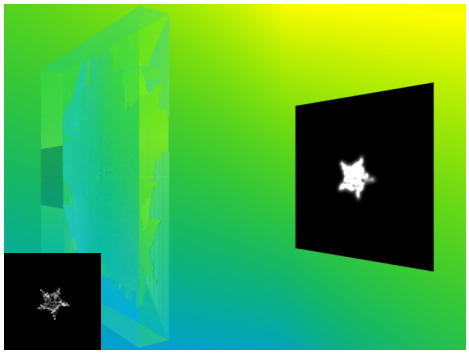
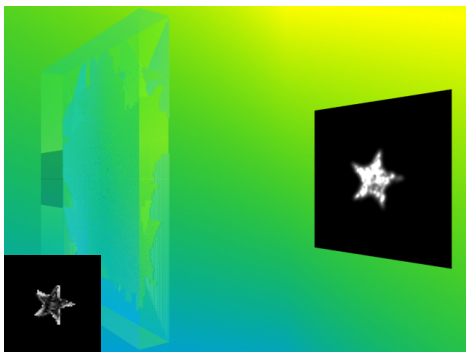
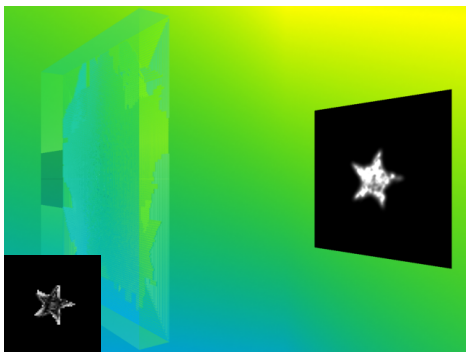
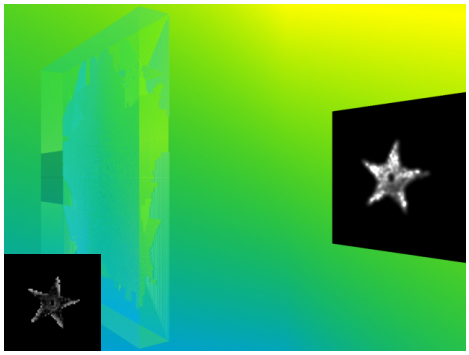
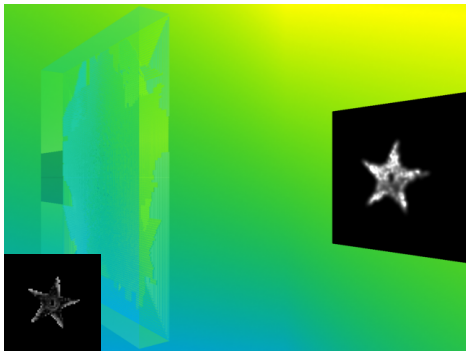
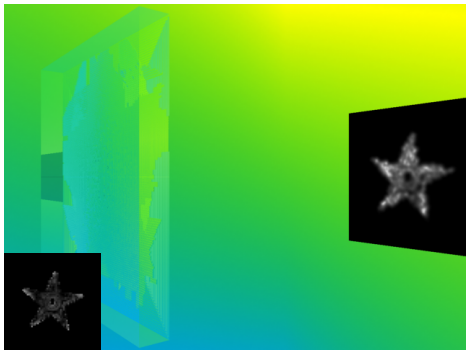
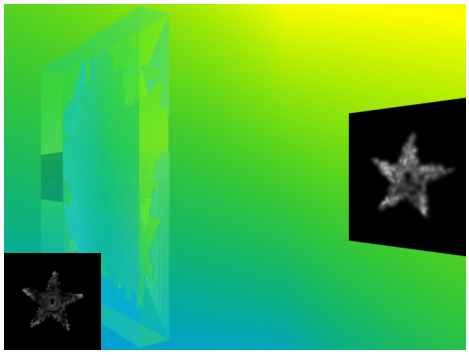
Input	Without Intensity Correction	With Intensity Correction
	 46.67%	 53.33%
	 80.00%	 20.00%
	 33.33%	 66.67%

Table B.8: Survey of intensity correction improvement (4th caustic pattern set)

Input	Without Intensity Correction	With Intensity Correction
	 33.33%	 66.67%
	 66.67%	 33.33%
	 66.67%	 33.33%
Average	48.89%	51.11%