FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Reflections, Refractions and Caustics in a Mixed-Reality Environment

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Christoph Johann Winklhofer
Matrikelnummer 0426461

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Dipl.-Ing. Mag.rer.soc.oec. Martin Knecht, Bakk.techn

Wien, 31.01.2013

(Unterschrift Verfasserin)

(Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Reflections, Refractions and Caustics in a Mixed-Reality Environment

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Visual Computing

by

Christoph Johann Winklhofer
Registration Number 0426461

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Dipl.-Ing. Mag.rer.soc.oec. Martin Knecht, Bakk.techn

Vienna, 31.01.2013

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Christoph Johann Winklhofer
Fuxpointweg 8, 5204 Straßwalchen

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasserin)

Acknowledgements

With these few lines I would like to thank everyone who helped me writing this thesis. Without your support, this work would still be virtual. Therefore, I want to thank the following people:

My first supervisor *Michael Wimmer* for the interesting research topic, the useful comments in structuring and writing, and the uncomplicated and quick replies to my questions.

My second supervisor *Martin Knecht* for pushing me into the right directions during the implementation, your motivating screen-shot YEAH-comments, proofreading, and showing me what it means to work scientifically accurate.

Christoph Traxler who initiated the *Reciprocal Shading for Mixed Reality* project and makes it at all possible to write on this topic.

The *Imagination Computer Services* company for kindly providing a Studierstube-Tracker license.

Andreas Seidl for modeling the virtual drinking glass, cheers.

My parents *Liselotte* and *Gottfried Winklhofer* who are always there when I need them and who supported me in all my decisions through life.

And finally, *Anne Biber* for the invaluable help, proofreading, for your patience when I stared into my monitor, and just being near ***.

Abstract

In a mixed-reality environment virtual objects are merged into a real scene. Such an augmentation with virtual objects offers great possibilities to present content in new and innovative ways. The visual appearance of these virtual objects depends on a plausible lighting simulation. Otherwise, virtual objects look artificial and out of place, which destroys the overall impression of the perceived scene.

Reflective and refractive objects are an inherent part of our physical environment. Accordingly, virtual objects of this type also enhance the overall impression and scope of a mixed-reality application. Many mixed-reality systems still neglect them: Such objects require a complex light simulation that is hard to embed in a mixed-reality system, which demands real-time frame rates to handle the user interaction.

This thesis describes the integration of reflective and refractive objects in a mixed-reality environment. The aim is to create a realistic light distribution that simulates reflections and refractions between real and virtual objects. Another important aspect for a believable perception are caustics, light focusing due to the scattering from reflective or refractive objects. Until recently, this effect was simply excluded in the lighting simulation of mixed-reality systems.

The proposed rendering method extends differential instant radiosity with three other image space rendering techniques capable to handle reflections, refractions and caustics in real time. By combining these techniques, our method successfully simulates the various lighting effects from reflective and refractive objects and is able to handle user interactions at interactive to real-time frame rates. This offers a practicable possibility to greatly improve the visual quality of a mixed-reality environment.

Kurzfassung

In einer Mixed-Reality Umgebung oder gemischten Realität werden virtuelle Objekte in eine reale Szene integriert. Eine solches Hinzufügen von virtuellen Objekten bietet eine innovative Möglichkeit, um Inhalte interessant und ansprechend aufzubereiten. Für das visuelle Erscheinungsbild der virtuellen Objekte ist eine authentische Beleuchtungssimulation notwendig. Andernfalls wirken die virtuellen Objekte künstlich und deplatziert, was den Gesamteindruck der wahrgenommenen Szene stört.

Reflektierende und refraktierende Objekte sind in der Realität allgegenwärtig. Virtuelle Objekte dieser Art können entsprechend das Erscheinungsbild und die Nutzungsmöglichkeiten einer Mixed-Reality Anwendung verbessern. In vielen Mixed-Reality Systemen werden sie dennoch vernachlässigt: Für ihre Darstellung ist eine komplexe Lichtsimulation erforderlich. Eine solche lässt sich jedoch in ein Mixed-Reality System, das wegen der Benutzerinteraktion auf eine schnelle Bildausgabe angewiesen ist, nur schwer integrieren.

Die Diplomarbeit befasst sich mit der Integration von reflektierenden und refraktierenden Objekten in eine Mixed-Reality Umgebung. Das Ziel ist eine realistische Lichtverteilung für die Simulation von Reflexionen und Refraktionen zwischen realen und virtuellen Objekten. Für eine überzeugende Darstellung sind außerdem Kaustiken wichtig. Dabei handelt es sich um Lichtfokussierungen durch Streuung an reflektierenden und refraktierenden Objekten. Bis vor Kurzem wurde dieser Effekt in der Beleuchtungssimulation anderer Mixed-Reality Systeme nicht berücksichtigt.

Die von uns vorgeschlagene Methode erweitert den Differential Instant Radiosity Algorithmus mit drei anderen im Bildraum arbeitenden Techniken, die Reflexionen, Refraktionen und Kaustiken in Echtzeit berechnen können. Diese Algorithmen werden bei unserer Methode erfolgreich kombiniert. Dadurch können wir verschiedenste Lichteffekte von reflektierenden und refraktierenden Objekten simulieren und außerdem die Benutzerinteraktion in Echtzeit gewährleisten. Die Methode bietet eine praktische Möglichkeit um die visuelle Qualität einer Mixed-Reality Umgebung zu verbessern.

Contents

CHAPTER 1

Introduction

In a *mixed-reality* environment, virtual objects are merged into a real scene. A mixed-reality environment lies between the real environment, which contains only real objects, and the virtual environment, which just consists of virtual objects, as illustrated by Figure 1.1. This *reality-virtuality continuum* [29] provides a classification for a mixed reality that includes *augmented reality*, as in our case, and *augmented virtuality*, in which real objects are merged with a virtual scene.

Such an augmentation with virtual objects offers great possibilities to present content in new and innovative ways. There is a variety of interesting application areas that could, or already do benefit from a mixed-reality approach. Figure 1.2 shows a small selection of some application scenarios. A classical application example is advertising, in which a mixed-reality approach allows actively involving the consumer into the advertising campaign, see Figure 1.2a. Another application field is product visualization, like in Figure 1.2b or Figure 1.2c. Besides entertainment, mixed reality can also be useful in medical areas, for the planning of operations or to aid a surgeon during an operation, see Figure 1.2d. Other application fields are cultural heritage or the education sector, see Figure 1.2e.

From a technical point of view, a mixed-reality system also needs to merge several different subsystems that depend on each other. These subsystems are responsible for capturing the perceived environment, handling the user interaction, determining the position and orientation of the objects and simulating the light distribution. Finally, the mixed-reality system outputs an image of the augmented environment.

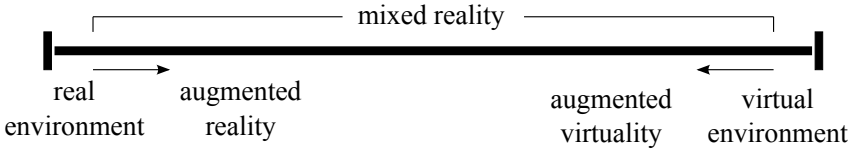


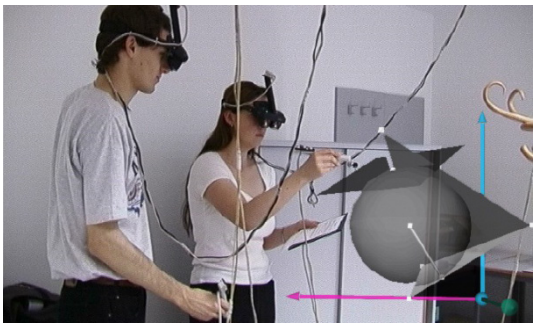
Figure 1.1: Reality-virtuality continuum [29]

(a) Advertising [2]

(b) Product visualization [45]

(c) Product visualization [26]

(d) Medical areas [15]



(e) Education [20]

Figure 1.2: Fields of application for mixed reality. Figure (a) shows an advertisement for the National Geographic Channel that draws virtual animals next to pedestrians. On a nearby screen, it seems that these animals are part of the real scene and people even tried to touch the virtual animals. Image copyright by Appshaker Ltd: www.appshaker.co.uk [2]. Figure (b) shows a mixed-reality application on a mobile device that helps the user on the decision for new furniture. Hence, it is possible to try different furnitures and get an impression of how they would look like in their real environment. Image copyright by ViewAR - www.viewar.com [45]. In Figure (c), the customer holds a LEGO box in front of a camera and gets an impression of the assembled model in three dimensions. Image copyright by metaio - www.metaio.com [26]. Figure 1.2d shows an overlay of a X-ray photograph on the camera image of leg. Image from an article of the Technische Unversität München [15]. Figure 1.2e shows a mixed-reality application for geometry teaching. Image courtesy of Kaufmann [20].

The computation time is a limiting factor for all these tasks. Usually, a mixed-reality system has some sort of user interaction. A lagging input system or a low output frame rate is annoying for the user and would disturb the overall immersion. Therefore, a mixed-reality system should generate the output images at interactive (about 10 frames per second) or preferably at real-time (about 30 frames per second) frame rates.

An authentic lighting simulation is the most important factor for the visual appearance of a mixed-reality scene. This means that the lighting of real objects and virtual objects should interact in a natural way, so that the user perceives virtual objects as part of the real environment.

1.1 Scope of the work

The augmentation of a real scene with virtual content requires a plausible lighting simulation. Otherwise, virtual objects look artificial and out of place, which destroys the overall impression of the perceived scene. This master thesis focuses on a plausible light distribution in a mixed-reality environment.

The emphasis lies on the integration of light effects from reflective and refractive objects in a mixed-reality environment. This means for example that a virtual object should appear on a reflective real object (e.g. a real mirror) or a real object should shine through a refractive virtual object (e.g. a virtual drinking glass).

Reflective and refractive objects also scatter the incoming light rays, which may result in a higher concentration of light in several areas. This focusing of light is called a caustic. This thesis also describes the visualization of plausible caustics in a mixed-reality scenario.

We extend an existing mixed-reality framework that is part of the *Reciprocal Shading for Mixed Reality* (RESHADE) project [42]. The rendering subsystem builds upon *differential instant radiosity* [22]. It neatly combines several different rendering techniques and is able to simulate a realistic light distribution in a mixed-reality environment. Besides some fundamental principles about light, we describe the framework in more detail in Chapter 2.

1.2 Problem statement

Reflective and refractive objects are an inherent part of our physical environment and consequently improve the appearance of a mixed-reality scenario. However, these kinds of objects need special considerations during the lighting computation. A *local illumination* method, which evaluates the lighting solely between the visible surface and the light sources, neglects the light interaction between different surfaces and is therefore insufficient. In a scene with reflective and refractive objects, the lighting simulation also needs to consider the surrounding objects of a visible surface point, what is called a *global illumination* method. However, such an approach needs more effort and indeed, a global illumination method is hard to embed in a mixed-reality system that demands real-time frame rates to handle the user interaction.

The intent of our lighting simulation is to model the light interaction between real and virtual objects in a perceptually plausible manner, so that it appears realistic to a human observer. Summarized, the two main research topics of this thesis are:

1. Research a technique to simulate the lighting of reflective and refractive objects in a mixed-reality environment and integrate it into the RESHADE framework.

2. Research a technique to simulate caustic effects in a mixed-reality environment and integrate it into the RESHADE framework.

An important constraint for a mixed-reality system is the computation time. For this reason, the techniques to simulate reflections, refractions and caustics should at least run at interactive frame rates, allowing user interaction without disturbing delays. Chapter 3 presents several image space algorithms to simulate each of these three effects at real-time frame rates. Moreover, it includes a short overview and comparison of other mixed-reality systems that handle reflective and refractive objects.

Differential rendering is a fundamental technique to combine real and virtual objects in a scene. For now it is sufficient to know that this method computes one light simulation exclusively for all real objects and one together for all real and virtual objects in a scene. Then, the difference between these two simulations is added to the captured image. It is crucial for differential rendering to evaluate the light paths of all included components in the light simulation. These paths determine the radiance contribution into the two differential rendering parts, i.e. should the light contribute only to the real part or to the real and virtual part. Reflective and refractive objects, either real or virtual, add several new light path combinations, including caustics and indirect light bounces. Thus, a considerable part of this thesis is to classify such light paths.

As opposed to the original differential rendering algorithm, we also want to apply this technique to reflected or refracted real objects. For instance, inserting a reflective or refractive virtual object into a scene invalidates the occupied area in the image. Hence, the corresponding reflected or refracted surface has no relation to the visible area in the image because these points originate at a different location. In such a case, the differential rendering method needs to detect the associated value in the camera image. Therefore, it back-projects the reflected or refracted surface position into the camera image and adjusts the color values accordingly. As part of the light path analysis, we need to identify these areas and also care about cases where no valid back-projection data is available.

The following list contains the additional problem statements which arise from the two main research questions:

- Reflection, refraction and caustic techniques should at least run at interactive frame rates.

- Classify the light paths for the differential rendering method between the light source, the interacting objects and the camera.

- Identify the areas for reflective and refractive objects which need a back-projection and handle the cases where no valid data is available.

1.3 Contribution

We successfully integrate, reflective and refractive objects in a mixed-reality environment. Built upon differential instant radiosity from Knecht et al. [22], our proposed method is able to handle various lighting situations in a mixed-reality environment for reflective and refractive objects, either real or virtual. This is the first rasterization-based approach that covers all these effects in its entirety in a mixed-reality environment.

Due to the tight coupling with instant radiosity [21], indirect illumination on diffuse objects also appears in reflections and refractions. Furthermore, these two types of objects realistically react to real or virtual spotlights. Until recently, caustics were just ignored in the lighting simulation of other mixed-reality systems. Our method is able to scatter light from reflective and refractive objects and create plausible caustics on nearby real or virtual objects. Figure 1.3 shows reflections, refractions, and caustics in a mixed-reality scenario, generated with our framework.

Figure 1.3: Reflections, refractions, and caustics in a mixed-reality environment, generated with our framework at interactive to real-time frame rates. The left figure shows a scene with different virtual objects on a real desk that are illuminated by a virtual spotlight. The virtual Utah teapot reflects the surrounding real and virtual objects. In the right figure, a virtual spotlight illuminates a virtual glass, resulting in a colored caustic on the real desk. The desk is also realistically refracted through the virtual glass.

We extend the traditional differential rendering method similar to Grosch [12] and apply this effect also to reflected or refracted objects. This includes a detailed analysis of all light paths, identifying the correct differential rendering buffer and a description of the back-projection. The two main parts of this thesis describe the radiance computation for reflective and refractive objects and their integration in a mixed-reality environment with differential rendering, see Chapter 4. Chapter 5 describes the light simulation of caustics in combination with differential rendering.

The proposed method achieves interactive to real-time frame rates, so user interaction is still possible. The technical aspects, with implementation details about our intersection approximation and the caustic algorithm, are explained in Chapter 6. The remaining part of this thesis compares different intersection approximations and presents several example images of light effects from reflective and refractive objects in Chapter 7. Finally, we discuss the limitations in Chapter 8 and show possible improvements for future implementations.

The following list briefly summarizes the main contributions of this thesis:

- Extension for differential instant radiosity to simulate light effects from reflective and refractive objects in a mixed-reality scenario.
- Handles various lighting situations for reflective and refractive objects. These objects can be real or virtual and are able to reflect or refract other objects, either real or virtual.
- Plausible simulation of caustics with a fast splatting method in image space. Reflective and refractive objects realistically scatter the incoming light onto real or virtual objects.
- Use a back-projection to apply differential rendering also for reflective and refractive objects. The involved surfaces are identified and missing cases are handled accordingly.

CHAPTER 2

Basic Concepts

This chapter gives a short survey of some basic principles and tools that are needed to analyze and simulate the light distribution in a mixed-reality environment. The rendering methods in this thesis do not have the intent to replicate the physically correct behavior of light in the real world. Our aim is to simulate a perceptually plausible light distribution in a mixed-reality environment. Nevertheless, our assumptions and simplifications build upon physical properties of light, so it is essential to have a basic understanding of them.

In a simplified model [1], the light rays emanate from a light source into a scene. Typically, in a mixed-reality environment the scene is populated with real and virtual objects. Each of these objects behave differently to the incoming light rays. According to the material attributes of the object, the light is absorbed and scattered. After all, some of these light rays reaches the sensor of an observer, which results in a visual perception of the environment. The first three sections in this chapter deal with the physical properties of light, the material attributes, and the light distribution in a scene. We summarize this information from three different books [1,9,32] that cover this topic in more detail.

Apart from that this chapter describes the RESHADE framework. This description covers the subsystem to capture the environment, the tracking of the object position and orientation, and the several techniques used in the rendering subsystem.

2.1 Light and Radiometry

Radiometry is the science that measures electromagnetic radiation. Further, light is a kind of electromagnetic radiation. To be more precise, it is a flow of photons that carries energy. Every photon has a certain frequency and wavelength that are proportional to its energy. Electromagnetic radiation exists at every wavelength, but only a very small range is visible for the human eye. This range is called the *visible spectrum* and lies approximately between 380 nanometers and 780 nanometers. The wavelength of the radiation characterizes the color of the light. Figure 2.1 shows a schematic representation of the visible spectrum. The energy of a photon

increases with the wavelength λ, i.e. violet photons carries more energy than red photons. Natural light sources consists of different intensities per wavelength whereas a laser light source has most of its energy at a single wavelength (i.e. a monochromatic light source) [1].

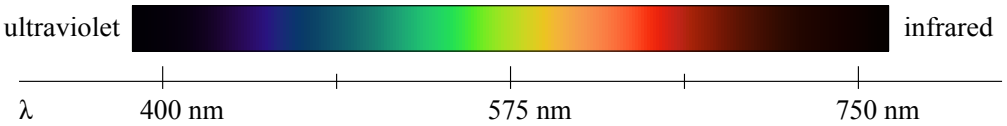


Figure 2.1: The visible spectrum of the human eye. It is a narrow band on the electromagnetic spectrum and lies approximately between 380 nm and 780 nm. (Color box from Gringer [11])

There are two different light models that describe certain lighting effects. Both of them built upon the *quantum optics* model. On the one hand there is the *wave model* that uses the wave properties of the light to describe the flow of photons. This model covers effects like diffraction or interference. They only occur when the light radiates very small objects (size is comparable to the wave length), so our rendering system does not support this model. On the other hand there is the *geometric optics* model that describes the flow of photons as a stream of particles. It makes several simplifications (i.e. the stream propagates on a straight line at infinite speed without external influences) and uses basic optical laws to reflect or transmit the incoming light rays on an object. Moreover, the wave-length of the light is vastly smaller than the radiated surface [9]. Thus, this model is perfectly valid for a rendering system in a mixed-reality environment.

The radiometric quantity to measure the amount of energy per unit time is called *radiant flux* ϕ. It is measured in the units of watt (W). For example a light source emits radiant flux. The quantities *irradiance* E and *radiosity* M (or radiant exitance) define the incoming and outgoing radiant flux per surface area. They are measured in the units of watt per square meter (W/m^2). Probably the most interesting quantity for a rendering algorithm is *radiance* L. It is the radiant flux per unit projected area per unit solid angle in the units of watts per steradian per square meter ($W/(steradian \cdot m^2$). The amount of radiance on a sensor of an observer determines the perception of the environment. Actually, a rendering method computes this quantity. Formally the radiance is:

$$L = \frac{d^2\phi}{d\omega dA \cos\theta}, \tag{2.1}$$

where $d\omega$ is the solid angle and $dA\cos\theta$ the projected surface area [32]. Figure 2.2 illustrates the radiance for a surface point x with a surface normal n.

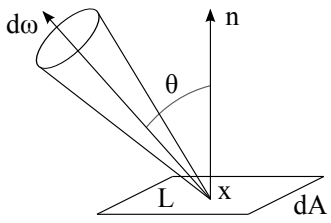


Figure 2.2: The radiance L depends on a position x and a direction. It is the radiant flux per unit projected area $dA\cos\theta$ per unit solid angle $d\omega$. Illustration after Dutré et al. [9].

Radiance has an important property in vacuum, which is called *invariance of radiance*, in which the amount of radiance leaving a surface in a specific direction is equal to the amount of radiance arriving at a surface from this direction. Our simplification neglects participating media, so the radiance is constant in a specific direction.

2.2 Bidirectional Reflectance Distribution Function

The material attributes of an object defines the impact of the arriving light on a surface. Some light might transmit through the object or gets reflected on its surface. Consequently, this affects the perception of the object by the observer. This section explains the *bidirectional reflectance distribution function* (BRDF) that determines how the surface of an object reacts to the incoming light. The BRDF for a surface point x is the ratio:

$$f_{brdf}(x, \omega_i, \omega_o) = \frac{dL_o(x, \omega_o)}{dE_i(x, \omega_i)}, \tag{2.2}$$

where dL_o is the outgoing radiance in the direction ω_o and dE_i is the irradiance from direction ω_i. The BRDF has the units of $steradians^{-1}$. Figure 2.3 shows the parameters of an isotropic BRDF. Contrary to an anisotropic BRDF, it is rotation independent. Hence, the orientation of the surface around the normal has no influence on the evaluation of the BRDF.

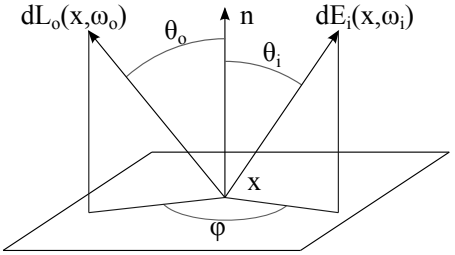


Figure 2.3: Schematic illustration of an isotropic bidirectional reflectance distribution function. An isotropic BRDF depends on a position x, an angle θ_i, and an angle θ_o. The two angles are formed between the surface normal n and the incoming radiance direction ω_i and the outgoing radiance direction ω_o. Illustration after Akenine-Möller et al. [1].

According to the laws of physics the BRDF fulfills the *Helmholtz reciprocity*. This principle states that switching the incoming and outgoing direction has no influence on the BRDF value:

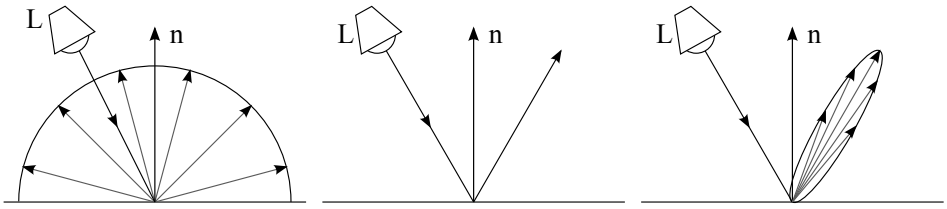
$$f_{brdf}(x, \omega_i, \omega_o) = f_{brdf}(x, \omega_o, \omega_i). \tag{2.3}$$

In addition, a physically based BRDF observes the law of *energy conservation*. Accordingly, the reflected radiance is always smaller or equal to the incoming irradiance:

$$f_{brdf}(x, \omega_i, \omega_o) = \int_\Omega f_{brdf}(x, \omega_o, \omega_i) \cos\theta_o d\omega_o \leq 1, \tag{2.4}$$

where the integral $\int_\Omega$ is evaluated over the hemisphere above the surface point x [32].

The relevant parameters for a BRDF can be acquired from real objects. This results in a huge amount of data that is hard to evaluate. Instead, analytical models describe a BRDF for different material characteristics. Figure 2.4 lists the different main categories. Diffuse surfaces equally distribute the incoming light in all directions, mirror surfaces reflect the incoming light into one specific direction and glossy surfaces reflect most of the light around one specific direction [9]. Usually, a BRDF is a combination of all these components.



(a) Diffuse surface (b) Mirror surface (c) Glossy surface

Figure 2.4: Schematic illustration of the main BRDF categories. It shows the BRDF for a diffuse surface, a mirror surface, and a glossy surface. Illustration after Pharr and Humphreys [32].

2.2.1 Ideal Reflection

A mirror like reflection of light, also named ideal specular reflection, underlies the *law of reflection* [1]. It states that the angle of incidence θ_i equals the angle of reflection θ_r. This means that the direction of the incoming light ray and the direction of the reflected light ray form the same angle with the surface normal. Figure 2.5a shows the derivation of the reflection vector r. Utilizing vector algebra, the light direction is projected onto the surface normal and added twice to the negative light direction:

$$r = 2(n \cdot l)n - l, \tag{2.5}$$

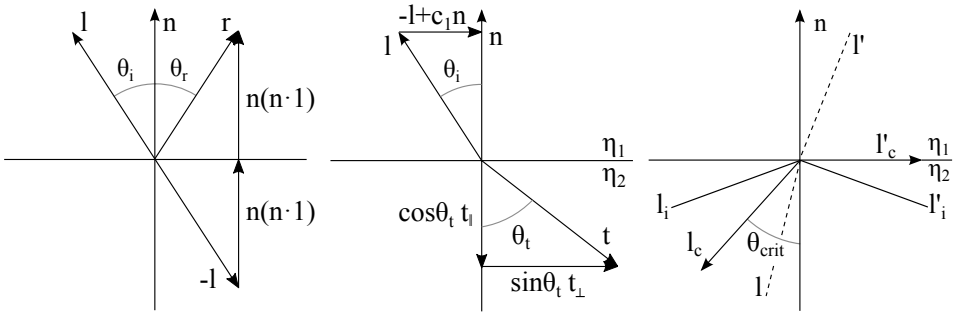
where n is the normalized surface normal, l the normalized direction to the light source and r the resulting reflection direction. Note that the three vectors are coplanar, i.e. the reflected direction r lies in the plane formed by the surface normal n and the light direction l.

2.2.2 Ideal Refraction

Refraction of light happens when the light interacts with a translucent object. Every time the light travels through an interface between two different materials (e.g. from air through a glass surface), it changes its direction. The *law of refraction* or *Snell's Law* [1] describes this behavior as:

$$\eta_1 \sin\theta_i = \eta_2 \sin\theta_t, \tag{2.6}$$

where η_1 and η_2 is the index of refraction of the two involved materials and θ_i the incident angle and θ_t the refraction angle. Heckbert [13] uses Snell's law to derive the refraction vector. The



(a) Reflection (b) Refraction (c) Total internal reflection

Figure 2.5: Derivation of the reflection direction, refraction direction, and total internal reflection. Figure (a) shows the computation of the reflection direction r from the light direction l and the surface normal n. Figure (b) illustrates the determination of the refraction direction t as a linear combination of the vector $t_{||}$ and the vector $t_{\perp}$. Figure (c) shows total internal reflection that occurs when the incident angle exceeds a critical angle θ_{crit}. Illustrations after Akenine-Möller et al. [1].

idea is to express the refraction vector as a linear combination between the vector $t_{||}$ and the vector $t_{\perp}$, see Figure 2.5b. The vector $t_{||}$ parallel to the unit surface normal n

$$t_{||} = -n \tag{2.7}$$

and the vector $t_{\perp}$ perpendicular to the unit surface normal n

$$t_{\perp} = \frac{-l + c_1 n}{||-l + c_1 n||} = \frac{-l + c_1 n}{\sin\theta_i}, \tag{2.8}$$

where $c_1 = \cos\theta_i = n \cdot l$. The linear combination of the refraction vector t is

$$\begin{aligned} t &= \cos\theta_t t_{||} + \sin\theta_t t_{\perp} \\ &= -\cos\theta_t n + \sin\theta_t \frac{-l + c_1 n}{\sin\theta_i} \end{aligned} \tag{2.9}$$

and expanded with $\eta = \frac{\eta_1}{\eta_2} = \frac{\sin\theta_t}{\sin\theta_i}$ from the law of refraction

$$\begin{aligned} t &= -\cos\theta_t n - \eta l + \eta c_1 n \\ &= (\eta c_1 - \cos\theta_t) n - \eta l \\ &= (c_1\eta - c_2) n - \eta l, \end{aligned} \tag{2.10}$$

where $c_2 = \cos\theta_t = \sqrt{1 - \sin^2\theta_t} = \sqrt{1 - \eta^2 \sin^2\theta_i} = \sqrt{1 - \eta^2(1 - c_1^2)}$.

Equation 2.10 is used to compute the refraction direction, where n is the surface normal, l the direction to the light source and η the ratio of the refractive indexes. The refraction index of air is one, for all other translucent materials it is greater than one.

A special case may occur when light travels from a higher refractive material into a lower refractive material. When the incident angle θ_i exceeds a critical angle θ_{crit} all light is reflected, instead of refracted, which is called *total internal reflection* [7]. Light rays incident at the critical angle refract them perpendicular to the surface normal, illustrated by Figure 2.5c. The critical angle follows from Snell's law, defined in Equation 2.6:

$$\theta_{crit} = sin^{-1}\frac{\eta_t}{\eta_i}, \tag{2.11}$$

where $sin\theta_t = 1$ for a refraction angle of $90°$. Note that in case of total internal reflection, the expression c_2 can not be evaluated because the value under the square root is negative.

2.3 Rendering Equation

The rendering equation from Kajiya [18] is a mathematical framework to describe the light distribution in a scene. A common representation of the rendering equation is the *hemispherical formulation*:

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_{\Omega_x} f_{brdf}(x, \omega_i, \omega_o) L_i(x, \omega_i) \cos\theta_i d\omega_i, \tag{2.12}$$

where Table 2.1 describes the involved terms.

Term	Description
$L_o(x, \omega_o)$	The outgoing radiance from the surface point x in the direction ω_o.
$L_e(x, \omega_o)$	The direct emitted radiance from the surface point x in the direction ω_o.
$\int_{\Omega_x}$	The integral evaluates to the total reflected radiance from the surface point x in the direction ω_o.
$f_{brdf}(x, \omega_i, \omega_o)$	The BRDF for the surface point x, the incoming direction ω_i and the outgoing direction ω_o.
$L_i(x, \omega_i)$	The incoming radiance from direction ω_i. Furthermore, this term also represents an outgoing radiance from another surface point x' in the direction $-\omega_i$ denoted with $L_o(r(x, \omega_i), -\omega_i)$, where the function $r(x, \omega_i)$ determines the surface point x'.

Table 2.1: Involved terms in the rendering equation.

In other words, the rendering equation computes the outgoing radiance $L_o(x, \omega_o)$ as the sum of the direct emitted light L_e from surface x and all the incoming light on surface x that is reflected in the direction ω_o. Figure 2.6 visualizes the involved terms of the rendering equation.

It is problematic to solve this integral equation analytically. The complexity lies in its recursive composition and the occurrence of the unknown term (the outgoing radiance L_o) on the left and right hand side of the equation. However, there is a formal solution that approximates the result with a step-wise evaluation. Several rendering algorithms take advantage of this and iteratively compute a solution for the outgoing radiance.

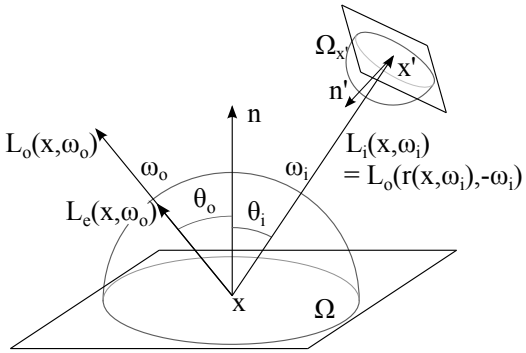


Figure 2.6: The involved terms in the rendering equation. The outgoing radiance $L_o(x, \omega_o)$ from the surface x is the sum of the emitted light $L_e(x, \omega_o)$ and all incoming light that is reflected from point x. The outgoing radiance $L_i(x, -\omega_i)$ is evaluated recursively, shown by the hemisphere Ω'_x above surface x'. Illustration after Dutré et al. [9].

2.4 Reciprocal Shading Framework for Mixed Reality

The existing framework that we use to integrate the reflective and refractive light effects is part of the RESHADE project [42], an abbreviation for reciprocal shading. In short, the scope of this project is to realistically combine real and virtual objects in a mixed-reality environment. This means that the light influence between different types of real and virtual objects should behave perceptually plausible to a human observer.

Figure 2.7: Output image, created with the RESHADE framework. The spotlight illuminates the virtual Stanford dragon that causes a green color bleeding on the real teapot. The markers are used to identify and track the position and orientation of the objects. Image courtesy of Knecht et al. [22]

2.4.1 Tracking and Capturing

The RESHADE framework combines different subsystems to handle the requirements of a mixed-reality scenario. An important aspect in such a scenario is the user interaction. In our case, the user is able to manipulate the position and orientation of real and virtual objects. Moreover, it is possible to change the viewpoint of the camera as well as the adjustment of the light sources. Therefore, it is necessary to identify the different entities and to track their relative location to a coordinate origin. The responsible subsystem uses the *Studierstube Tracker* [24] library and special markers (BCH ID-marker) for the identification. Figure 2.7 shows a typical setup in a mixed-reality environment.

Besides tracking, the framework uses a conventional camera to capture the mixed-reality scene every frame. Note that the tracking system processes the camera image, so this capturing is one of the first tasks in the framework. In addition, a fish-eye camera records the surrounding scene and stores it in a hemispherical environment map for further usage.

The following section explains the rendering subsystem of the RESHADE framework. It combines several different rendering methods to simulate the light distribution between real and virtual objects. However, before we describe the light simulation in detail we briefly present the underlying methods.

2.4.2 Instant Radiosity

Instant radiosity from Keller [21] is a *global illumination* algorithm that is able to approximate a solution for the rendering equation. It uses a set of virtual point lights and distributes them on the illuminated surfaces. Each virtual point light (VPL) represents an additional light bounce that possibly illuminates another surface, i.e. indirect illumination.

Originally, a ray-tracer determines the illuminated surfaces for the VPL placement. Alternatively, there exists a faster method for spotlight sources that uses *reflective shadow maps*, invented by Dachsbacher and Stamminger [6]. This algorithm exploits the standard shadow mapping technique that identifies the visible surfaces from a primary spotlight source. It stores additional attributes in the shadow map, which specify a VPL on a visible surface point, like the incident light direction or the radiant flux.

Moreover, it is important to account for the occlusion between all VPLs and a possible illuminated surface during the lighting computation. Conventional shadow maps are too slow to handle this large amount of VPLs that are needed for an appealing lighting result. Actually, it is sufficient to approximate this visibility test with an *imperfect shadow map* as proposed by Ritschel et al. [36]. This method renders an incomplete geometric representation (point splats) of the objects into a low resolution shadow map. Due to the low frequency nature of indirect light, this inaccuracy is appropriate and improves the performance vastly.

2.4.3 Differential Rendering

Differential rendering from Debevec [8] is a fundamental method to augment a captured image from a camera with virtual content. It computes two different parts of radiance buffers. One part contains the radiance from the light simulation of all real and all virtual objects. The other

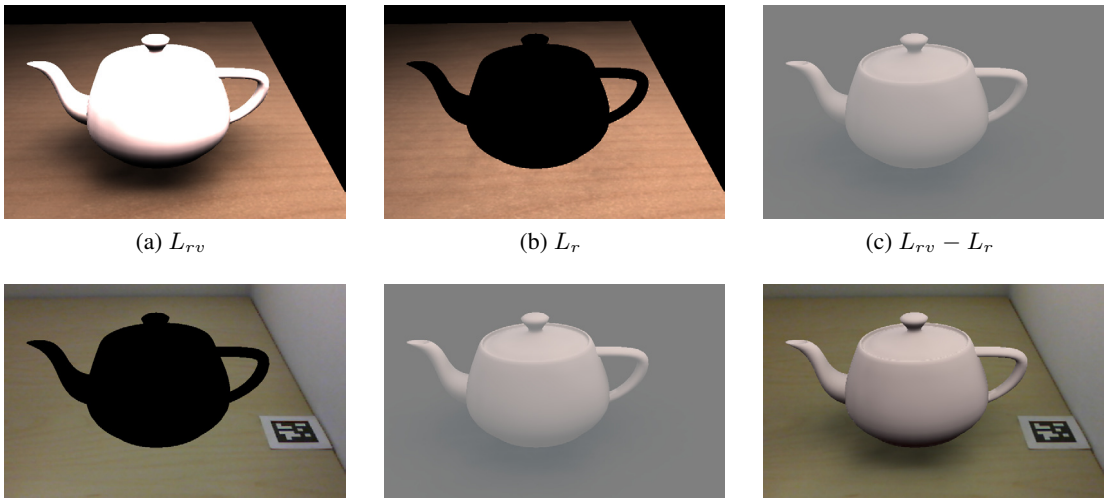
differential rendering part contains the radiance from all real objects that are only affected by real components, e.g. real light sources, real virtual point lights, and real shadow casters.

Consider, all real objects that contribute to the light simulation are pre-modeled and have approximated material attributes. Unfortunately, an accurate determination of the material attributes is hardly possible. Mainly, because the BRDF is an approximation and cannot cover all the subtle material details. Differential rendering minimizes this error by adding the difference between the two radiance parts to the captured background image of the camera:

$$L_f = L_b + L_{rv} - L_r, \tag{2.13}$$

where L_f is the final output image, L_b is the masked background from the camera, L_{rv} is the differential part with the radiance from all real and virtual objects, and L_r exclusively contains the radiance from only real components.

Figure 2.8 visualizes the differential rendering buffers for a scene containing a real desk and a virtual teapot, illuminated by a real environment light. The first row shows the difference between the real-virtual buffer L_{rv} and the real buffer L_r, in which the desk contributes to both buffers and the virtual teapot only to the buffer L_{rv}. Note that the color values in Figure 2.8c and Figure 2.8e are adjusted to represent also negative values, so gray means no difference. This difference is added to the masked background L_b as illustrated by the second row.



(a) L_{rv} (b) L_r (c) $L_{rv} - L_r$

(d) L_b (e) $L_{rv} - L_r$ (f) $L_f = L_b + L_{rv} - L_r$

Figure 2.8: Differential rendering buffers. It adds the difference between the two radiance buffers L_{rv} and L_r to the masked background L_b: $L_f = L_b + L_{rv} - L_r$.

The idea is that for an exact material representation the computed radiance for real objects L_r is equal to the captured image from the camera L_b. Note that the masked background L_b only contains values for visible real objects because the differential rendering effect is just applied to real objects. On the other hand, virtual objects directly use the computed radiance from the buffer L_{rv} for the final output image, hence the background is masked out for virtual objects.

2.4.4 Light Path Notation

The *light path notation* from Heckbert [14] is a formal expression that classifies the way of a photon from the light source over different surfaces to the observer. This notation is helpful to analyze the light distribution in a scene, to describe the potential of a light simulation, and to compare different rendering methods.

In general, the light path notation contains symbols for the light, the eye, and the interacting surface types. Quantifiers from the regular expression concept extends the light path notation and simplify its usage. Table 2.2 summarizes the symbols and quantifiers of the light path notation that we use in this thesis.

Nr	Symbol	Description
1	L	Represents the light source
2	E	Represents the eye point or the observer
3	D	Classifies a diffuse surface interaction
4	S	Classifies a specular surface interaction, e.g. a reflective or refractive surface
5	*	Arbitrary number of surface interactions

Table 2.2: Symbols occurring in the light path notation.

A light path expression begins with the symbol L for the light source and ends with the symbol E for the eye point. Moreover, the symbols D and S are appended to the path expression for each surface interaction (i.e. a light bounce on a diffuse or specular surface). For example the light path LDE describes a direct illuminated diffuse surface, whereas the light path $LDDE$ contains a light bounce on a diffuse surface, as described before with the virtual point light in Section 2.4.2.

2.4.5 Differential Instant Radiosity

The rendering subsystem builds upon *differential instant radiosity* from Knecht et al. [22]. As the name implies, this method combines differential rendering and instant radiosity to simulate the light distribution in a mixed-reality framework. It produces convincing results that simulates direct lighting on a diffuse surface (LDE) and indirect illumination between diffuse surfaces ($LDDE$), possibly with multiple light bounces ($LDD*E$) either on real or virtual objects. Figure 2.9a illustrates these different cases with a spotlight source.

The framework supports several types of light sources. An environment light source imitates the real ambient light. It is approximated with a set of virtual point light (VPL) sources that depend on the real lighting conditions in the mixed-reality scene, see Figure 2.9b. This approximation utilizes the hemispherical environment map captured from the fish-eye camera. The algorithm identifies the brightest areas on this image and uses importance sampling to place the light sources onto a hemisphere. Remember that the fish-eye camera captures the surrounding scene once per frame, so the VPLs are continuously repositioned according to the influence of the real light.

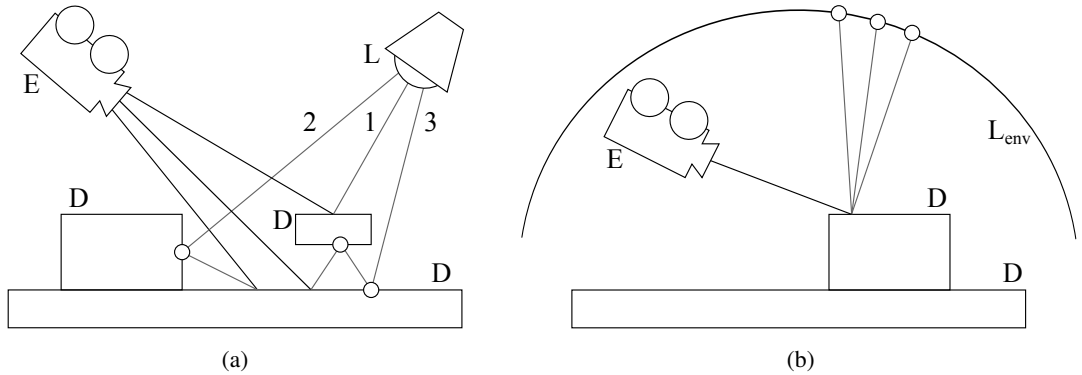


Figure 2.9: Possible light paths with differential instant radiosity. In Figure (a), the first path LDE shows a direct illumination from a spotlight source. The second path $LDDE$ is an indirect illumination between diffuse surfaces. The third path $LDDDE$ results from multiple light bounces on diffuse surfaces. Figure (b) shows that the real ambient light is approximated with a set of virtual point lights, which are located on a hemisphere.

The spotlight, which can be real or virtual, is a primary light source in the rendering subsystem. This means that it is able to directly illuminate a diffuse surface on the one hand. On the other hand, it is also possible to place VPLs on the illuminated surface to simulate the indirect illumination. Spotlights use the aforementioned reflective shadow map algorithm for the placement of the VPLs. Furthermore, a special type of primary light source also supports multiple light bounces, in which an emitted VPL distributes an associated set of VPLs in turn, illustrated with light path three in Figure 2.9a. Consider that all VPLs uses imperfect shadow maps for the occlusion test.

Compared to the original differential rendering algorithm, differential instant radiosity renders the scene only once to determine the two differential radiance buffers L_r and L_{rv}. All occurring light paths are pre-identified. Every component in the path, like the light source, the VPL, or the surface, has a *real* or *virtual* flag. These flags are evaluated during the radiance computation and determine the contribution to the buffers L_r and L_{rv}. The buffer L_r is only modified when all components in the path are real.

All components in the light path are marked as real or virtual, except the eye point as illustrated in Figure 2.10. For instance, the computed radiance from the first path $L_v D_R E$ only affects the buffer L_{rv} because the real surface D_r is illuminated by a virtual light source L_v. The same holds for the second path that contributes to L_{rv}. Note that the virtual surface D_v invalidates the visible point p' in the camera image. Hence, this area is masked out in the background image L_b, i.e. the differential rendering effect is just applied to real objects. In Figure 2.10b, path three $L_r D_r E$ contributes to both buffers L_r and L_{rv}, thus these two parts cancel out and the captured value from L_b remains. Consider that only real shadow blockers affect the radiance computation. Hence, differential instant radiosity ignores the virtual shadow blocker D_v in the radiance computation (path four).

Taken together, differential instant radiosity computes the radiance from all primary light

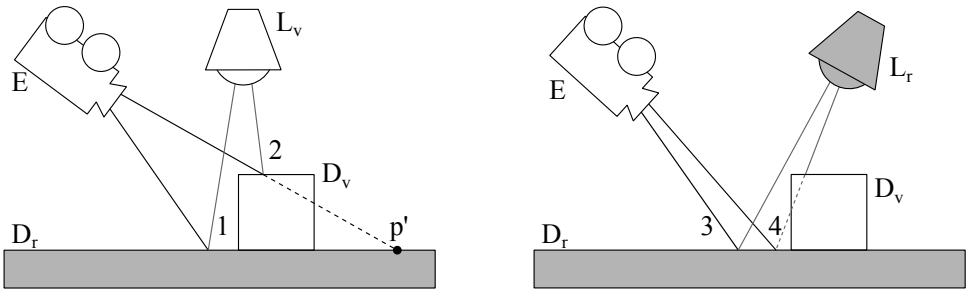


Figure 2.10: Concept of differential instant radiosity. The first and second path only affects the buffer L_{rv}. The third path contributes to both buffers L_r and L_{rv}. Only real shadow blockers affect the real part, see path four.

sources and all VPLs. Note that temporal coherence reduces the flickering from the limited amount of VPLs. Furthermore, all radiance computations happen in high dynamic range, so a tone mapper brings the buffers L_r and L_{rv} into low dynamic range before it adds their difference to the masked background L_b. Table 2.3 summarizes the possible light paths that are simulated with differential instant radiostiy.

Nr	Path	Description
1	LDE	Direct illumination from a light source.
2	LDDE	Indirect illumination of a diffuse surface.
3	LDD*E	Indirect illumination of diffuse surface with multiple bounces.

Table 2.3: Possible light paths with differential instant radiostiy

CHAPTER 3

Related Work

The following part discusses several techniques that are able to simulate reflections, refractions and caustics in real-time. Moreover, this chapter presents other mixed-reality systems that integrated reflective and refractive objects. We compare them with our approach and list the main differences.

3.1 Reflection

Ray tracing [1] is a powerful algorithm that is able to find a solution for the rendering equation. Apart from the fact that it is a global illumination approach, its strengths are emerging at a scene with many reflective objects. The principle of the algorithm is easy to understand. However, it is difficult to implement this technique in a fast way. The following section presents alternative methods that approximate the reflections in image space but run at real-time frame rates.

Planar Reflections

One of the most elementary reflection method is on a planar surface [25], like a flat mirror. Planar reflections are ideal reflections and underly the law of reflection, see Section 2.2.1. Another point to observe, the surface normal is identical for every incident ray. Hence, following the incident ray is sufficient to approximate a planar reflection. Reflecting either the view point or the objects above the plane accomplishes this task, see Figure 3.1.

It is important that reflected objects are only rendered into the area of the mirroring surface, marked with the symbol S. Otherwise, objects may appear at areas that do not belong to a reflector. For example, the diffuse object D_2 would wrongly reflect parts of diffuse object D_1, because it lies in the camera frustum. Former techniques exploits the stencil buffer to mark the reflector area. Whereas, on todays GPU architecture, it is beneficial to render the reflected objects into a texture and project it onto the reflector object.

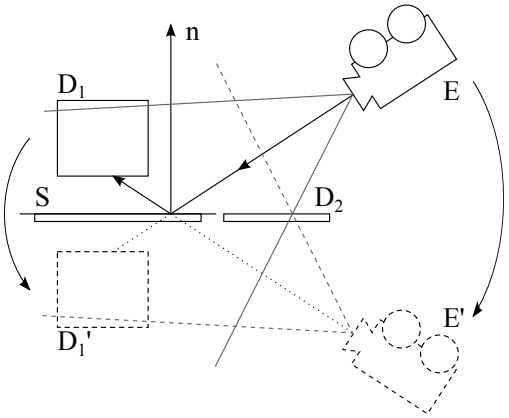


Figure 3.1: Planar reflection. Reflection above the planar surface S by mirroring the object D or the camera E above the reflective plane with the normal n.

Environment Mapping

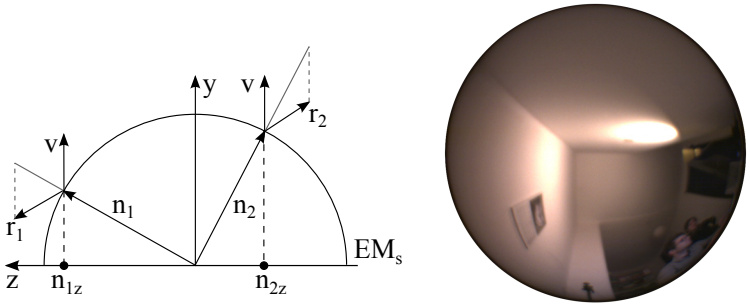
Environment mapping, introduced 1976 by Blinn and Newell [5], is able to approximate reflections on curved objects. Basically, they store the radiance of the surrounding scene in a two dimensional map and uses computed look-up coordinates to retrieve the reflected environment from this map.

Prominent extensions are *sphere mapping* and *cubic environment mapping*. Sphere mapping [1] uses a hemisphere to store the radiance values of the hole environment. For instance, a photograph of a real metallic sphere results in a sphere map that reflects the hole environment. Sphere maps are view dependent, which allows to compute the look-up coordinates only from an arbitrary reflection vector. The look-up vector corresponds to the normal, as illustrated in Figure 3.2a, where the normal n results from a vector addition of the view vector v and the reflection vector r. For a fixed view vector v, with coordinates $(0, 1, 0)$ it is computed as:

$$n = \frac{(r_x, r_y + 1, r_z)}{\sqrt{r_x^2 + (r_y + 1)^2 + r_z^2}}, \tag{3.1}$$

where r is the normalized reflection direction. The normal n contains the look-up coordinates (n_x, n_z) for the sphere map. We utilize a similar principle to determine radiance values from the captured environment of a fish-eye camera. The image in Figure 3.2b shows the captured environment from a fish-eye camera. Note that in this case, the picture only contains objects in front of the camera, so it is not possible to reflect objects from behind as shown with ray r_1 in Figure 3.2a.

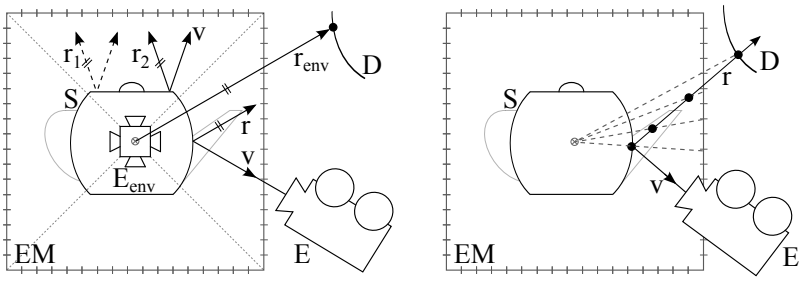
Cubic environment mapping, invented by Green [10] uses the six faces of a cube to store the surrounding environment. Compared to sphere mapping, it is view independent, has a better sampling characteristic, and is easier to generate in a renderer [1]. Figure 3.3a visualizes the generation, where the view point E_{env} is placed at the center of the reflector S. The surrounding environment is rendered for each cube face. Finally, a three dimensional reflection vector r_{env} is used to retrieve the reflected radiance from the environment map EM.



(a) Sphere mapping concept (b) Fish-eye picture

Figure 3.2: Concept of sphere mapping. Figure (a) shows that the look-up coordinate corresponds to the normal n, computed as the sum of vectors v and r. The vector r_1 reflects the environment behind the sphere. Illustration after Akenine-Möller et al. [1]. Figure (b) visualizes a hemispherical map captured with a fish-eye camera.

Reflections from classic environment mapping works well for distant objects, in which the look-up coordinate solely depends on the direction of the reflection. Unfortunately, it is inaccurate for objects near to the reflector or for flat surfaces. Figure 3.3a illustrates such a case, where the reflected ray r_1 and r_2 point to the same value in the environment map, although they should reflect a different surface. Different solutions exists to handle the limitations. The general idea is to additionally include the position of the reflected surface into the look-up coordinate.



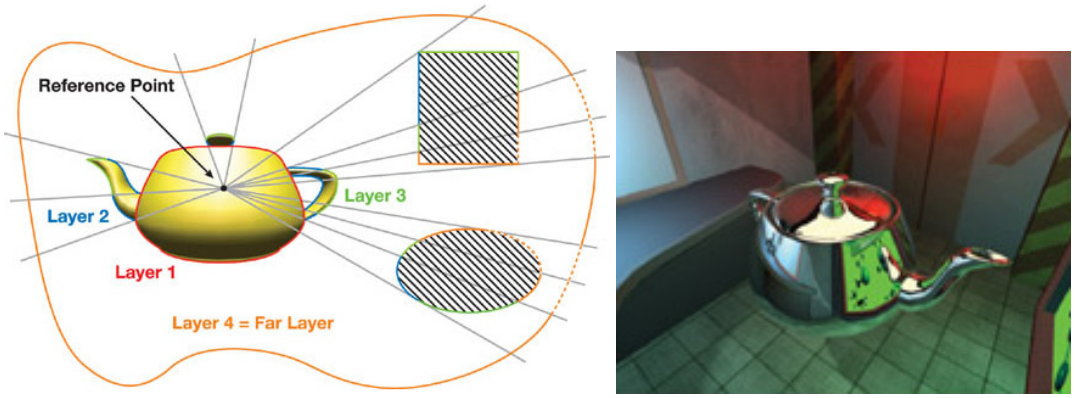
(a) Cubic environment mapping (b) Distance impostor

Figure 3.3: Cubic environment mapping. Only the direction of the reflected ray is used to determine the reflected surface. Figure (b) illustrates how to incorporate the ray position to find a more accurate intersection point by marching along the reflected ray.

Bjorke [4] uses a proxy geometry for the surrounding environment. His method computes the intersection between the reflection ray and the proxy geometry (e.g. a sphere) and modifies the look-up coordinate accordingly. Usually, the surrounding scene does not match with the proxy geometry, which results in visible artifacts, i.e. the structure of the proxy geometry emerges in the reflection.

The method of Szirmay-Kalos et al. [41] additionally stores the distance from the reflection center to the reflected surface point in the environment map. This so called *distance impostor* is used to iteratively compare the distance along the reflected ray with the distance in the environment map. Figure 3.3b illustrates the marching on the reflected ray r. Starting from the reflection position, a brute force approach would step along the ray, comparing every corresponding environment map entry until it finds an accurate intersection point. To improve this iteration, their method finds two enclosing intersection points with a fast linear search along the ray. Finally, a secant search computes a refined intersection location for these two points.

The work from Umenhoffer et al. [44] extends this idea and uses several environment maps to store the surrounding scene. This allows to store surfaces that are occluded from the reference point of the environment map, illustrated in Figure 3.4a. The different layers can be obtained by depth peeling or by rendering a fixed number of layers. Besides the distance, these layers also contain the normal and the material properties of the surface. Each layer is intersected with the reflection direction and the closest intersection determines the environment map layer. Depending on the material properties of the surface, the algorithm returns the radiance or computes an additional ray that is used to intersect the set of layers in turn. Hence, their technique is able to simulate reflections and refractions with multiple bounces, including self-reflections (see Figure 3.4b).



(a) Environment map layers (b)

Figure 3.4: Figure (a) shows the concept of layered cubic environment maps that are able to store attributes also of occluded surfaces. Figure (b) shows a self-reflection. Images courtesy of Umenhoffer et al. [44]

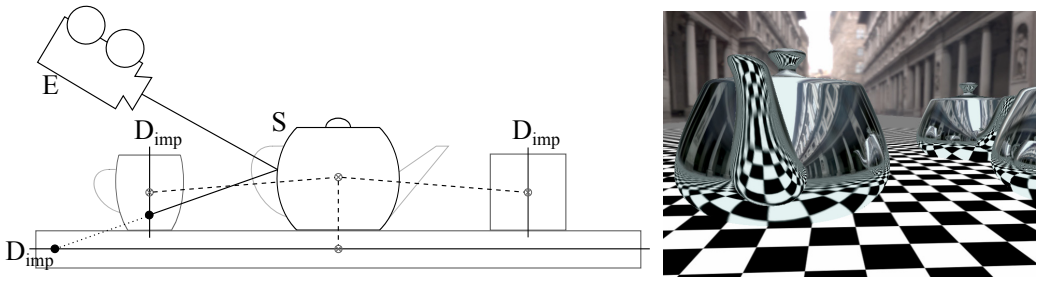
The two aforementioned environment mapping method runs both at real-time frame rates. However, there are difficulties to determine the correct step size along the ray, which may result in artifacts.

Reflective Impostors

Instead of environment maps, Popesco et al. [35] uses *impostors* to handle reflections at real-time frame rates. An impostor is a rectangle with a two dimensional texture that contains the rendered

geometry of an object. This simplification is used for the reflection computation, instead of the original geometry. Figure 3.5a illustrates the principle of impostor reflections.

Every reflective object has a set of impostors. Each element in this set represents an object that should appear in the reflection. Changing the orientation or the position of the object, or the position of the reflector invalidates the impostor. In such cases, the impostor is newly created.



(a) (b)

Figure 3.5: Figure (a) illustrates impostor reflections. The impostor set of the reflective teapot S contains three impostors. Each is intersected with the reflection ray, instead of the original geometry, and the closest distance specifies the intersection point. Figure (b) shows an example of multiple reflections with impostors, image courtesy of Popescu et al. [35]

To determine the reflected radiance, the algorithm iterates over all impostors in the corresponding set and intersects each with the reflection vector. The nearest intersection point identifies the radiance of the reflected surface. Even higher order reflections are possible but that would require to treat an impostor as reflective, see Figure 3.5b. Moreover, their work extends the simple billboard impostors with depth information. They intersect the reflection ray with the depth map from the impostor to produce higher quality reflections.

3.2 Refraction

Image-space refraction methods approximate the interaction of light with translucent objects. More specifically, they apply the law of refraction, see Section 2.2.2, to the view ray, which has the same effect. The direction of the incident vector changes on every interaction with a surface boundary, i.e. on entry and exit points.

Single refractions compute the bending only once, at the entry point. The resulting refracted direction can be used to look-up the radiance from an environment map, or with any other method from Section 3.1. Single refractions are useful for relatively thin objects, for instance a glass window.

Cubic Distance Impostors

Usually, general shapes of translucent objects need more effort. The aforementioned distance impostor technique from Szimary-Kalos et al. [41] is also capable to simulate multiple refractions. In place of the surrounding scene, the distance impostor stores the depth and normal of

the refracting object, e.g. the teapot as seen from inside. Moreover, this time the intersection algorithm uses the refraction operation instead of the reflection. To compute the refraction, the algorithm starts identical as in the single case but all successive surface interactions use the distance impostor technique to approximate the refraction. Distance impostors can be pre-computed for static geometry. Their approach works well for convex shapes, where most points are visible from the object-center.

Refractions on two Surface Interfaces

The human visual system is tolerant to refraction approximations. For this reason it is acceptable to limit the refraction computation to the entry and exit point. The technique from Wyman [46] computes the refraction direction from two surface interactions.

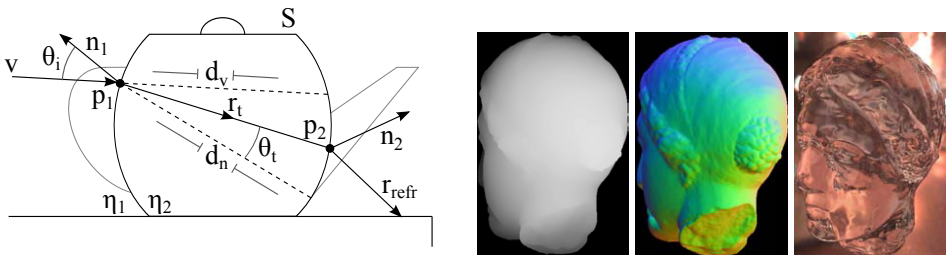
The algorithm uses two passes. The first pass draws the back-faces of the refractive object from the camera position and stores the depth and the normal in a texture. The subsequent pass draws the front faces and computes the refraction direction for the first surface intersection. Without ray tracing, it is tricky to determine the location of the second surface intersection. From Figure 3.6a we observe that point p_2 lies on the ray $p_1 + t \cdot r_t$, where t is the ray parameter to estimate. Therefore, Wyman estimates t with the distance between the front and back-faces d_v, which is computed with the stored depth values from the first pass. This gives an adequate approximation for convex shapes and a low index of refraction ratio. For higher ratios, the refracted ray r_t tends into the direction of the inverted normal n_1. For this reason, Wyman pre-computes the distance to the back-face d_n for every vertex. Interpolating between d_v and d_n gives a preciser estimate for ray parameter t:

$$t = \frac{\theta_t}{\theta_i} d_v + (1 - \frac{\theta_t}{\theta_i}) d_n, \tag{3.2}$$

where θ_i is the incident angle and θ_t the refraction angle. The normal on the second surface is determined from the normal buffer of the first pass by generating projective texture coordinates from the second surface location. Finally, they retrieve the radiance from an environment map with the computed refraction direction. Figure 3.6b illustrates the different texture buffers.

In a follow up, Wyman [47] proposes an extension to refract nearby geometry. The extended algorithm draws the objects behind the refractive object in an additional pass into a background texture. This texture is mapped onto a plane that acts as proxy geometry for the background scene. Instead of the environment map, the refracted ray intersects this background plane. Storing also the depth in the background texture allows an iterative refinement of the final intersection position, resulting in a more accurate refraction. Note that the reflected or refracted position must lie inside the field of view, otherwise the background texture contains no valid color information.

Pre-computing the distance to the back-faces d_n is only feasible for static geometry. In their work, Oliveira and Brauwers [30] present a method that handles also deformable objects. This approach performs a ray-intersection with a depth texture to find the position on the second surface. The first pass stores the depth and normal of the back-faces in a texture, identical to the previous method of Wyman. In a subsequent pass, minimum and maximum depth values are extracted from this depth texture, which limit the search interval for the ray-intersection. The



(a) (b)

Figure 3.6: Figure (a) illustrates a refraction on two surface interfaces. The second surface point p_2 lies on the refraction ray r_t and is estimated with a weighted combination of the distances d_v and d_n. The images in Figure (b) shows the distance between the front and back surface, the normals at the back-faces and the final refraction. Images courtesy of Wyman [46].

third pass computes the refraction ray for the first surface. Moreover, the algorithm marches along the refraction ray and checks for an intersection with the depth texture. Essentially, the depth of the projected ray position is compared with the corresponding value in the depth texture. This resulting position is used for the second refraction.

3.3 Caustic

Reflective and refractive objects scatter the incoming light, resulting in areas with a higher concentration of photon energy. This light focusing is known as a caustic. Figure 3.7 illustrates a caustic on a diffuse surface from a refractive sphere.

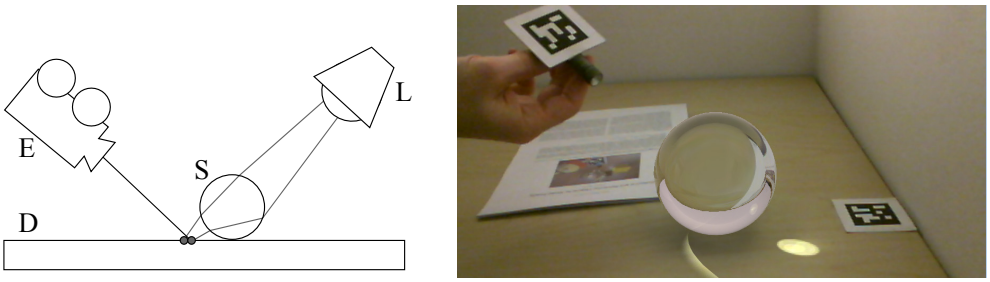


Figure 3.7: A caustic is a light focusing due to the scattering on reflective or refractive objects.

Backward Ray-Tracing

The two fundamental principles to simulate caustics arise from ray tracing. In his work, Arvo [3] reverts the ray tracing process, starting the rays from the light source, instead of the camera. In a

pre-processing step, a light map gathers the incoming light for every surface in texture space. A classic ray tracing pass, from the camera, additionally uses these maps to simulate the caustics.

Photon Mapping

Jensen [17] presents another technique, called *photon mapping* to simulate the effect of caustics. It is a full global illumination method, able to approximate a solution for the rendering equation. This method shoots photons from the light source, traces them through the scene and finally stores the last hit position on a diffuse surface in a spatial data structure, i.e. photon map. Jensen generates two different types of photon maps: First, a global photon map for the general light distribution, to speed up the subsequent ray tracing pass and second, a photon map with a higher resolution for every object that is able to create a caustic. For caustic photon maps, the method shoots the photons directly towards the caustic caster, resulting in a dense distribution of photons. Caustics are simulated directly with this photon map. The algorithm queries the photon map (remind, it is a spatial data structure) and collects the arriving flux of the N nearest neighbors within a sphere of radius r. The total flux is weighted by the occupied area of the circle with radius r. This filtered flux is used to illuminate the diffuse surface.

Caustics in Image Space

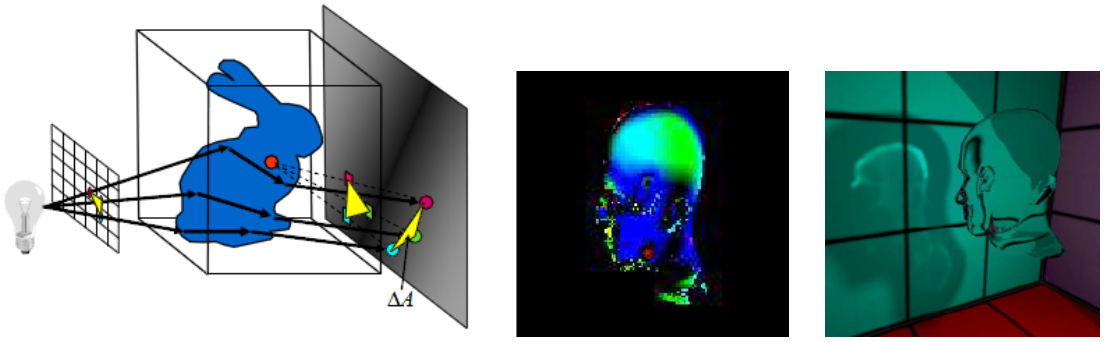
Image-space methods borrow ideas from both approaches to simulate caustics at real-time frame rates. Essentially, they render the caustic caster from the location of the light source and uses an image-space approximation of ray tracing (as discussed in Sections 3.1 and 3.2) to identify the final diffuse surface. This information is stored in a texture, representing the photon map. This photon texture is later used to visualize the caustic patterns.

Szimary-Kalos et al. [41] uses distance impostors to locate the diffuse surface position. Their method stores the arriving photon power, the texture identification, and texture coordinates of the surface in a photon texture. In a subsequent pass, they utilize this map to render point splats, which are modified by the BRDF, to visualize the caustics. The filtering of the arriving flux depends on the splat size, which must be manually adjusted.

Each texel in the photon map corresponds to an emitted photon. Functionally, this texture imitates the spatial data structure of the photon mapping algorithm. Consider that photon energy spreads out and also influences the neighboring area. Therefore, Jensen filters the photon energy with a nearest neighbor search. However, such queries in a spatial data structure do not fit well on todays GPU architectures, due to their random access nature.

Wyman and Davis [49] present a novel method to imitate this filtering, which gathers the photon energy in a separate caustic buffer. They create the photon map similar to the aforementioned technique. Instead of a direct visualization, the algorithm splats the photons in a caustic buffer, which can happen in light or screen space. A subsequent pass filters this buffer by accumulating the light contribution of neighboring texels and weights it by the occupied area. This follows from the idea that neighboring texels in the caustic buffer represents coherent areas on the hit diffuse surface. Finally, this caustic buffer modulates the color of the diffuse surface to simulate the caustic pattern.

Alternatively, the approach from Umenhoffer et al. [43] splats the photons into an environment map (i.e. a caustic buffer) that acts as a further point light source, see Figure 3.8. As before, it draws the caustic caster from the view of the light and approximates the final hit position on the diffuse surface with a layered environment map [44]. The resulting photon map contains the arriving flux and the direction to the center of the caustic buffer. A subsequent pass, generates triangles from the photon buffer and draws them into the caustic buffer. This buffer, which is an environment map, stores the accumulated irradiance and the distance with respect to its center. In a last step, the caustic buffer acts as an additional point light source that creates the caustic effect. Note, that this illumination only affects surfaces with a similar distance than the one stored in the caustic buffer. Splatting into the caustic buffer imitates the filtering process, so this technique omits the manual adaption of the splat size.



(a) Caustic triangle (b) Photon hits (c) Caustics

Figure 3.8: Caustics triangles are generated from the photon hits and rendered into an environment map. This map acts as an additional light source and visualizes the caustics. Images courtesy of Umenhoffer et al. [43].

3.4 Mixed Reality

As already mentioned, reflective and refractive light effects are difficult to integrate in a mixed-reality environment. This section provides an overview of previous work on this topic and elaborates the differences to our method.

One of the first methods that incorporate reflective and refractive objects into a mixed-reality scene comes from State et al. [40]. In their approach, they place a real metallic or glass sphere into the scene. They extract the areas occupied by this object from the camera image and remap them onto a reflective or refractive object. Originally, this is used to show the potential of their tracking algorithm but it also produces convincing reflections on virtual objects. However, this method is only able to reflect the real environment, see Figure 3.9a.

In his fundamental work about differential rendering, Debevec [8] augments the camera image with reflective and refractive objects, see Figure 3.9b. As opposed to our method, the differential rendering effect is ignored for these objects. For example, the differential effect

(a) (b)

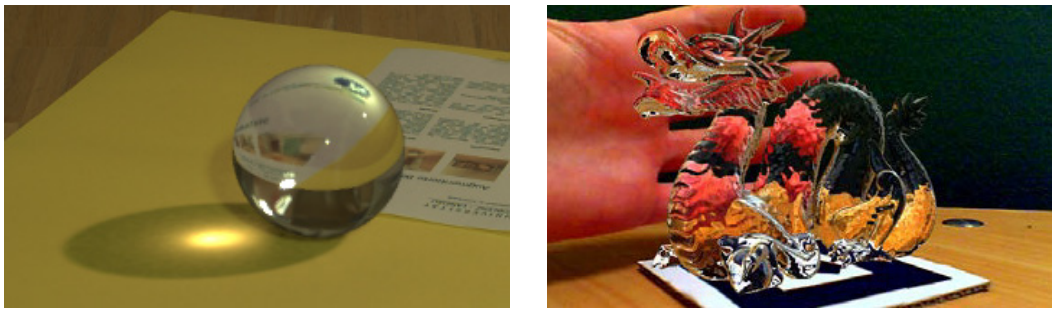
Figure 3.9: In Figure (a), a real metallic sphere is placed into the scene, extracted from the camera image and mapped onto the virtual teapot to simulate the reflection, Image from a presentation video of State et al. [40]. Figure (b) shows a mixed-reality scene with reflective and refractive objects from Debevec [8].

is missing on a reflected real surface, so only an approximated representation (i.e. L_{rv} part) appears in the reflection.

The approach of Grosch [12] does not have this limitation, which combines photon mapping and differential rendering to include virtual objects in a photograph. It emits photons onto virtual objects and stores their energy on the final hit position. The resulting differential photon map also contains negative values to mark occluded areas. During the final composition, rays through each pixel of the photograph identify the objects as real or virtual. Color values for real and diffuse surfaces are altered according to the differential photon map entry. For reflective and refractive virtual objects an additional ray-tracing pass determines the final diffuse surface intersection. By projecting this point back onto the photograph, it is possible to apply the differential rendering effect also to reflective and refractive objects. This technique produces convincing results for reflections and refractions from virtual objects, including caustics (Figure 3.10a). However, the involved techniques restrict the approach to off-line rendering and is currently not practicable in an interactive mixed-reality scenario.

A mixed-reality scenario with reflective and refractive objects is presented in the master thesis of Pirk [33]. His method uses a static environment map to approximate reflections. However, classic environment map reflections work only well for distant objects. Generally, such an approximation is obvious for objects near to the reflector, which is a common scenario in a mixed-reality environment. Refractions work either with a static environment map or with the captured camera image. The refraction ray is computed just for the first surface intersection, i.e. when the ray enters the object. Except for the static limitation, the method provides nice results for simple scenes and runs at real-time frame rates, see Figure 3.10b.

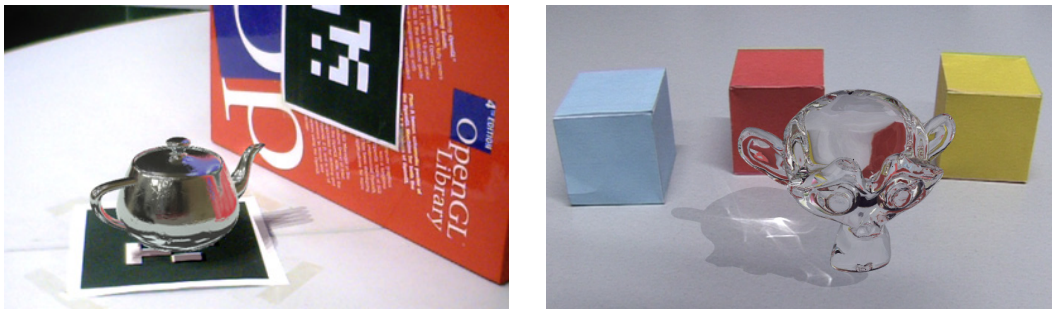
The work of Pessoa et al. [31] also builds upon environment maps to include reflective and refractive objects into a mixed-reality system (Figure 3.11a). They use a static environment map, built from photographs of the application area, to define the entire surrounding scene. Further, manually placed light sources imitate the real light conditions. Every virtual reflective object needs another environment map, which is created once per frame. Subsequent filtering of this



(a) (b)

Figure 3.10: Figure (a) shows an image created with differential photon mapping from Grosch [12] that is able to simulate caustics. Image courtesy of Grosch [12]. Figure (b) shows a refractive dragon in mixed-reality scene generated with the framework of Pirk [33]. Image courtesy of Pirk [33].

map enables indirect light or glossy reflection effects. Virtual refractive objects use the captured camera image directly, instead of an environment map, to determine the refraction contribution. They combine these techniques with a spatial BRDF, Fresnel reflectance and differential rendering to simulate various light effects. Compared to our approach, a few differences emerge. We capture the surrounding scene dynamically with a fish-eye camera and determine the real light condition automatically from the resulting environment map. The framework from Pessoa et al. handles reflections and refractions only from virtual objects and the differential rendering effect is missing. Moreover, refracted objects have no transmittance color and caustics are not considered. Apart from that our method currently simulates only ideal reflections and refractions whereas the method of Pessoa et al. is also able to approximate glossy reflections. Their method produces convincing results for various lighting situations at interactive frame rates.



(a) (b)

Figure 3.11: Figure (a) shows a mixed-reality scene with a reflective teapot, image courtesy of Pessoa et al. [31]. In Figure (b), the virtual object realistically refracts the surrounding objects and casts a caustic onto the real desk, image courtesy of Kán and Kaufmann [19]

In a recent publication, Kán and Kaufmann [19] describe a rendering system that is based upon GPU ray-tracing to simulate the light distribution in a mixed-reality environment at interactive frame rates. Similar to Knecht et al. [22], they compute the differential rendering solution in only one pass. For this, Kán and Kaufmann proposes an adapted ray-tracing process that computes the radiance with four different types of rays, i.e. a separate light and shadow ray for the L_{rv} and L_r buffer. Moreover, this method creates a photon map to simulate caustics. Final diffuse surface hits of light photons are identified on the GPU and later processed on the CPU to build the spatial data structure for photon mapping. This photon map is then additionally used in the aforementioned ray-tracing process. Their technique simulates various lighting effects in a physical correct and realistic manner, including multiple reflections, refractions and caustics. Compared to our approach, the biggest difference is rendering speed. Instead of ray-tracing we use an image-space approximation for the intersection computation. Of course, this approximation is not as accurate but it produces convincing results at significantly higher frame-rates. Apart from that, their approach does not simulate indirect illumination from diffuse surfaces. Whereas in our method, indirect illumination also appears in reflections and refractions. Nonetheless, the method of Kán and Kaufmann creates stunning light effects from reflective and refractive objects that realistically merge with the real environment, see Figure 3.11b.

CHAPTER 4

Reflections and Refractions with Differential Rendering

The light simulation in the RESHADE framework builds upon differential instant radiosity and is able to realistically simulate the light distribution in a mixed-reality environment. This method uses differential rendering, the fundamental technique to integrate virtual objects into a captured image from a camera.

Differential rendering works well for diffuse objects, however, it is challenging to apply this technique also to reflective or refractive objects. The reason is that the radiance of a visible point on such an object now comes from a reflected or refracted surface. Therefore, it is important for a plausible augmentation to apply the differential rendering effect also to reflective and refractive objects, so that they seamlessly merge with a real scene (like in Figure 4.1).

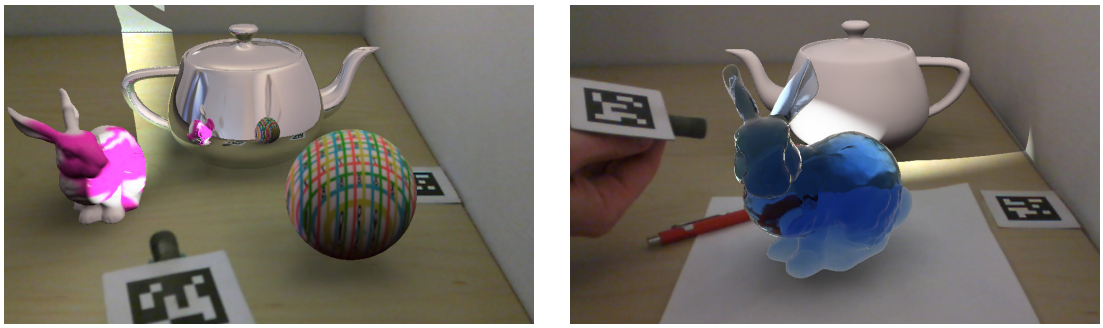


Figure 4.1: Reflections and refractions in a mixed reality environment. In Figure (a), the differential rendering effect is also applied to the real desk in the reflection of the virtual teapot. In Figure 4.1b, the virtual Stanford bunny refracts real and virtual objects.

Two questions arise: how do we compute the radiance for a reflected or refracted surface and how do we combine the computed radiance values with the captured camera image. This chapter

describes the radiance computation for reflective and refractive objects and further explains the concept behind their integration in a mixed reality environment with differential rendering.

4.1 Computing the Radiance of Reflected and Refracted Light Paths

Recall that radiance has an important property in a setting without participating media, as in our mixed-reality scenario: The amount of radiance that leaves a surface in a specific direction is equal to the amount of radiance that arrives at a surface from this direction (see Section 2.1). According to this invariance of radiance along a ray, our lighting simulation simply computes the radiance for every visible reflected and refracted surface point.

Initially, our method identifies the two visible surfaces from the reflected and the refracted view direction (p_{refl} and p_{refr} in Figure 4.2). This intersection computation is based on an image-space approximation, which is presented later in Chapter 6. Hence, for the following section, we assume that this intersection data are already available.

The incoming radiance is collected from all primary light sources and from all virtual point lights, which simulate the indirect illumination. This computation evaluates the BRDF at the reflected and refracted surface points p_{refl} and p_{refr} with all light sources in the scene. All input parameters are known from the intersection approximation. These include the material attributes, the surface normal and the two directions, which are the incoming direction from the light source and the outgoing direction along the reflection or refraction ray. Moreover, at this stage, we test for an occlusion between the surface point and the corresponding light source to handle shadows.

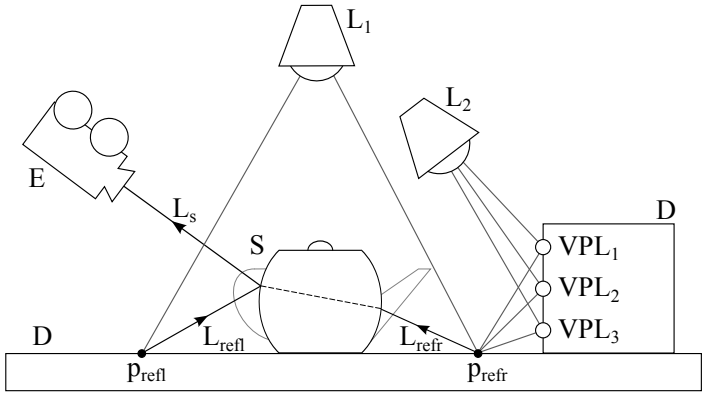


Figure 4.2: Illustrates the radiance computation for the reflected and refracted surface points p_{refl} and respectively p_{refr}. The incoming radiance L_s towards the observer E is a linear combination between L_{refl} and L_{refr}, weighted by the reflectance characteristics of object S.

Figure 4.2 outlines this gathering. First, the view ray hits the refractive teapot S and is split into a reflective and refractive part. The intersection approximation traces these two rays and

finds the reflected and refracted diffuse surface points p_{refl} and p_{refr}. Afterwards, the lighting stage evaluates the BRDF with every possible light source and accumulates the incoming radiance.

As a result, we get two different radiance solutions, one for the reflected surface L_{refl} and one for the refracted surface L_{refr}. Note that this exitant radiance is along the reversed reflected and refracted ray direction, indicated by the arrows in Figure 4.2. Both originate at the intersection point between the view ray and the reflective and refractive object. Consequently, the incoming radiance L_s for an observer along its view direction is simply a linear combination of L_{refl} and L_{refr}. The coefficients in this linear combination depend on the reflection and refraction characteristics of the visible object. Their determination follows in the next section. Note that the following example images are already presented in a mixed-reality context, although this integration into a real scene is discussed later in Section 4.2.

4.1.1 Fresnel Equations

Ideal specular reflections perfectly mirror the incoming radiance towards the reflected ray direction. The corresponding BRDF consists of a *Dirac delta function* $\delta(\theta_i - \theta_o)$, which evaluates to zero for every possible direction except the reflection direction [9]. The angles θ_i and θ_o are between the surface normal and the incoming or outgoing ray direction, compare with Figure 2.5. However, this BRDF model only applies to objects that entirely reflect or refract the incoming radiance.

A more realistic model should involve the *Fresnel equations*, which are based on physical assumptions. The Fresnel equations define the amount of reflected and refracted light from a perfectly flat surface, flat in terms of the geometric optics light model. For a specified set of material attributes, this amount only depends on the incident angle θ_i. We use the approximation from Schlick [38] to evaluate the complex Fresnel equations, which is a de-facto standard in real-time rendering. Schlick observes that for grazing angles (θ_i goes towards $90°$) the reflectance quickly converges to one. This results in the following equation

$$f_r(\theta_i) = f_\perp + (1 - f_\perp)(1 - \theta_i)^5, \tag{4.1}$$

where $f_\perp$ is the Fresnel factor at normal incidence. Due to the conservation of energy, the refracted part is $1 - f_r(\theta_i)$. Thus, radiance is either reflected or refracted but no portion gets absorbed.

We manually specify a Fresnel factor $f_\perp$ as a material attribute for each reflected and refracted object. Figure 4.3 shows the influence of the Fresnel factor $f_\perp$ on the reflected radiance. At grazing angles, noticeable on the border of the teapot in the right image, the reflection appears brighter. Alternatively, we allow to directly compute this value from the refraction index:

$$f_\perp = \frac{(\eta - 1)^2 + k^2}{(\eta + 1)^2 + k^2}, \tag{4.2}$$

where η is the refraction index and k is the extinction coefficient, required for metallic objects [44]. Note that η is a ratio between the refraction indexes of the two involved materials. However, in our scenario the second surface is always air, which has a refraction index of one, so η is simply the refraction index of the object.

(a) (b)

Figure 4.3: Image (a) shows an ideal reflection. Image (b) shows the influence of the Fresnel equation on the reflected radiance.

4.1.2 Transmittance

Refractive objects have another interesting characteristic. They are translucent, so a certain amount of light is absorbed when it goes through the object. A color filter imitates this absorption process for colored objects. This means that the refracted radiance L_{refr} is multiplied with a color filter to simulate objects with a colored translucent material.

(a) (b)

Figure 4.4: A refractive object with a translucent material color. Image (a) uses a color filter with a default thickness. Whereas the image (b) shows an absorption that varies with the thickness of the object.

Our approach uses a color filter t_r that varies with the thickness of the object as defined by the Beer-Lambert law [1]:

$$t_r = e^{-\alpha' cd}, \tag{4.3}$$

where α' is the absorption coefficient, c is the concentration and d is the thickness of the object, i.e. the distance the light travels through the object. The coefficient α' is the minimal absorption

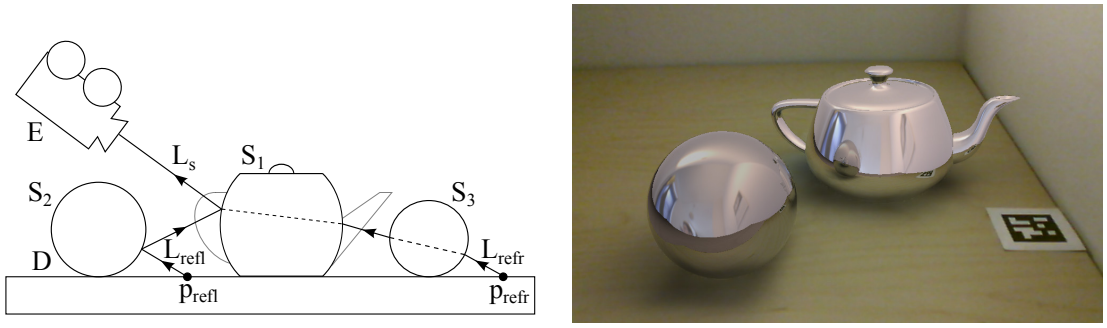
at a specified thickness and computed as:

$$\alpha' = \frac{-\log(t_{rgb})}{d_{max}}, \tag{4.4}$$

where t_{rgb} is the translucent color weighted by the thickness d_{max}. In our system, a refractive material stores the required parameters for the computation of the color filter t_r. Note that the thickness d varies with the view orientation and is computed by the intersection approximation. On the other hand, our system also supports a filter with a default depth, which results in an equal absorption of light, see Figure 4.4 for a comparison.

4.1.3 Multiple Reflections and Refractions

Up to this point, the described radiance computation only handles one reflective or refractive object. To express it in light bounces, this means that the light hits a reflective or refractive object once, changes its direction and arrives at a diffuse surface, i.e. the paths $LSDE$ or $LSSDE$. In case of multiple reflective or refractive objects in a scene, the light bounces possibly multiple times before it finally hits a diffuse surface. Figure 4.5a illustrates multiple bounces on reflective and refractive objects.



(a) (b)

Figure 4.5: Multiple bounces of the light on reflective and refractive objects. The radiance is attenuated at every surface interaction, see the reflected sphere in the image (b).

The outgoing radiance of the diffuse surface is attenuated with every bounce on a reflective or refractive object. Therefore, our system computes the Fresnel factor at every surface interaction from the incoming and outgoing radiance direction, whereas the intersection approximation provides the surface normal and material attributes. All these factors are multiplied together and result in a Fresnel factor f_r for the hole path:

$$f_r = f_{r_1} \cdot f_{r_2} \cdot \cdots \cdot f_{r_n}, \tag{4.5}$$

where f_{r_i} are the Fresnel factors at each light bounce. Figure 4.5b visualizes this attenuation, where the reflective sphere appears darker on the reflective teapot.

The same principle holds for second-order rays that interact with colored refractive objects. Our approach stores the absorption color filter t_r for the hole path:

$$t_r = t_{r_1} \cdot t_{r_2} \cdot \cdots \cdot t_{r_n}, \tag{4.6}$$

where t_{r_i} is the color filter at each light bounce on a refractive object. Contrary to a single reflection, the reflected radiance L_{refl} may also be affected by an absorption color filter, which happens when the reflected view ray hits a colored refractive object. Accordingly, our method stores a separate transmittance color filter for the reflective and refractive radiance part.

Unfortunately, this approach has some limitations when a second-order ray hits a refractive object. Ideally, this ray is split into a reflective and refractive part and traced independently. However, our intersection approximation only handles one path for the reflected view direction and one path for the refracted view direction, i.e. it stores the intersection data of only one hit diffuse surface per path. Under these circumstances, our approach is to exclusively follow the refracted ray direction when a higher order bounce happens on a refractive object. We observed that this approximation works reasonably well in most scenarios.

4.1.4 Composition of the Radiance Equation

We now return to the composition of the incoming radiance along the view direction. The incoming radiance L_s is a linear combination of the outgoing radiance from the reflected and refracted diffuse surface:

$$L_s = f_{r_{refl}} \cdot t_{r_{refl}} \cdot L_{refl} + f_{r_{refr}} \cdot t_{r_{refr}} \cdot L_{refr}, \tag{4.7}$$

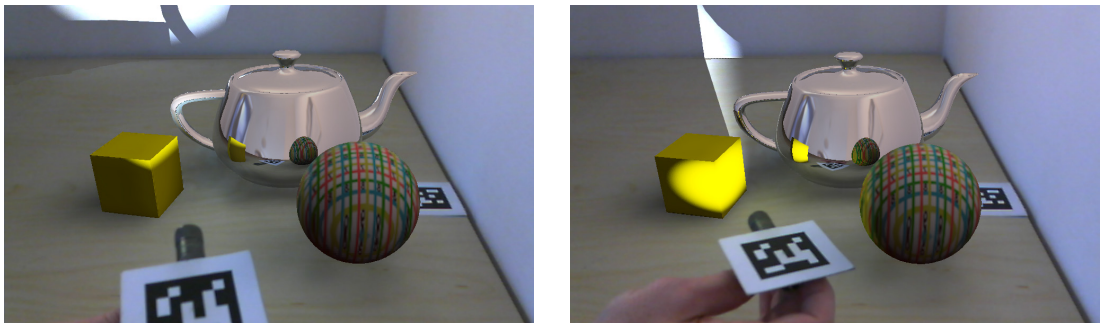
where L_{refl} and L_{refr} are the outgoing radiances of the reflected and refracted surfaces. The weighting coefficients of the linear combination depend on the Fresnel factors ($f_{r_{refl}}$, $f_{r_{refr}}$) and the transmittance color filters ($t_{r_{refl}}$, $t_{r_{refr}}$) along the path of the reflected and refracted view direction. Figure 4.6 shows the outgoing radiance from a reflective teapot for two illumination scenarios. Note the color bleeding on the reflected sphere in the teapot (Figure 4.6b).

Furthermore, we use an additional parameter to simulate slightly reflective materials that incorporates the outgoing diffuse radiance of the reflective or refractive object. The diffuse radiance part L_d is computed by the differential instant radiosity method as described in the previous Section 2.4. To enhance flexibility, reflective and refractive objects also store diffuse material attributes. A reflectivity parameter controls the additional amount of diffuse radiance:

$$L = (1 - k_r) \cdot L_d + k_r \cdot L_s, \tag{4.8}$$

where k_r is the reflectance parameter, L_d is the diffuse radiance, and L_s is the radiance from the reflection and refraction. To hold the law of the conservation of energy, the parameter k_r always is less or equal to one. Generally, this parameter is explicitly set to one or to zero, which is a diffuse surface in the first case or an ideal reflective or refractive surface in the second case. The teapot in Figure 4.6a has a reflective parameter k_r slightly smaller than one to simulate the direct illumination from the spotlight, see the handle of the teapot.

Table 4.1 summarizes the possible light paths that are handled by our radiance computation. Reflective and refractive objects react to direct and indirect illumination, taking into account



(a) (b)

Figure 4.6: Images show the outgoing radiance for a reflective teapot from a direct and indirect illumination. The bright handle of the teapot (Image (a)) results from a reflectivity parameter slightly smaller than one. Note the color bleeding on the reflected sphere in the teapot (Image (b)).

multiple bounces on reflective or refractive objects. Note that these multiple bounces are listed explicitly with the term $S'*$ in the light path because they are subjected to some limitations. We already mentioned one limitation that restricts the radiance computation to only one diffuse surface per ray path. The other limitations are related to the image-space intersection approximation and are discussed further in Chapter 6.

Nr	Path	Description
1	LSE	Direct illumination from a light source.
2	LDS'*SE	Direct illumination of diffuse surfaces appearing in reflections.
3	LDS'*SSE	Direct illumination of diffuse surfaces appearing in refractions.
4	LDDS'*SE	Indirect illumination of diffuse surfaces appearing in reflections.
5	LDDS'*SSE	Indirect illumination of diffuse surfaces appearing in refractions.

Table 4.1: Possible light paths in the radiance computation. The term $S'*$ in the light path describes multiple light bounces.

We have seen how to compute the incoming radiance for reflective and refractive objects. Most of the presented example images already show reflective or refractive objects in the context of a mixed-reality environment. The following section describes the integration of this radiance computation in a scenario that requires a plausible simulation of the light interaction between real and virtual objects.

4.2 Differential Rendering

Differential rendering is essential for a smooth integration of virtual objects into a mixed-reality environment but difficult to apply for reflective and refractive objects. A simple approach would be to ignore the differential rendering technique for these kind of objects as done by Debevec [8]. However, the estimated material becomes immediately noticeable to an observer and reduces the visual quality of the augmented scene. Consequently, it is preferable to use differential rendering also for reflective and refractive objects, which improves the overall impression of the mixed-reality environment, compare the appearance of the desk in the reflection of the teapot in Figure 4.7. The following section explains the application of differential rendering for reflective and refractive objects, which includes the radiance contribution to the real L_r and real-virtual buffer L_{rv}, and the handling of the buffer L_b from the captured camera image.

(a) without differential rendering (b) with differential rendering

Figure 4.7: Figure (a) shows a reflective teapot, rendered without differential rendering. Hence, the estimated material attributes of the real desk are quickly noticeable in the reflection of the teapot. Otherwise, Figure (b) shows that differential rendering improves the overall impression of this scene.

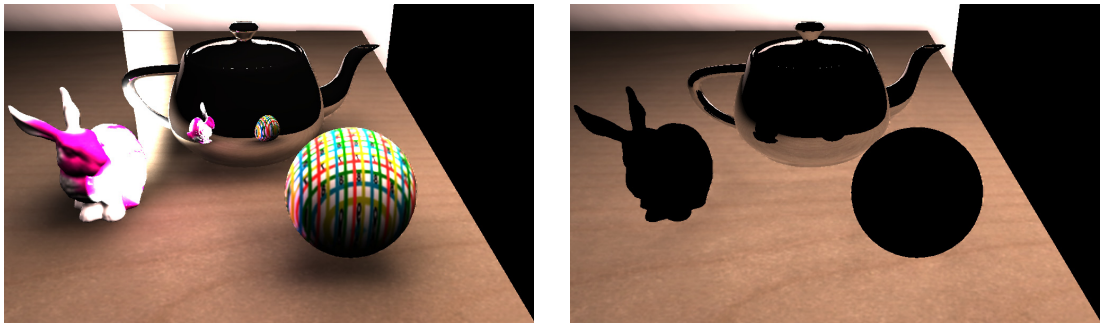
4.2.1 Real and Real-Virtual Radiance Buffer

Linking the incoming radiance from the reflected and the refracted surface with differential rendering requires an identification of all contributing components, whether they are real or virtual. As in differential instant radiosity, we analyze the light path for every visible surface. Depending on the real-virtual flags of the included components in the light path, the outgoing radiance from a surface contributes either to both buffers L_r and L_{rv} or solely to the buffer L_{rv}.

As mentioned before, the radiance of a visible point on a reflective or refractive object refers to the outgoing radiance from the corresponding reflected and refracted surfaces. This means that the real-virtual flag of the reflective or refractive object has no influence on the chosen differential buffer, only the real-virtual flag of the reflected or refracted surface is crucial.

Consider the reflective virtual teapot in Figure 4.8 that shows the two differential rendering buffers L_{rv} and L_r from Figure 4.7. The virtual teapot reflects the real desk that is illuminated by the real VPLs from the environment. Therefore, the outgoing radiance contributes to both

buffers. Otherwise, the illumination of the virtual spotlight, the radiance from the virtual bunny, and the radiance from the virtual sphere only affect the buffer L_{rv}. Note that the area on the reflective teapot refers to the outgoing radiance of these reflected surfaces.



(a) L_{rv} (b) L_r

Figure 4.8: This images depicts the two radiance buffers L_{rv} and L_r of the differential rendering method for the final image L_f in Figure 4.7. The area on the reflective teapot corresponds to the outgoing radiance of the reflected surfaces, which is the bunny, the desk, and the sphere for L_{rv} and just the desk for L_r.

This means that we need to classify the involved components for every visible reflected and refracted surface. Note that this analysis already includes multiple bounces on reflective and refractive objects that occur between the observer and the first visible diffuse surface, i.e. the reflected or refracted surface. Moreover, it is identical for a reflected and a refracted surface. The outgoing radiance contribution from a reflected surface to the differential rendering buffer depends on the components in the path from the light source to the reflected surface. The other part of the light path, which goes from the reflected surface to the observer, has no influence on the decision about the chosen buffer. Even if this portion contains multiple bounces on reflective objects, no matter if on real or virtual objects, important is only the real-virtual flag of the reflected point on a diffuse surface and the corresponding path of its incoming radiance. Accordingly, the outgoing radiance affects the buffer L_{rv} when the path from the light source to the reflected object contains a virtual component. Otherwise, if all components are real, the radiance contributes to both buffers L_r and L_{rv}.

Figure 4.9a shows two different light paths, in which the first is $L_v D_v S_v E$ and the second is $L_v D_v D_r S_v E$. Both paths contribute only to the buffer L_{rv}, because the first part of the path (dashed line) contains virtual components. The same applies to the third path $L_r D_r D_v S_v E$ in Figure 4.9b: although the light source is real, the radiance only goes to the buffer L_{rv} because the last two bounces are on the virtual box and the virtual teapot. On the other hand, the fourth path $L_r D_r S_v E$ affects both buffers L_r and L_{rv}, because the reflected surface interacts only with real components. During the radiance computation, our method tests for an occlusion between the surface point and the light source, illustrated by the dotted line in the left image. Only real shadow casters have an influence on the L_r buffer, as described by Knecht et al. [22].

To incorporate differential rendering with the radiance computation, we define a helper func-

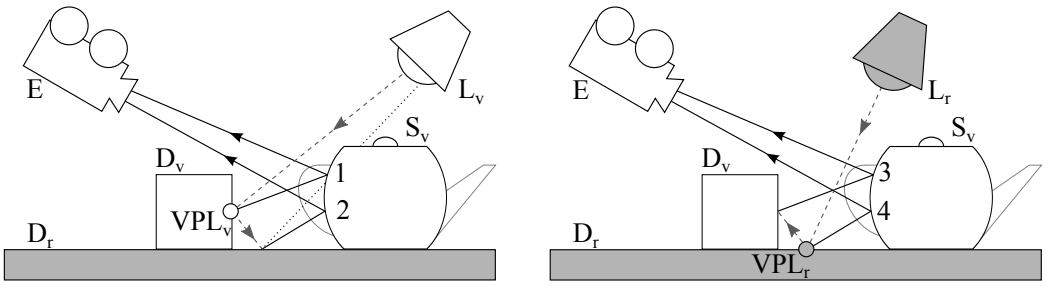


Figure 4.9: The labeling in the light path helps to determine the corresponding differential rendering buffer. Paths one, two, and three affect only buffer L_{rv}. The reflected radiance in path four goes to both buffers L_r and L_{rv}.

tion $r(x)$ that identifies a real component in a given light path:

$$r(x) = \begin{cases} true & \text{if x is real} \\ false & \text{otherwise} \end{cases}, \tag{4.9}$$

where the argument x is a light source, a virtual point light, or a point on a surface. Next, the radiance equation that computes the incoming radiance along the view direction (defined with Equation 4.7) directly addresses the light sources and is rewritten as:

$$L'_s = \sum_{ls \in lights} f_{r_{refl}} \cdot t_{r_{refl}} \cdot L'_{refl}(ls) + f_{r_{refr}} \cdot t_{r_{refr}} \cdot L'_{refr}(ls) \tag{4.10}$$

$$L'_{refl}(ls) = \begin{cases} L_{refl}(ls) & \text{if} \quad r(ls) \wedge r(p_{refl}) \\ 0 & \text{otherwise} \end{cases} \tag{4.11}$$

$$L'_{refr}(ls) = \begin{cases} L_{refr}(ls) & \text{if} \quad r(ls) \wedge r(p_{refr}) \\ 0 & \text{otherwise} \end{cases}, \tag{4.12}$$

where $lights$ is the set of all direct light sources and virtual point lights. The points p_{refl} and p_{refr} represent the first visible point on a diffuse surface from the reflected or refracted view direction. Note that the reflected or refracted ray might bounce multiple times on reflective or refractive objects before it finally reaches a visible diffuse surface point, i.e. the reflected or refracted surface location. Consider, the buffers with superscript account for the correct differential rendering buffer. This equation computes the incoming radiance L'_s from the outgoing radiance of a reflected and a refracted surface and accounts for the differential rendering effect. Finally, the two differential rendering buffers are:

$$L_{rv} = (1 - k_r) \cdot L_d + k_r \cdot L_s \tag{4.13}$$

$$L_r = (1 - k_r) \cdot L'_d + k_r \cdot L'_s, \tag{4.14}$$

where k_r is the reflectivity parameter, L_d is the amount of diffuse radiance, and L_s is the incoming radiance from the reflected and refracted surface. The diffuse output L'_d from differential rendering already accounts for this effect (see Chapter 2.4) and reflective or refractive objects use the aforementioned computation of L'_s.

Some parts of the differential rendering equation have not been mentioned. Virtual objects, or in our case a virtual reflected or refracted surface, need to mask out the occupied area of the virtual surface in the background image L_b. Only the L_{rv} buffer is used for them, because the background is covered by the virtual surface and becomes invalid. Aside from that, the captured area from the camera image of a reflective or a refractive object usually has no relation to the corresponding area in the differential buffers, i.e. the reflected or refracted surface maps to a different location in the camera image. Hence, this means that the buffer L_b needs a special handling. The following section describes this in more detail and finally presents the composition of the output image.

4.2.2 Composition of the Output Image

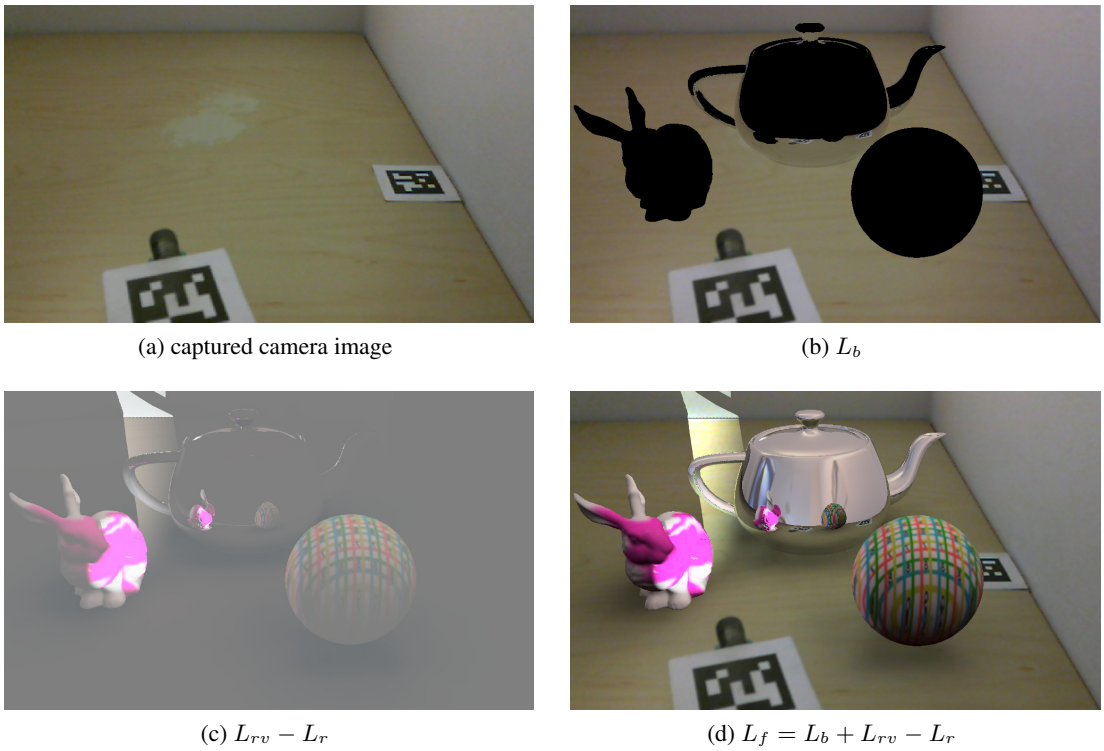
Inserting a virtual object, like the teapot in Figure 4.7, invalidates the values in the captured camera image. Hence, the screen space color value from the buffer L_b has no relation to the screen space color value, with the same xy-coordinates, of the differential rendering buffers L_r or L_{rv}. Usually, this area is masked out for virtual diffuse objects, but it may refer to a real surface in case of a reflective or refractive object.

The original differential rendering technique uses only the buffer L_{rv} in such an event, neglecting the differential rendering effect for reflective objects. For instance, the real desk would appear with the approximated radiance L_{rv} in the reflection of the teapot, which reduces the overall impression of the scene (see Figure 4.7a). To improve the visual appearance, our approach also applies differential rendering to reflected and refracted real surfaces.

Therefore it is important to identify the corresponding value in the captured image of the camera for a real reflected or a refracted surface. This requires a back-projection of these surface positions into the camera image, similar to the method of Grosch [12]. The value at the position of the back-projected point in the camera image adjusts the original buffer L_b. However, our method also supports real reflective or refractive objects. In such cases, the original value in the camera image might be appropriate when it reflects and refracts a real surface or becomes invalid for a virtual surface.

Figure 4.10 illustrates this concept for the virtual reflective teapot. The reflected real surface points, which are visible on the virtual teapot, are back-projected into the camera image, see Figure 4.10a. The corresponding value adjusts the masked background L_b, so the real desk appears on the reflective teapot, as in Figure 4.10b. Differential rendering adds the difference between the two radiance solutions $L_{rv} - L_r$ to this masked and adjusted background image L_b. Note that gray means no difference between the computed values in L_{rv} and L_r in Figure 4.10c.

Three different scenarios arise for the usage of the buffer L_b. The value is masked out, the original value remains valid, or comes from the back-projection into the camera image. To distinguish these cases, we need to classify the components in the light path, whether they are real or virtual. Contrary to the classification of the real and virtual radiance buffer, we now use the second part of the light path that goes from the reflected or refracted surface to the observer.



(a) captured camera image

(b) L_b

(c) $L_{rv} - L_r$

(d) $L_f = L_b + L_{rv} - L_r$

Figure 4.10: Differential rendering adds the difference $L_{rv} - L_r$ to the masked background L_b. Usually, virtual objects are masked out in the buffer L_b, except for virtual reflective or refractive objects. These areas are adjusted with the values from the back-projection into the captured camera image.

Figure 4.11 illustrates these cases. The first light path is LD_vS_rE, in which the second part of the path (dashed line) contains a virtual object. Hence the value of L_b is masked out. The color value of L_b is valid in the second path LD_rS_rE, because the real teapot reflects a real object. On the other hand, the value of L_b in the right image needs a back-projection, because the reflected path three $LD_rS_vS_rE$ contains a virtual component. Viewed in isolation, the corresponding refracted path four $LD_rS_rS_rE$ would map to the original value of L_b. However, the original value of L_b is only valid when both, the reflected and refracted path portions, contain real components.

Let us briefly consider again the decision on the correct radiance buffer from the last section in case of multiple bounces on reflective objects, as with light path three in Figure 4.11b. Regardless if the multiple bounces occur on real or virtual reflective objects between the observer and the visible reflected surface: only the portion from the light source to the reflected point (on the diffuse surface D_r) is used to determine the correct differential-rendering buffer L_r or L_{rv}. The Sections A.1 and A.2 in the Appendix list all possible cases and describe the derivation of L_b, including the composition of the final output image L_f, in more detail.

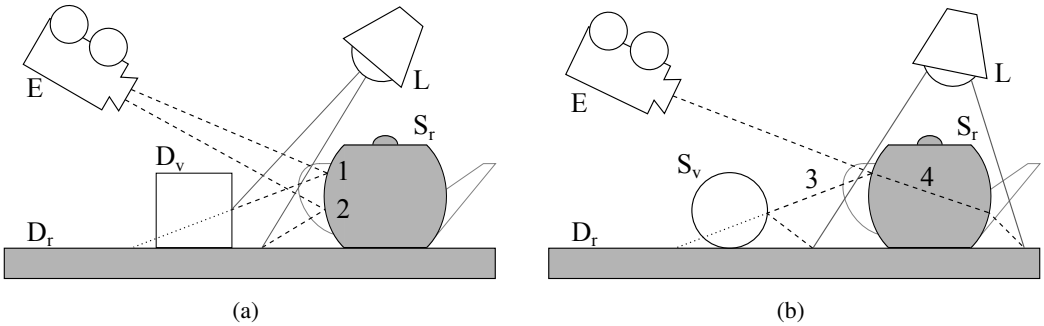


Figure 4.11: Light path classification for the back-projection. L_b is masked out for path one. L_b remains the original value for path two. Path three and four must be considered in conjunction, resulting in a back-projection to determine L_b.

To summarize, the original value of the camera image is valid when the reflected and refracted surfaces are real and the path from the observer to these surfaces contains only real objects. In all other cases, we need a back-projection into the camera image or need to mask the corresponding area in the image.

Before we show the composition of L_b, we define the following four helper functions:

1. Function $r(x)$ maps a real surface to true and a virtual to false, see Equation 4.9.

2. Function $r_p(x, y)$ evaluates the portion of the light path between two surface points x and y. It returns true when all components, including the start and end points, are real or false if one is virtual:

$$r_p(x,y) = \begin{cases} true & \text{if} \quad \forall c \in x \to y : r(c) \\ false & \text{otherwise} \end{cases} \tag{4.15}$$

3. Function $L_b(x)$ takes a surface point, projects it into the camera space and returns the corresponding color value from the captured camera image.

4. Function $L'_b(x)$ determines this color value only for a real surface:

$$L'_b(x) = \begin{cases} L_b(x) & \text{if} \quad r(x) \\ 0 & \text{otherwise} \end{cases} \tag{4.16}$$

Unfortunately, the intersection approximation and the back-projection function are not always able to return a valid result because back-projected points could be outside of the view frustum of the camera. However, for the following part we assume correct return values, the missing cases are discussed further in the next two parts, see Section 4.2.3 and Section 4.2.4.

The back-projection function is specified as:

$$L'_{bp}(p, p_{refl}, p_{refr}) = \begin{cases} L_b(p) & \text{if} \quad r_p(p, p_{refl}) \wedge r_p(p, p_{refr}) \\ f_{r_{refl}} \cdot t_{refl} \cdot L'_b(p_{refl}) \\ f_{r_{refr}} \cdot t_{refr} \cdot L'_b(p_{refr}) & + \quad \text{otherwise} \end{cases} \tag{4.17}$$

It returns the original value from the camera image, $L_b(p)$, when all involved objects are real. Otherwise, it returns the corresponding color value in the camera image for the back-projected reflected, $L'_b(p_{refl})$, or refracted, $L'_b(p_{refr})$, location. Note that these two color values are attenuated by the Fresnel factors and the transmittance color filters.

Note that the radiance computation happens in high dynamic range, so a tone mapper brings the two radiance parts, L_r and L_{rv}, into low dynamic range before their difference is added to the background L_b. Moreover, we introduce an additional scaling parameter α for the differential part that further reduces the approximation error from the real radiance buffer L_r. This parameter α is the ratio between the background value L_b and the real radiance part L_r. Essentially, this factor is equal to the alternate formulation from Debevec [8] that accounts for the relative error in the differential rendering effect.

The differential rendering equation accounts for the correct background value with the following adaption:

$$L_b = (1 - k_r) \cdot L'_b(p) + k_r \cdot L'_{bp}(p, p_{refl}, p_{refr}) \tag{4.18}$$

$$\Delta L = L_{rv} - L_r \tag{4.19}$$

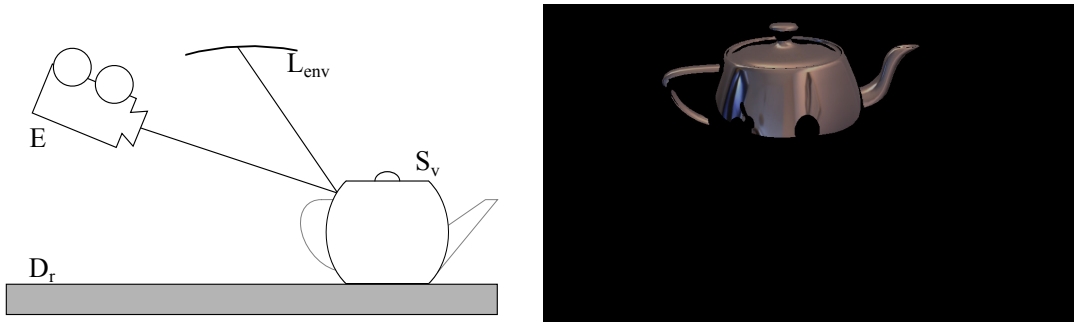
$$L_f = L_b + L'_{env} + \alpha \cdot \Delta L, \tag{4.20}$$

where argument p is the visible surface point on the object, either on a diffuse object or on a reflective or refractive object. The arguments p_{refl} and p_{refr} are the reflected and refracted diffuse surface positions. Function $L'_{bp}(p, p_{refl}, p_{refr})$ is the back-projection function for reflective and refractive objects. Frequently, the reflection or refraction ray misses all real and virtual objects, which is handled by the term L'_{env}, see the next Section 4.2.3. Note that this formulation of the differential rendering equation already accounts for diffuse objects (k_r is zero) and the correct masking of their background value.

As stated before, the back-projection function is not always able to deliver a correct color value. Furthermore, the differential rendering equation needs to deal with missing intersection data that is approximated with the term L_{env}. The next section discusses these cases and addresses the two limitations in more detail.

4.2.3 Missing Intersection Data

In the previous section, it was assumed that all necessary intersection data are available to compute the outgoing radiance for a reflected and a refracted surface. However, not all reflected or refracted rays intersect some geometry of a real or a virtual object. Actually, the probability that these rays miss all objects is high, since a mixed-reality environment is only populated with a limited amount of pre-modeled real objects. In such an event, there is no surface information to carry out the radiance computation, see Figure 4.12a for an illustration.



(a) (b)

Figure 4.12: Missing intersection data are approximated by an environment map. Image (b) shows the L_{env} buffer of the rendering in Figure 4.7, note the attenuation from the Fresnel factor.

Therefore, our method approximates the missing geometry with an environment map. This map contains the captured environment from a real-time video feed of a fish-eye camera, updated every frame. Similar to sphere mapping, we compute look-up coordinates from the reflected or refracted ray and store this radiance value in a separate buffer L_{env}. This radiance is attenuated with the Fresnel factors and multiplied with the transmittance color filter. Figure 4.12b shows the approximated radiance from the environment map for the final output image of Figure 4.7. Also note in this image that the buffers L_r and L_{rv} remain unaffected in these areas of the teapot because the computed radiance L_s is empty for the reflected and refracted surface.

Generally, only virtual reflective or refractive objects require the information from the L_{env} buffer, because the captured pixel for a real object in the camera image already maps to the correct reflected or refracted environment value:

$$L'_{env} = \begin{cases} 0 & \text{if} \quad r_p(p, p_{refl}) \wedge r_p(p, p_{refr}) \\ L_{env} & \text{otherwise} \end{cases} \tag{4.21}$$

Note that it is always preferable to directly use the information from the camera image, because it accurately represents the captured scene. In certain cases, when the reflective or refractive surface is real, it is possible to use the original value in the camera image although the reflected or refracted ray hits the environment map. Therefore, our approach classifies an environment map hit as real, i.e. the helper function $r_p(x, y)$ treats them as real components.

4.2.4 Missing Back-Projection Data

Reflections and refractions are not limited to the visible area of the camera. These rays may intersect geometry outside the field of view on the one hand or are occluded by some other geometry inside the camera frustum on the other hand. For this reason, our method also needs to deal with missing data during the back-projection. Figure 4.13 illustrates this two cases.

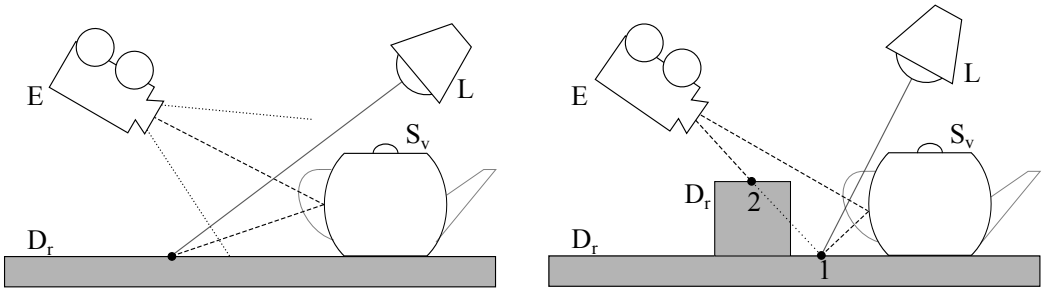


Figure 4.13: Missing data in the back-projection. Figure (a) shows a reflected surface outside the field of view (dotted line). Figure (b) shows an occluded real geometry that lies inside the camera frustum.

If the reflected or refracted surface lies outside the field of view, our method simply uses the radiance from the L_{rv} buffer and thus ignores the differential rendering effect.

In case of occluding geometry, the back-projection would return a wrong color information from the camera image, which has no relation to the reflected or refracted surface. Figure 4.13b illustrates this case, in which the real reflected surface at point 1 lies inside the camera frustum. However, the back-projection would falsely return the color value from point 2.

To identify such a problem, our approach compares the depth value between the visible point and the back-projected point. When these two depths lie within an epsilon range, the back-projected color value is suitable, otherwise the L_{rv} buffer is used, as before. In Figure 4.14a, the green top face of the box wrongly appears in the reflection whereas a depth comparison correctly handles this missing case, as shown in Figure 4.14b. Note that we only use the depth information from real objects, i.e. we exclusively render the real objects in a depth only pass and store their distance.

For a refractive object, the result of the back-projection may be valid for the reflected and the refracted surface or may fail for one of them. Ideally, we would use the back-projected value for the valid part and the buffer L_{rv} for the missing case. Unfortunately, we do not have access to the separate reflective or refractive radiance part for the differential rendering buffers L_r and L_{rv}. Currently, in our approach, L_r and L_{rv} contains the radiance from both surfaces. Hence, we tested two different methods to handle the missing cases for refractive objects, where Figure 4.15 visualizes the results.

If one back-projection fails, either of the reflected or the refracted surface, we use the value from L_{rv} and ignore the differential rendering effect, see Figure 4.15a. The other approach

(a) (b)

Figure 4.14: The green top face incorrectly appears in the reflection of the teapot (Image (a)). A depth comparison detects these errors and therefore uses the data from the buffer L_{rv} (Image (b)). Also note the missing back-projection data in the field of view at the border of the desk appearing in the reflection.

(a) (b)

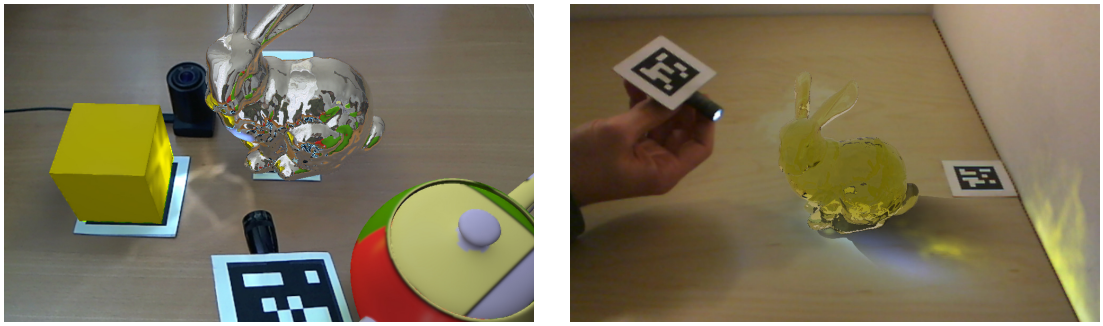
Figure 4.15: In Figure (a) we use L_{rv} when one back-projection fails. In Figure (b) we always use the refraction part when it is valid.

simply ignores the reflective part if one back-projection fails, but always tries to use the refracted part, see Figure 4.15b. From a formal point of view, the first method is more appropriate, however, the second result creates less artifacts.

CHAPTER 5

Caustics with Differential Rendering

Until recently, caustics were just ignored in the lighting simulation of other mixed-reality systems. However, caustics are tightly connected to reflective and refractive objects and improve the overall impression of a mixed-reality scene, see Figure 5.1. The absence of such effects may disrupt the perception of the user, who is familiar with real light conditions and also expects a similar behavior for a mixed-reality environment.



(a) Reflective caustic (b) Refractive caustic

Figure 5.1: Caustics due to the scattering of light from reflective and refractive objects. In Figure (a), a virtual spotlight illuminates the virtual bunny, which causes a caustic on the real desk. In Figure (b), a virtual object is illuminated by a real spotlight, resulting in a virtual caustic on the desk and the wall.

Caustics occur due to a higher concentration of photons in some areas on a diffuse surface. The light path of the scattered photons is complementary to the already presented reflective or refractive light path. In case of caustics, the light interaction on a reflective or a refractive surface is located in the first portion of the path. Whereas for reflections and refractions, this interaction happens in the last part. Consider, for instance in Figure 5.1a, a scene with a reflective object illuminated by a spotlight: The corresponding photon path is $LSDE$, in which the reflected

photon arrives at a visible diffuse surface (e.g. the real desk), and the complementary reflective light path is $LDSE$, which represents the reflected radiance from a diffuse surface (e.g. the real desk in the reflection of the bunny).

Our approach simulates caustics from reflective and refractive objects, either real or virtual. Instead of a physically correct light distribution, our aim is to create perceptually plausible caustics that appear on real or virtual diffuse surfaces. The following section describes this method and its incorporation with differential rendering.

5.1 Radiance Computation

We simulate caustics with the approach from Wyman [49], which imitates the classic photon mapping algorithm but achieves higher frame rates. The classic algorithm emits photons from the light source towards the reflective or refractive object and traces the scattered photons until they land on a diffuse surface. This location and the incoming flux are stored in a spatial data structure, i.e. the photon map. Unfortunately, this spatial data structure (kd-tree) is not directly usable on todays GPU architecture. Hence, we utilize the idea from Wyman and store the final hit location with the attributes of the arriving photon in a two dimensional texture buffer that acts as the photon map. Instead of a ray-tracer, as in the original photon mapping algorithm, we use an image-space intersection approximation to detect the final hit position.

Conceptually, the photon arrives at an infinitesimally small area that results from the intersection of the ray and the diffuse surface. However, the photon energy also spreads over to the neighboring area. For this reason, Wyman introduces an additional caustic buffer that contains the filtered photon energy from the surrounding area. Our method is based upon this concept but differs in the generation and application of the caustic buffer. The creation of the photon buffer and the caustic buffer is described further in Section 6.6.

Figure 5.2 illustrates the caustic simulation. First it emits the photons from the light source and stores their diffuse hit location p_{hit}, see Figure 5.2a. Then it gathers the photon intensities of the neighboring areas and applies this radiance L_c to the corresponding diffuse surface, see Figure 5.2b.

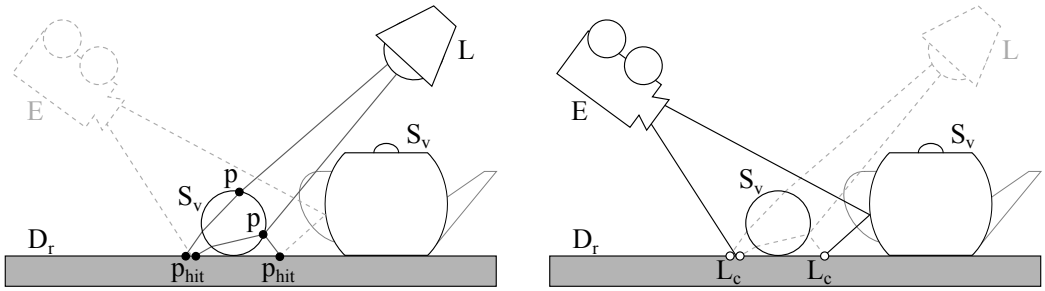
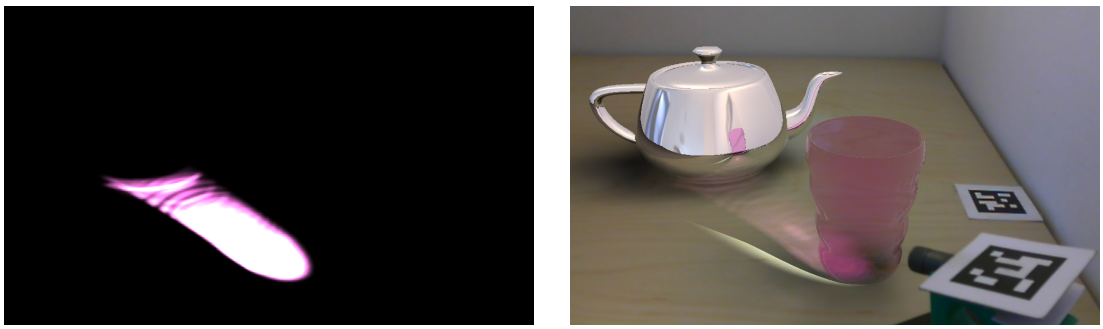


Figure 5.2: Photon emission from the light source (Image (a)). Applying the additional radiance L_c to the diffuse surfaces (Image (b))

For the following part, let us suppose that the caustic buffer is already available. Note that this buffer is separately created for the reflected and refracted photon direction, which originates at the caustic caster. The buffer contains the gathered energy from all incoming photons in a specified neighborhood, represented by the small circles in Figure 5.2b. This incoming radiance is attenuated by the distance to the light source, but also depends on the Fresnel terms and the transmittance color filter, see Figure 5.3a, which shows the intensity from a refractive caustic buffer. Moreover, the caustic buffer contains the incoming direction of the photons.



(a) (b)

Figure 5.3: Refractive caustic from a virtual glass. Image (a) shows the intensity from the caustic buffer in screen space. The caustic affects the diffuse surface and also appears in the reflection and refraction, as shown in Image (b).

To simulate caustics, our method evaluates the BRDF of the diffuse surface with the corresponding attributes in the caustic buffer, i.e. the incoming direction and the intensity. This resulting radiance is added to the diffuse radiance. In addition, caustics should also appear in reflections and refractions. Therefore, we also evaluate the reflected and the refracted diffuse surfaces. This additional radiance is added at the following three places:

$$L_d = L_d + L_c(p_{diff}) \tag{5.1}$$

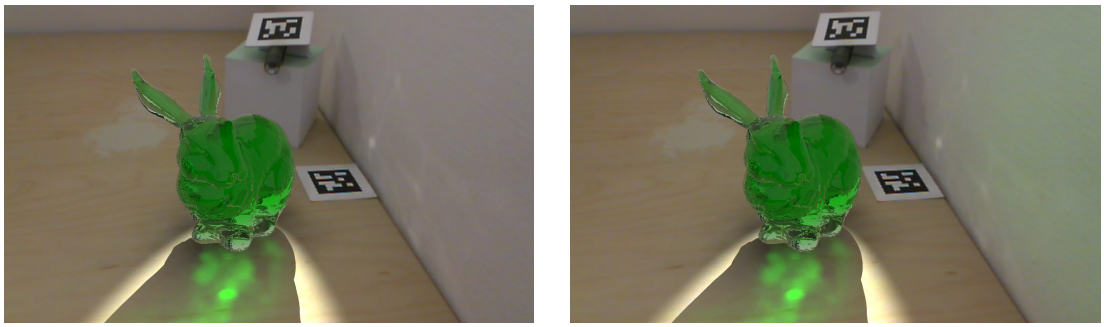
$$L_{refl} = L_{refl} + L_c(p_{refl}) \tag{5.2}$$

$$L_{refr} = L_{refr} + L_c(p_{refr}), \tag{5.3}$$

where L_d is the diffuse radiance, L_{refl} is the radiance from the reflected, and L_{refr} is the radiance from the refracted surface. The caustic radiance is defined with L_c and returns the corresponding radiance for the passed surface position, in which p_{diff} is a diffuse surface position, p_{refl} is a reflected surface position, and p_{refr} is a refracted surface position. Figure 5.3b depicts the caustic effect for all three cases.

Virtual Point Lights So far, the caustic radiance directly influences a diffuse surface. However, this additional radiance also affects other nearby objects, similar to an indirect light bounce on a diffuse surface. Generally, a virtual point light should be placed at an illuminated object. In case of a reflective or refractive object, our approach reflects or refracts the virtual point light

and places it on the final hit position of the photon. The intensity of this virtual point light depends on the photon intensity and the material properties at the hit location. Figure 5.4 shows an example image without this effect and with the indirect illumination from additional VPLs placed on the caustic.



(a) without VPLs (b) with VPLs

Figure 5.4: Caustic on a real desk. Image (a) without VPLs and Image (b) with VPLs resulting in an indirect illumination of the real wall.

Nr	Path	Description
1	LSS'*DE	Caustic from a reflective object.
2	LSSS'*DE	Caustic from a refractive object.
3	LSS'*DDE	Indirect illumination of a diffuse surface from reflected photons.
4	LSSS'*DDE	Indirect illumination of a diffuse surface from refracted photons.

Table 5.1: Possible light paths for the caustic simulation. The term $L'*$ in the light path describes multiple light bounces.

Table 5.1 summarizes the possible light paths that are handled by our caustic method. The multiple bounces $L'*$ suffer from the same limitations as in the reflective and refractive radiance computation. Also note that we currently only support spotlight sources in our caustic simulation.

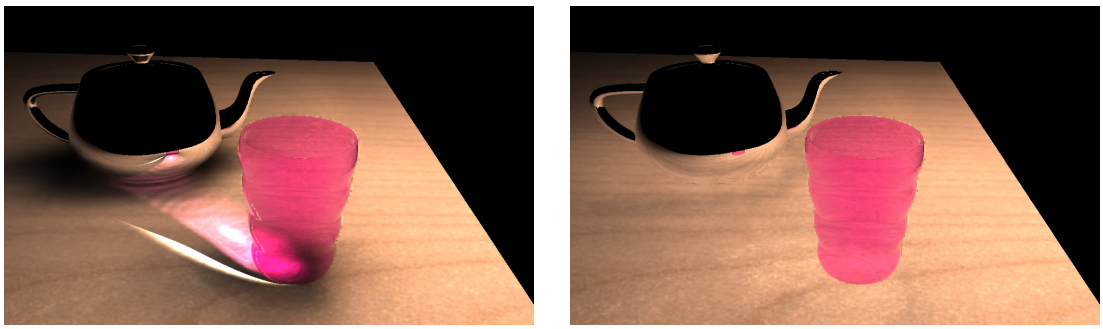
The images in this chapter already incorporate the virtual caustics in a mixed-reality scene. Differential rendering accounts for the various combinations of real and virtual caustic casters and receivers, which we consult in the next part.

5.2 Differential Rendering

Merging the additional radiance from a caustic of a reflective or refractive object also requires a classification of the included components in the light path. Whether the photons scatter from a

real or a virtual object and finally hit a real or virtual diffuse surface determines the contribution to the corresponding differential rendering buffer.

The resulting radiance from the caustics affect the buffers L_r and L_{rv} when all components are real in the path between the light source and the diffuse surface hit. Otherwise, the radiance only contributes to the buffer L_{rv} when it contains one virtual object. Figure 5.5 shows these two buffers, which additionally contain the radiance from a refractive caustic. The final output image is shown in Figure 5.3b.



(a) L_{rv} (b) L_r

Figure 5.5: The two differential rendering buffers from a refractive caustic. L_{rv} contains the virtual caustic (Image (a)) whereas the buffer L_r is not affected (Image (b)). Note that the upper part of the glass is missing in the reflection of the teapot because the reflected and refracted second-order ray finally hits the environment map and therefore contributes to L_{env}, see Section 4.2.3.

This means that the caustic radiance is added to the differential-rendering buffers. Formally, this is expressed as:

$$L_{rv} = L_{rv} + L_c \tag{5.4}$$

$$L_r = L_r + L'_c \tag{5.5}$$

, where L_c is the radiance from the caustic and L'_c contains the caustic radiance only from real components. It is defined as:

$$L'_c = \begin{cases} L_c & \text{if} \quad r(ls) \wedge r_p(p, p_{hit}) \\ 0 & \text{otherwise} \end{cases} , \tag{5.6}$$

where the argument ls is the light source that emits the photons, argument p is the visible point on the caustic caster that scatters the photon, and p_{hit} is the final position of the photon on a diffuse surface. The helper function $r(x)$ evaluates the real-virtual flag of a light source, as in Equation 4.9, and the function $r_p(p, p_{hit})$ classifies the path between the visible point on the caustic caster p and the final hit location p_{hit}, see Equation 4.15. Instead of two single locations, we need to consider the hole path, because a photon may bounce multiple times on reflective or refractive objects before it lands on a diffuse surface.

CHAPTER 6

Implementation

The previous two chapters described the general concept of the radiance computation of reflections, refractions and caustics and their integration in a mixed-reality environment. Some implementation-specific details were omitted for the sake of clarity. The following part deals with these gaps and presents the techniques behind the intersection approximation for reflections and refractions. Beyond that, this chapter also describes the generation of the photon map and the caustic buffer and their application to simulate caustics on diffuse surfaces.

A brief technical specification of the programming environment follows before we turn to the implementation details. The RESHADE framework is implemented in C# and uses DirectX 10 as the rendering API, wrapped with the SlimDX library, and HLSL with Shader Model 4 as the shading language.

6.1 Deferred Shading

One of the main building blocks in the RESHADE framework, especially for the differential instant radiosity method, is *deferred shading* [1]. Deferred shading decouples the rendering of the object geometry from the lighting computation. This means that in a first step, all objects are rendered once from the viewpoint of the camera. The objects are transformed into screen space and their geometric attributes and material parameters are stored in several texture buffers. This resulting set of textures is called a geometric buffer, or short *G-Buffer*.

Accordingly, the illumination computation of all contributing light sources is carried out only with the data stored in the G-Buffer, which is an advantage in many ways. The G-Buffer is created once per frame, consequently also the object geometry is rendered once, regardless of the number of light sources. Moreover, the lighting computation takes only visible pixels into account. Apart from the performance gain, this method also has advantages from a software-engineering perspective. The G-Buffer acts like an interface between the geometry stage and the lighting stage. Introducing a new type of object (for example a new material) requires no adjustment of the lighting stage. Conversely, a new type of light has no influence on the geometry stage, as long as both conform to the interface of the G-Buffer.

6.1.1 Primary G-Buffer

One of the first tasks in the rendering system of the RESHADE framework is the creation of the primary G-Buffer, $GBuf_{view}$, from the view point of the camera. This pass draws all real and virtual objects, including the reflective and refractive objects. Each value in the G-Buffer corresponds to a set of attributes from an object. Table 6.1 lists the content of the primary G-Buffer.

Nr	Attribute	Description
1	Color	The diffuse surface color.
2	Depth	The depth of the surface in normalized device coordinates.
3	FlagRV	The flag to distinguish between a real and virtual surface.
4	Material	The material attributes of the surface, e.g. the diffuse intensity.
5	Normal	The normal of the surface in world-space coordinates.
6	Reflectance	The reflectance of the surface, which is usually zero for a diffuse surface and one for a reflective or refractive surface.

Table 6.1: Content of the primary G-Buffer $Gbuf_{view}$.

The primary G-Buffer $GBuf_{view}$ is passed to the lighting stage, which computes the diffuse radiance (L_d) for all visible surfaces. Note that each value in the G-Buffer represents a visible surface. The radiance computation evaluates all direct light sources and all virtual point lights with the stored surface attributes in the G-Buffer. Except for the position, all surface attributes can be directly retrieved from the G-Buffer. Utilizing the view-projection matrix of the camera, the world position is reconstructed with the stored depth and its corresponding coordinates in screen space. Note that this layout is only a semantic consumption of the G-Buffer. Technically, we combine several attributes to lower the memory size of the buffer. For instance, the real-virtual flag (FlagRV) is packed together with the surface depth (Depth). A positive depth value represents a real surface whereas a negative value indicates a virtual surface.

6.1.2 Reflective and Refractive G-Buffer

As already stated, our intersection method determines the diffuse surface visible from the reflected and refracted view direction. The resulting surface attributes are written into two additional G-Buffers, one for the reflected surface, $GBuf_{refl}$, and one for the refracted surface $GBuf_{refr}$. Note that both G-Buffers are also in the screen space of the camera. This means that a value in one of these G-Buffers is a visible surface on a reflective or refractive object but it refers to the attributes of a reflected or refracted surface. Figure 6.1 illustrates the primary G-Buffer and the G-Buffer from the reflected view direction. The reflective G-Buffer contains only attributes in the area of the reflective teapot.

Table 6.2 shows the structure of the reflective and refractive G-Buffer. Compared to the primary G-Buffer, they also contain the Fresnel factor and the transmittance color filter. Instead of a depth value, this buffer stores the full world space position because reflections and refractions are not limited by the field of view and possibly lie outside the visible area. Furthermore, it

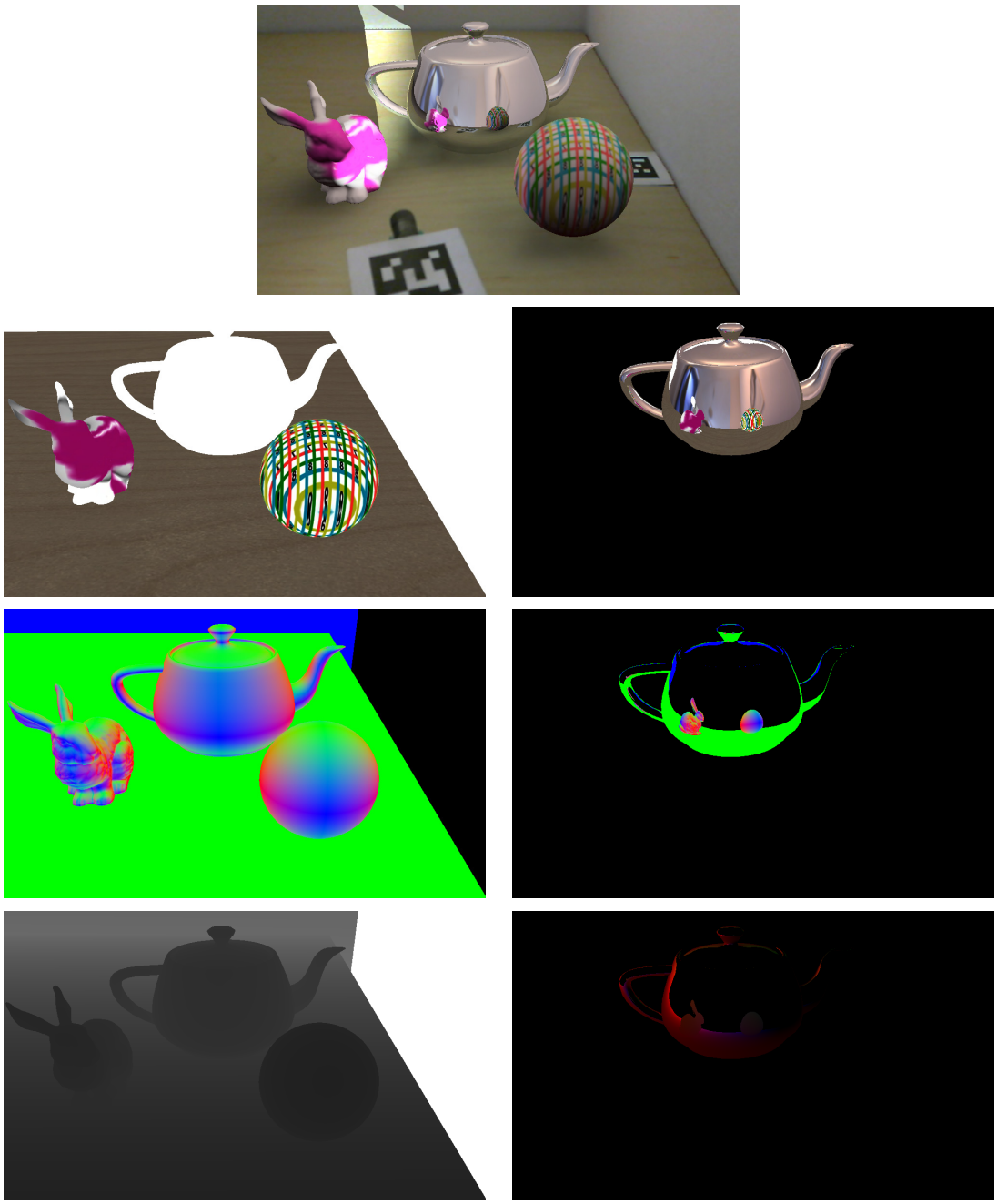


Figure 6.1: Stored attributes in the G-Buffer for real and virtual objects. The left column shows the primary G-Buffer and the right column the reflective G-Buffer. First row is the final output image, second row the diffuse color, third row the normal (with absolute values) and the last row contains the depth and world position.

Nr	Attribute	Description
1	Color	The diffuse surface color.
2	FlagRV	The flag to distinguish between a real and virtual surface.
3	FlagPathRV	The flag to evaluate the path to this surface.
4	Fresnel	The Fresnel factor.
5	Material	The material attributes of the surface, e.g. the diffuse intensity.
6	Normal	The normal of the surface in world space coordinates.
7	Position	The world space position of the surface.
8	Transmittance	The transmittance color filter.

Table 6.2: Content of the reflective and refractive G-Buffer $Gbuf_{refl}$ and $Gbuf_{refr}$.

contains an additional flag for the reflective or refractive path. This flag is true when all components from the first intersection along the path to the reflected or refracted surface are real. Otherwise, it is false, so at least one bounce occurs on a virtual object. The two real-virtual flags help to determine the differential rendering buffers, L_r and L_{rv}, and are used during the back-projection to find a correct value for L_b. The reflective and refractive G-Buffer are passed to the radiance computation, which calculates the outgoing radiance and applies the differential rendering effect.

6.2 First Intersection for Reflections and Refractions

Our method uses on an image-space approximation to find the intersection point between the reflected or refracted ray and the objects in a mixed-reality scene. In contrast to a ray-tracing approach, each reflective or refractive surface interaction needs special treatment. Considering the possible light paths from Table 4.1, we separately handle the first surface bounce, or the first two surface bounces in case of a refraction. A subsequent step identifies the reflected or refracted surface, contained in the remaining path. Take for example the path $LDS'*SSE$ (number three in Table 4.1). Starting from the eye-point, our approach finds the refraction ray from the first portion (SSE) and determines the reflected and refracted surface in the remaining path ($DS'*$).

The next part describes the first surface intersection, including the computation of the reflection ray and the approximation of the refraction direction on two surface boundaries. Once the reflection and refraction rays are determined, they are passed further to the next stage, which finds the reflected and refracted surface position. We support different intersection modes, which are discussed in the subsequent sections: basic planar reflections in Section 6.3 and two more general intersection techniques that are based upon cubic environment mapping in Section 6.4, and reflective impostors in Section 6.5. Note that due to deferred shading, it is easy to switch to another intersection technique as long as the output is consistent with the format of the reflective and refractive G-Buffer.

6.2.1 First Intersection for Reflections

Initially, we need to intersect the view ray from the camera with the reflective object, which simply is the visible point in a *z-buffer* based renderer. Figure 6.2a outlines this process, where the first surface intersection is at location p_1. The incoming view direction is then reflected about the surface normal. This resulting ray is passed further to the actual intersection approximation that determines the reflected surface.

Note that the depth buffer is already populated with correct values from the primary G-Buffer pass. Hence, we can use the *equal* function for the depth comparison, which speeds up the rendering and computes the subsequent intersection only for truly visible surface points.

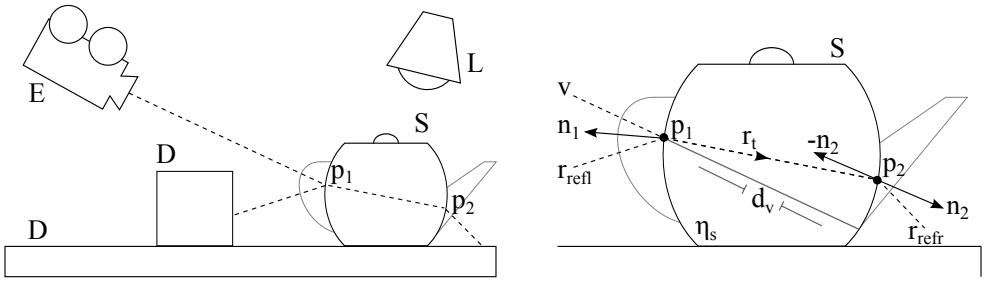


Figure 6.2: Initial surface intersection. Figure (a) shows the detection of the first intersection points for reflections (p_1) and refractions (p_2). These points are passed further to find the reflected and refracted surface points. Figure 6.2b illustrates the refraction approximation. The second surface point p_2 is estimated with the distance d_v between the front and back-faces: $p_2 \approx p_1 + d_v \cdot r_t$.

6.2.2 First Intersection for Refractions

Refractions on a single surface are equal to the aforementioned reflection case, except that this technique refracts the view ray. To enhance the visual quality of refractions, we additionally implement the method from Wyman [46]. This method refracts the viewing ray twice, on the first surface boundary and on the second surface boundary at the exit point of the ray, corresponding to p_1 and p_2 in Figure 6.2a.

Determining the exit point p_2 on the second surface boundary would require marching along the refracted ray r_t, see Figure 6.2b. The idea of Wyman is to approximate this exit point in a much faster way as:

$$p_2 \approx p_1 + d_v \cdot r_t, \tag{6.1}$$

where p_1 is the first intersection position, d_v is the distance between the depth of the front and back face along the view direction v, and r_t is the refracted view direction.

Utilizing deferred shading, our implementation draws the front-faces and the back-faces of the refractive object. Note that we only render the geometry of the refractive object in these two

passes. It stores the distance along the view ray and the surface normal in intermediate texture buffers. As suggested by Wyman, we store the attributes of the most distant back-faces. For this reason, the depth-comparison function is set to *greater* when the back-faces of the refractive object are rendered in the second pass. Figure 6.3 visualizes these buffers. Please note that the image is cropped to omit the black border areas.

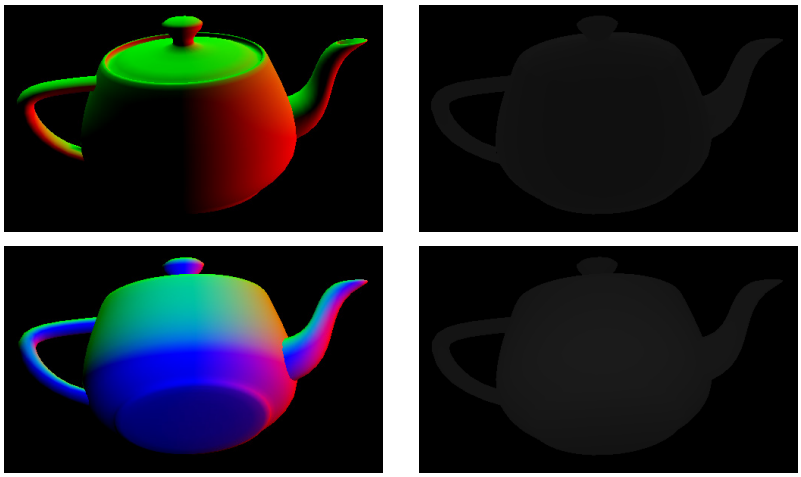


Figure 6.3: Intermediate texture buffers for the refraction. First row shows the normal and depth of the front-faces and the second row shows the values of the back-faces.

The following pass uses the intermediate texture buffers. First it reconstructs the surface position p_1, similar as in the primary G-Buffer pass, and computes the first refraction ray r_t. Moreover, it calculates the distance d_v between the stored front and back-face distance along the view ray. The approximated point $p_2 = p_1 + d_v \cdot r_t$ is projected into screen space and used to look-up the surface normal from the back-face texture. Now we are able to determine the second refraction ray. For this computation, we use the negative surface normal and the reciprocal index of refraction $\frac{1}{\eta_s}$, because this time the ray leaves the object. Finally, the resulting ray is forwarded to the intersection approximation.

A higher index of refraction bends the ray stronger, hence the second surface location p_2 may point to an area outside of the refractive teapot (black area in Figure 6.3). Consequently there is no valid surface normal. As described by Wyman, we substitute the missing normal with the projection of the refraction ray r_t onto the plane defined by the view direction v. Furthermore, we neglect total internal reflections. The idea of Wyman is to use the tangential vector at the surface location p_2 as the resulting refraction ray in such an event. This is essentially the same as the projection of the refraction ray r_t onto the plane defined by the second surface normal n_2.

Note that the article of Wyman also contains a more accurate approximation that requires a pre-computation step, see the previous Section 3.2. However, this simple refraction approximation already provides plausible results since small deviations in refractions are barely perceptible for the human eye.

6.3 Planar Reflection

Our implementation supports basic planar reflections. This special case is only able to find one surface intersection from the reflected view direction. Multiple bounces, refractions, or the scattering of photons are not possible with our method.

First, this technique mirrors the camera above the reflection plane, draws all possible reflected objects and stores their surface attributes in intermediate texture buffers. A subsequent pass computes the first intersection position and uses projective texture coordinates to acquire the reflected surface attributes from the intermediate textures. Finally, it writes these attributes into the reflective G-Buffer $GBuf_{refl}$. The possible light paths are: $LDSE$ and $LDDSE$.

6.4 Cubic Environment Mapping

Cubic environment mapping is a standard technique in real-time rendering that determines the reflection on a curved object. This method is used to compute the location of the reflected and refracted surface after determining the first reflection or refraction ray. We implemented the standard algorithm and additionally extend it with depth information to refine the intersection position.

6.4.1 Standard Cubic Environment Mapping

Every reflective and refractive object has its own environment map. The six faces of the cubemap represent the surrounding scene from the center of the object. For the creation, the environment mapping algorithm places a camera, with a field of view of $90°$, at the center of the reflective or refractive object. Then, it draws the surrounding objects and stores their surface attributes in several cube map texture buffers. Figure 6.4 shows the resulting cube map from the scene in Figure 6.5, the surrounding objects are captured from the center of the teapot. The coordinate axes define the view and up direction of the camera for each individual face as specified in the DirectX documentation [27]. Moreover, it keeps the distance of the surface to the center that is later used with the look-up direction to reconstruct the world space location. Note that we utilize a *geometry shader* and draw the surrounding objects in one pass, instead of six passes for each face, as described in the DirectX SDK [28].

An additional pass identifies the reflective or refractive ray for the first surface intersection. The resulting ray acts as the look-up vector for the cube map. This queried surface information is then written into the reflective or refractive G-Buffer.

The standard approach has several limitations. The look-up direction for the cube map just depends on the reflection or refraction direction, i.e. it ignores the start position of the ray. Moreover, the captured environment is only accurate at its center. Figure 6.5a depicts these limitations, see the wrong shadow in the reflection. Moreover, when the camera moves, the surface appears to swim in the reflection.

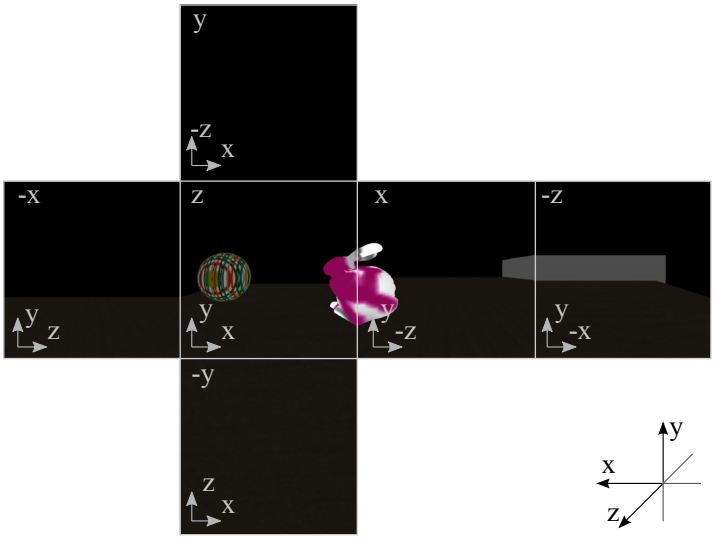


Figure 6.4: Cubic environment map. Illustrates the viewing coordinate system for the corresponding faces.

6.4.2 Layered Cubic Environment Mapping

For this reason we extend the standard approach with the more sophisticated method from Umenhoffer et al. [44]. This technique additionally utilizes the distance to the surrounding objects in the intersection computation and is therefore able to approximate intersections from nearby objects more accurately. Furthermore, it supports self-reflections and multiple bounces on reflective and refractive surfaces, see the handle of the teapot in Figure 6.5b.

Our implementation creates two additional cube map layers, a layer for the front-faces and one for the back-faces of the reflective or refractive object. As suggested by Umenhoffer, our technique marches along the reflective or refractive ray. At each step, it computes a new look-up vector from the corresponding position on the ray and retrieves the stored distance from the cube map. It compares this distance with the actual distance on the ray, finding the first pair of overshooting and undershooting, i.e. the ray distance is greater or smaller compared to the stored distance. These two locations are further used to refine the intersection position with a secant search, as described by Umenhoffer.

This intersection computation is repeated for each of the three layers. After all, the adjusted look-up vector of the closest distance is used to read the surface attributes from the corresponding cube map layer. As before, these attributes are output into the reflective or refractive G-Buffer.

Our implementation achieves good results for convex objects but produces severe artifacts for flatter surfaces, unfortunately also with the enhanced layered cube map approach. Note that we will compare the different intersection approximations in Chapter 7.

(a) (b)

Figure 6.5: Cubic environment mapping. The standard approach uses only the reflection direction to look-up the reflected surface, what produces wrong results, e.g. the wrong shadow in the reflection of the teapot (Image (a)). Layered cubic environment mapping traces along the reflection ray to refine the intersection approximation, see Image (b). Note the self-reflection of the teapot-handle.

6.5 Impostors

An impostor approximates the geometric shape of an object with a rectangle and several textures, containing its surface attributes. Instead of the entire geometry, the algorithm just intersects the reflected or refracted ray, from the first surface intersection, with the impostor rectangles.

In nearly all of our mixed-reality scenarios, the impostor method provides a plausible and stable intersection approximation, which also includes multiple bounces on reflective and refractive objects. Compared to the aforementioned implementation of layered cubic environment mapping, impostors are faster and are also able to handle flat reflective or refractive surfaces. Apart from that, their usage is intuitive and they are straight-forward to implement.

Our approach is based on the method of Popescu et al. [35], which we briefly mentioned in Section 3.1. In their article, Popescu et al. describe two types of impostors: reflective billboards and reflected depth maps, which additionally utilize stored depth values to refine the intersection position. However, our refinement method does not rely on their concept of a *simplified rotated depth map*, in exchange it combines ideas from *relief mapping*, invented by Policarpo et al. [34], and the aforementioned layered cubic environment mapping from Umenhoffer et al. [44]. In combination, these two methods provide a fast and robust intersection refinement and are, in our opinion, more intuitive to implement compared to the original approach.

6.5.1 Impostor Array

Initially, a set of impostor objects is assigned to each reflective or refractive object. This means that each reflective or refractive object has a corresponding impostor set, containing all other real and virtual objects of the mixed-reality scene. Figure 6.6a illustrates this for a scene with two reflective and three diffuse objects: the impostor set of the reflective teapot S_1 is $\{D_1, D_2, D_3, S_2\}$ and the impostor set for the reflective sphere S_2 is $\{D_1, D_2, D_3, S_1\}$.

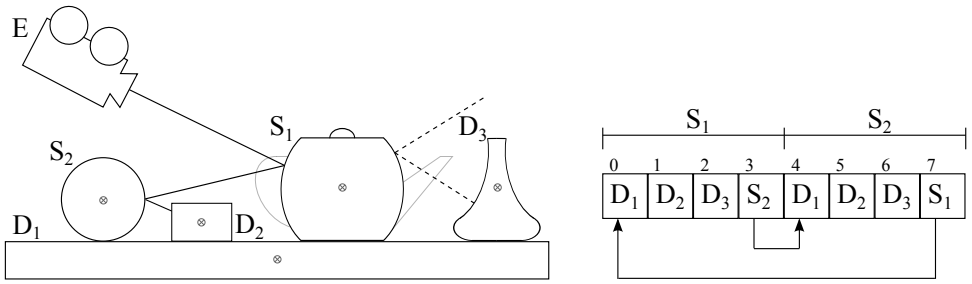


Figure 6.6: Each reflective object has a set of impostors. In Figure (a), the impostor set for the object S_1 is $\{D_1, D_2, D_3, S_2\}$ and for object S_2 it is $\{D_1, D_2, D_3, S_1\}$. Figure (b) shows the structure of the impostor array. Reflective and refractive objects point to their corresponding impostor subset, indicated by the arrows.

The several sets of impostors are combined in a single array, in which one array element refers to a specific impostor. Impostor entries of reflective and refractive objects additionally link to the corresponding range of their impostor subset, depicted with the arrows in Figure 6.6b. Hence, this array defines all necessary data for the intersection computation.

6.5.2 Impostor Update

Before their actual usage, we need to refresh the impostor data once per frame. Each element in the impostor set is updated, although this update is only required for a change in the alignment between the reflective object and the impostor object. Note that this update procedure, as well as the intersection, is identical for reflective and refractive objects.

For this recreation, we place a camera at the center of the reflective object, render the impostor object and store the surface attributes in several textures. This update process is repeated for every impostor object. The view direction of this camera is along the center of the two objects, which is from c_s to c_i as illustrated in Figure 6.8a. To reduce the distortion for flat objects, we also allow to manually set the view direction. For instance, the desk D_1 in Figure 6.6a produces less intersection artifacts when the view direction is aligned with the up vector of the world coordinate system. As described by Hu and Qin [16], we use an *orthographic projection* to draw the impostor object. This projection matrix is defined by the minimum and maximum extents of the transformed axis aligned bounding box (AABB) with the view matrix of the impostor camera.

The surface attributes are output into several texture arrays, in which one slice refers to a specific impostor. Figure 6.7 visualizes two textures with surface attributes from the three impostors used in the scene of Figure 6.9. An impostor array element also contains the center of the near plane p_n, the center of the far plane p_f (both in world space) and the impostor normal n, which corresponds to the view direction of the camera. Additionally, we store the view-projection matrix of the camera that inherently specifies the dimension of the impostor rectangle, i.e. points inside the rectangle are within the range -1 to +1 in normalized device coordinates (NDC).

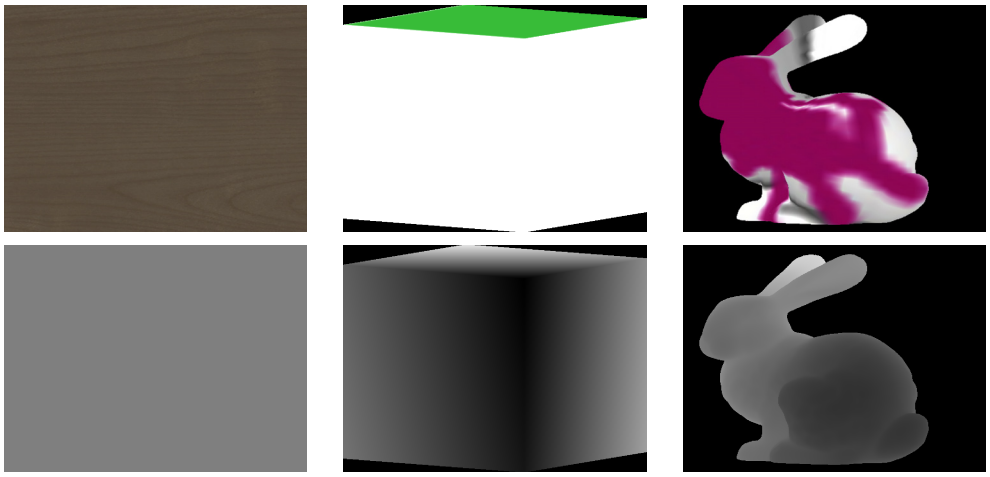


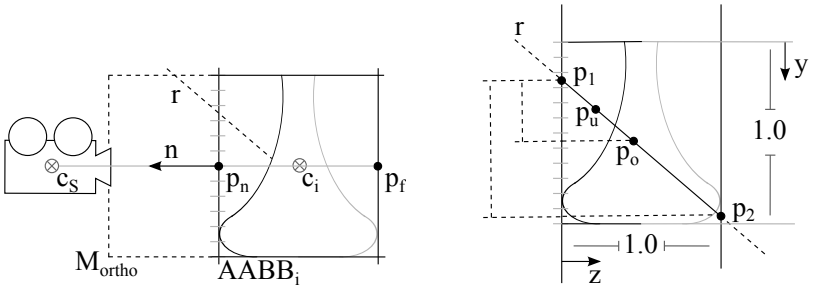
Figure 6.7: Each column represents an array slice that refers to one specific impostor, i.e. the real desk, the real box, and the virtual bunny. The first row shows the surface color and the second row the depth values for each impostor from Figure 6.9.

6.5.3 Impostor Intersection

Once the impostor set is created, our method uploads the hole impostor array to the GPU with the initial range pointing to the corresponding subset of the reflective object. Then it iterates over this set, defined by the passed range, and finds the closest intersection with the reflection ray. This means that every impostor object in the subset is intersected with the passed ray.

If the closest intersection point lies on a diffuse surface, the algorithm returns the surface attributes. Otherwise, for a reflective surface, a new ray is computed from the stored surface attributes and is used to compute the next intersection. This time, it iterates over the subset of the intersected impostor which is determined by the stored range. This process is repeated until the ray hits a diffuse surface or reaches a maximum number of bounces. Note that in case the algorithm misses all impostors, it queries the environment map with the corresponding ray and outputs the captured radiance value instead. Moreover, if a second-order ray hits a refractive object it bends the ray only at the first surface boundary, a limitation of our approach.

We support two different intersection methods. The first uses only the rectangle of the impostor, similar to the billboard reflection from Popescu et al. [35]. This method intersects the ray with the plane defined by the center of the near plane p_n and the impostor normal n, see Figure 6.8. In a subsequent step, we transform this point with the stored view-projection matrix of the impostor camera and map it to the texture coordinate range between 0 and 1. The ray intersects the impostor rectangle when the coordinates of the intersection lie between 0 and 1, otherwise the ray misses the impostor. Moreover, the algorithm retrieves the depth value from the impostor texture, in which a valid entry indicates that the ray truly hits an impostor surface. This means that the ray may intersect the rectangle but misses the surface, i.e. the black areas in Figure 6.7. In Figure 6.9a, the mixed-reality scene is rendered with the basic billboard reflections.



(a) (b)

Figure 6.8: Figure (a) visualizes the creation of an impostor. A camera is placed at the center of the reflective or refractive object c_s with the view direction pointing to the impostor center c_i. The axis aligned bounding box (AABB) determines the projection extends. Figure (b) shows the intersection of a ray with an impostor. Initially, it detects the undershooting (p_u) and overshooting (p_o) points on the ray r. It then uses these two points to refine the intersection with a binary search.

(a) Reflective billboards (b) Impostors with a depth map

Figure 6.9: Figure (a) was rendered with reflective billboards, in which the bunny appears too near in the reflection of the teapot. Also note the wrong reflection of the green top face of the real box. Compare with Figure (b), which produces a more accurate reflection by utilizing the stored depth values in the impostor intersection.

The second method utilizes the stored depth to refine the intersection position and produces more accurate results, see Figure 6.9b. Essentially, we trace a ray in a depth map, similar to relief mapping from Policarpo et al. [34]. They perform a fast linear search to detect the first intersection and refine this position with a subsequent binary search. However, we changed the linear search to the method from Umenhoffer et al. [44], which detects a pair of undershooting and oversshooting points, see the points p_u and p_o in Figure 6.8b. We find this method more practical because it reduces the interval for the binary search. For this, we intersect the ray with the impostor planes located at p_n and p_f, resulting in points p_1 and p_2. We then map them to the texture coordinate range and determine the step size along the ray between p_1 and p_2. Moreover, we compute the delta increments along the ray with similar triangles. Now, our algorithm marches along the ray and detects the first pair of under- and over-shooting points. i.e. it compares the depth on the ray with the stored depth in the impostor texture, as described by Umenhoffer et al. [44]. Finally, the algorithm refines these two points with a binary search (a secant search would also be possible) and outputs the corresponding surface attributes from the impostor texture into the reflective or refractive G-Buffer.

6.6 Caustics

We simulate caustics with the image-space method from Wyman [49]. This technique requires two passes. In the first pass it emits photons from the light source onto the reflective or refractive object, i.e. the caustic caster, and stores the hit position in a texture, called the photon map. The second pass splats these photons into an additional texture, the caustic buffer, which is used to visualize the caustics.

6.6.1 Photon Map

For the creation of the photon map, we render the caustic caster from the view-point of the light source and determine the reflection or refraction ray from the first intersection. This ray is passed further to the intersection approximation that determines the final diffuse surface with one of the previously discussed methods, i.e. cubic environment mapping or the impostor technique. We store attributes of the hit diffuse surface and attributes of the arriving photon in the photon map, see Table 6.3.

Nr	Attribute	Description
1	Direction	The incident direction of the arriving photon.
2	FlagPathRV	The real-virtual flag to evaluate the photon path.
3	Intensity	The intensity of the arriving photon.
4	Normal	The normal of the hit diffuse surface.
5	Position	The world space position of the hit diffuse surface.

Table 6.3: Content of the photon buffer.

The photon intensity depends on the flux of the light source, weighted by the number of emitted photons. We use an occlusion query to count the emitted photons as proposed by Shah et al. [39]. Note that occlusion queries are asynchronous, hence we use the result from the evaluation of the last frame. In addition, the photon intensity is attenuated by the distance to the spotlight and the cone filter of the spotlight as well as by the Fresnel factor and transmittance color along its path. Figure 6.10 shows the content of the photon map in light-space from the final image in Figure 6.11a.

(a) Incident direction (b) Intensity (c) Surface normal

Figure 6.10: Photon map. Contains the incoming photon direction (absolut values), photon intensity and the surface normal of the diffuse hit position.

6.6.2 Caustic Buffer

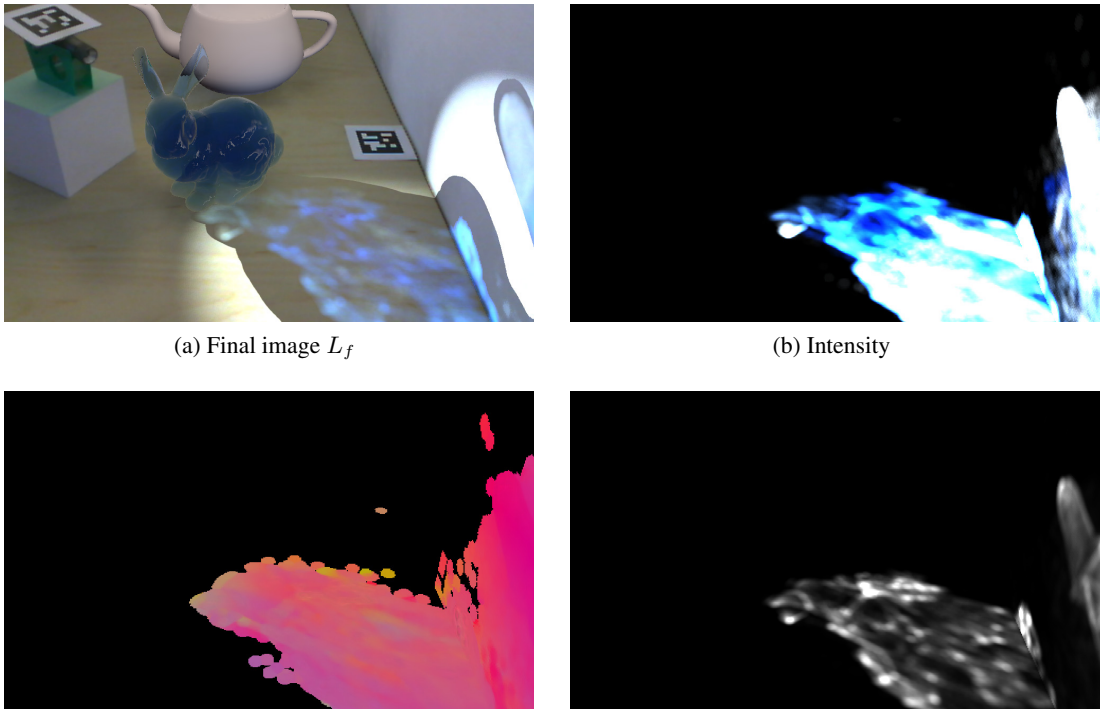
Photon energy not only affects the hit location but also the neighboring area. This energy accumulation is estimated with a caustic buffer as described by Wyman. Therefore, we render a grid with the same dimension as the photon map and look-up the corresponding world position from the stored photon texture in the vertex shader. This point is expanded to a surface aligned quad in the geometry shader, in which the size of the quad is computed from the world-position of the adjacent photons, similar as in the work of Wyman [48]. Moreover, we constrain this splat size by a user-defined minimum and maximum value. Additive blending is enabled to count the incoming photons on an area, gather the photon intensity and the incident direction. Table 6.4 summarizes the content of the caustic buffer.

The resulting caustic buffer is in screen-space and each value contains the gathered photon information from its neighboring area, see Figure 6.11. One can imagine this as a kind of light-map, which is dynamically recreated every frame. Analog to a light-map, each entry in the caustic buffer modifies the color of the corresponding surface. These values are simple treated as an exclusive point light source for one specific diffuse surface, i.e. we evaluate the BRDF for this surface and the caustic light source. Note that we compare the stored depth in the caustic buffer with the depth of the diffuse surface form the primary G-Buffer. The caustic effect is only applied when these depth values are within a specified epsilon range.

Additionally, the intensity is scaled to adapt for the photon concentration, so areas with a higher photon count should appear brighter than areas containing fewer photons. We multiply

Nr	Attribute	Description
1	Depth	The depth of the hit diffuse surface, which is used to reconstruct the world-space position.
2	Direction	The accumulated incident direction of the arriving photons. The average incident direction is determined by a division through the number of incident photons.
3	FlagPathRV	The real-virtual flag of the caustic splat.
4	Intensity	The gathered intensity of the arriving photons.
5	Photons	The number of incoming photons.

Table 6.4: Content of the caustic buffer.



(a) Final image L_f

(b) Intensity

(c) Average direction

(d) Photon count

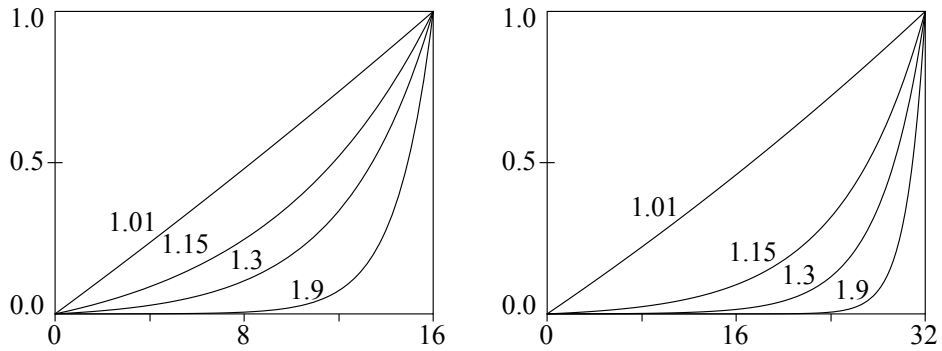
Figure 6.11: Caustic buffer. It contains the gathered intensity, the averaged incoming photon direction, and the number of arriving photons.

the photon intensity with an empirically determined scaling factor k_c:

$$k_c = \frac{b^{min(n,n_{max})} - 1}{b^{n_{max}} - 1}, \tag{6.2}$$

where n is the number of incoming photons, the parameter n_{max} is the required photon count for a maximum intensity and parameter b weights the interpolation. Figure 6.12 visualizes this function for different parameters b and the maximum intensity n_{max} at 16 and 32 incoming photons.

Accordingly, this caustic buffer is used in the lighting computation of a diffuse surface to visualize the caustic pattern. Consider that we use the caustic buffer also for the reflected and refracted surfaces. To accomplish this task, we project the reflected or refracted surface point into screen-space and look-up the corresponding values in the caustic buffer. As with a diffuse surface, we compare the depth values and apply the caustic effect only when they are within a specified epsilon range.



(a) $n_{max} = 16$ (b) $n_{max} = 32$

Figure 6.12: Scaling factor for the caustic intensity. Illustrates the scaling factors for different weighting factors b. Figure (a) has the maximum intensity at 16 arriving photons and Figure (b) reaches the maximum at 32 incoming photons. The scaling factor lies between zero and one.

CHAPTER 7

Results

This chapter presents output images of several mixed-reality scenarios, simulated with our rendering approach. Our primary test platform is a PC with an AMD Phenom X2 Processor at 3.10 GHz, 4 GB of main memory, and a NVIDA GeForce GTX 260 GPU with 896 MB of dedicated memory. We use a Logitech C250 Webcam to capture the mixed-reality scenario and an uEye camera from IDS with a fish-eye lens to generate the environment map. The environment map is pre-captured and identical for all scenes. The lighting conditions are nearly identical in our test cases and this capturing has a minimal influence on the frame-rate.

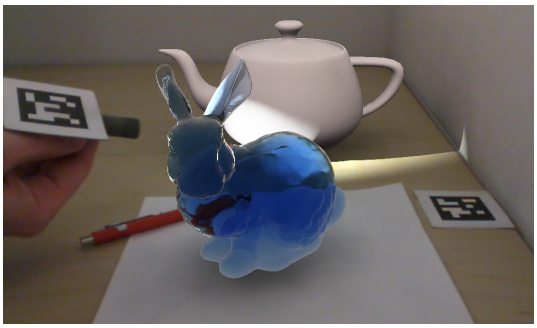
The second test platform is a PC with an Intel Core2 Quad CPU at 2.8 GHz, 8 GB of main memory, and a NVIDIA GeForce GTX 580 with 1.5 GB of memory. For this test-platform, the environment map is dynamically recreated once per frame. We use an uEye camera from IDS with a fish-eye lens and a Logitech HD Pro 910 webcam to capture the mixed-reality scenario. In general, we use the primary test-platform, otherwise it is mentioned for each figure.

The resolution of the final output images is 1024x768 pixels. Differential instant radiosity uses 256 VPLs for the light simulation, with an imperfect shadow map size of 128 pixels and 1024 points per VPL to approximate the scene. The linear search interval of the impostor intersection has 64 steps and the binary search has 8 steps. The resolution of the impostor texture is 512x512 pixels. Environment map intersection uses a resolution of 512x512x6 pixels. Photon mapping was carried out with a photon map and a caustic buffer size of 512x512 pixels for each. We use a minimum splat size of 0.005 and a maximum splat size of 0.007 with a maximum intensity at 32 photon hits, and a base factor of 1.3. All of these parameters can be manually adapted, unless otherwise mentioned we use these standard settings in combination with reflective impostors to generate the final output images.

Figure 7.1a shows the virtual Utah teapot with a reflective material. The teapot reflects the virtual Stanford bunny and a virtual sphere. The objects in the scene are illuminated with a virtual spotlight. Note that the pink color bleeding from the bunny on the sphere appears in the reflection of the teapot. Moreover, the spotlight directly affects the handle of the teapot. This scene is rendered with 11 fps. The frame-rate increases to 22 fps when the number of virtual

point lights is reduced to 64 VPLs. In general, the frame-rate mostly depends on the number of virtual point lights and on the number of visible reflective and refractive pixels.

(a) Reflective virtual teapot (11 fps)

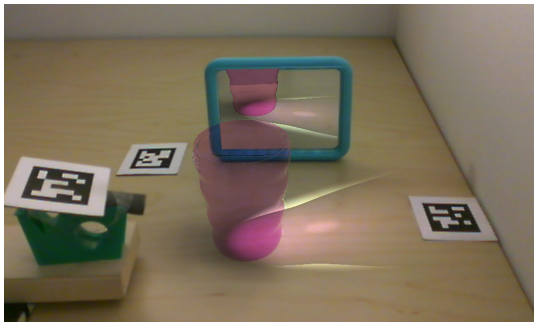


(b) Refractive virtual bunny (7 fps)

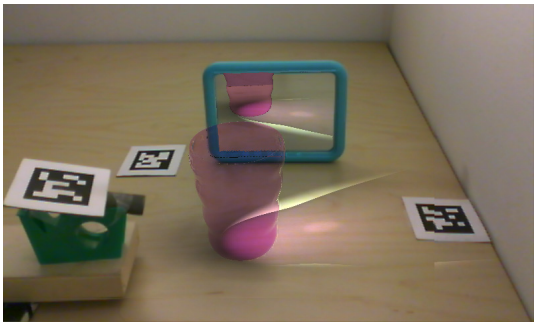
Figure 7.1: Figure (a) shows the virtual Utah teapot with reflective material illuminated by a virtual spotlight, rendered with 11 fps. Note the pink color bleeding on the sphere in the reflection of the teapot. Figure (b) shows the virtual Standord bunny with a translucent blue material, rendered with 7 fps.

In Figure 7.1b, the virtual Stanford bunny has a refractive colored material. The thickness of the bunny influences the translucent material color. For instance, the body of the bunny appears more blueish than its ears. The bunny realistically refracts the real desk and the virtual teapot. Note that the real red pen has no pre-modeled geometric representation, so the intersected surface lies on the real desk. However, these small errors are hard to detect for the human eye. The frame-rate is about 7 fps.

In case of caustics, the performance also depends on the number of emitted photons, i.e. the size of the photon map. Figure 7.2 shows a scene with a real mirror that reflects a virtual glass. The virtual glass casts a caustic on the real desk, which also appears in the reflection



(a) Caustic from a virtual object. (6 fps)



(b) Image size 800x600, photon map 256x256 (10 fps)

Figure 7.2: Caustic from a virtual object in a real mirror.

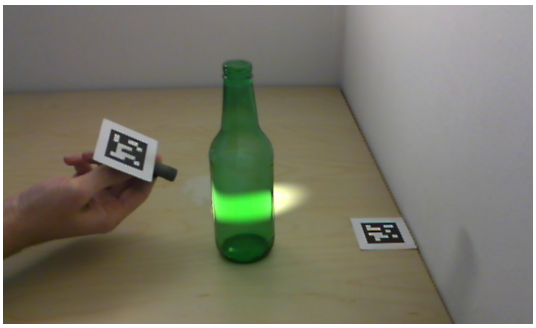
of the real mirror. The image is rendered with 6 fps. Reducing the resolution of the photon

map to 256x256 pixels, which corresponds to less emitted photons, increases the frame-rate to 8 fps. Moreover, setting the image size to 800x600 reduces the amount of visible pixels and therefore also the complexity of the lighting computations. The frame-rate increases to 10 fps, see Figure 7.2b.

Figure 5.1a from the introduction in Chapter 5 was rendered with the second test platform, which shows a reflective virtual bunny illuminated by a virtual spotlight. It casts a reflective caustic onto the real desk. All intersection computations occur on the GPU, so the second test platform achieves higher frame rates due to its more powerful hardware configuration. This scene was rendered with 19 fps.

Our approach handles various lighting situations in a mixed-reality environment. Figure 7.3 shows a scene with a real bottle illuminated by a virtual spotlight. The real bottle is pre-modelled and reacts to the virtual spotlight in a realistic manner.

(a) Reflection (8 fps)



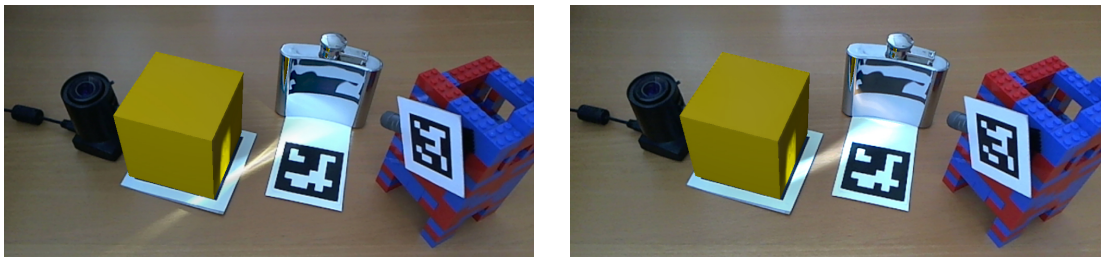
(b) Refraction (9 fps)

(c) Caustic (6 fps)

Figure 7.3: Light effects from a real bottle illuminated by a virtual spotlight.

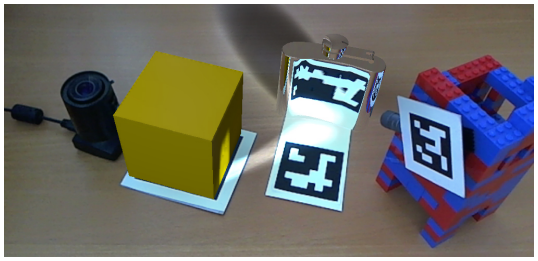
Figure 7.4 compares the lighting effects for a real or virtual reflective object that is illuminated by a spotlight source, either real or virtual. In Figure 7.4a, the real caustic wrongly appears on parts of the desk that should ideally be occluded by the virtual yellow box. The reason is that the virtual yellow box blocks the photons of the real spotlight source, so the real differential-rendering buffer L_r remains empty for these areas. Hence, differential rendering is not able to cancel out the real caustic on the desk resulting in a wrong illumination. These images are gen-

erated with the second test platform, resulting in a higher frame rate. Also note that the virtual yellow box is correctly reflected in the real pocket bottle.

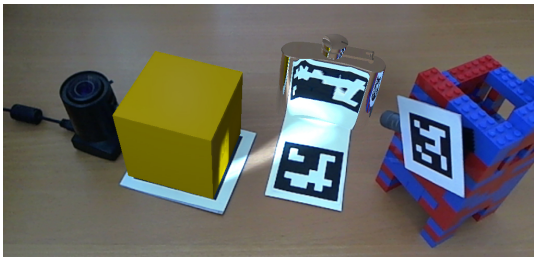


(a) Real light and real object (20 fps)

(b) Virtual light and real object (20 fps)



(c) Real light and virtual object (20 fps)



(d) Virtual light and virtual object (20 fps)

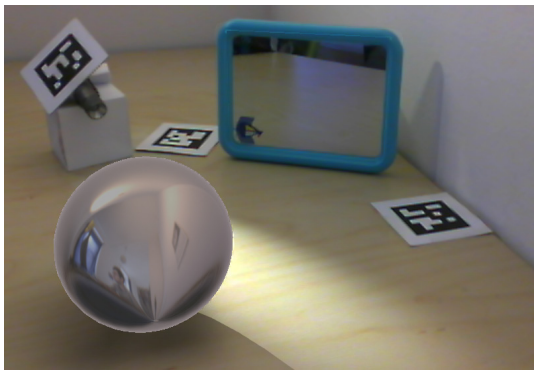
Figure 7.4: Comparision of light effects from real or virtual objects. Images are rendered with the second test platform.

Impostor reflections are able to handle multiple bounces. Figure 7.5 presents a scene with a real mirror, reflecting a virtual sphere with a reflective material. Multiple light bounces contribute to the image quality. However, more than three bounces are merely recognizable.

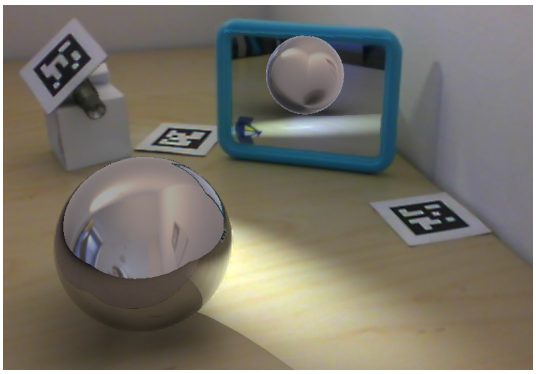
Reflective impostors provide convincing and stable results in most test scenarios. However, they are not able to handle self-reflections and artifacts may arise because not the whole geometry is visible from the center of the reflective or refractive object. For instance in Figure 7.6a, parts of the ears from the bunny are missing in the reflection of the teapot.

Layered cubic environment mapping is able to handle self-reflection, see Figure 7.6b. Our approach uses three environment layers, which produces good results for convex objects. Each reflective or refractive object has an exclusive environment map. Compared to the impostor reflections, the frame-rate is lower because it needs to render the surrounding scene for every layer.

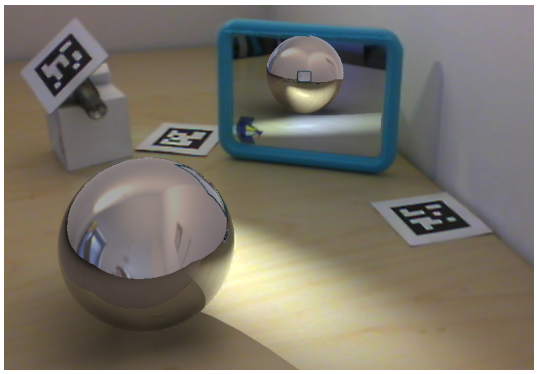
Figure 7.7 compares different intersection methods. Cubic environment mapping produces wrong results for flat reflective or refractive surfaces, see Figure 7.7a. The reflected surface appears too near in the reflection. Our approach uses a static number of cubic environment layers. Hence, occluded surfaces are missing and produce artifacts in the approximation of the reflection, see Figure 7.7b. With the standard billboard approach, the reflections also appears too near in the mirror, compare Figure 7.7c with Figure 7.7d. Furthermore, flat objects need a special handling with the standard billboard approach, i.e. the alignment of the view direction during



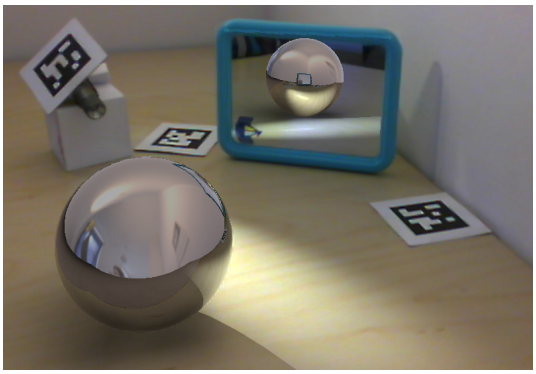
(a) Zero iterations (16 fps)



(b) One iteration (10 fps)

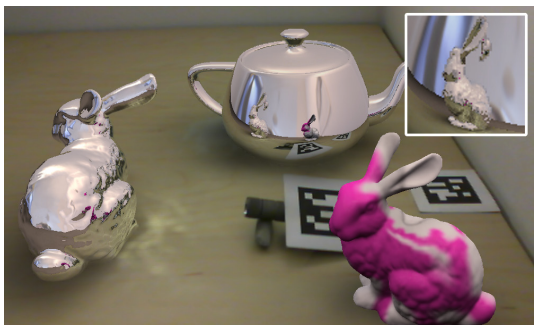


(c) Two iterations (10 fps)

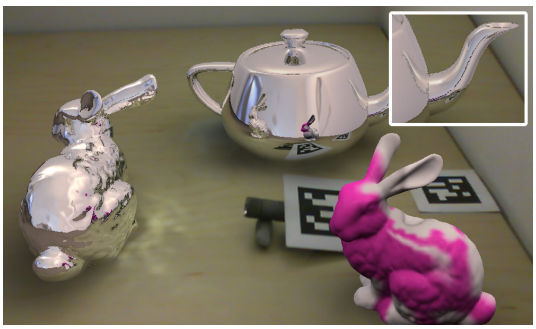


(d) Three iterations 10 fps)

Figure 7.5: Multiple reflections. A real mirror reflects a virtual sphere with a reflective material.



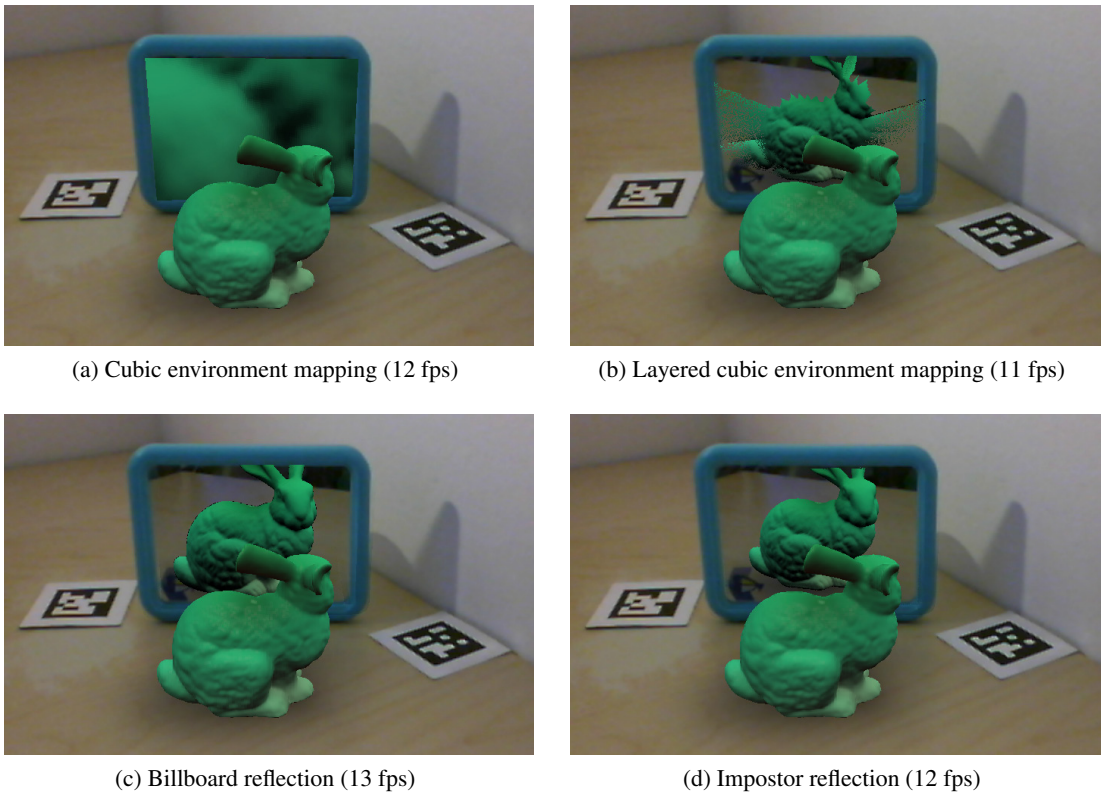
(a) Reflective impostors. (6 fps)



(b) Layered cubic environment mapping (3 fps)

Figure 7.6: Comparison of reflective impostors and layered cubic environment mapping, showing multiple reflections and reflective caustics. The detail in Figure (a) shows artifacts of the impostor approximation, in which the geometry might be missing in the reflection. The detail in Figure (b) shows a self-reflection.

the creation. Otherwise, the billboard approximation of the geometry appears in the reflection. These artifacts from the standard billboard reflections are more evident in Figure 6.9a from the Section 6.5. For instance, the alignment of the impostor normal was not adjusted for the real box, so the green top face wrongly appears in the reflection. Moreover, the reflected surrounding scene is to near in the reflection of the teapot.



(a) Cubic environment mapping (12 fps)

(b) Layered cubic environment mapping (11 fps)

(c) Billboard reflection (13 fps)

(d) Impostor reflection (12 fps)

Figure 7.7: This figure compares different intersection methods.

CHAPTER 8

Conclusion and Future Work

This thesis discussed a method to simulate the lighting effects for reflective and refractive objects in a mixed-reality environment. The presented method handles various lighting situations for real and virtual objects and light sources. It provides convincing results and runs at interactive to real-time frame rates, so user interaction is possible.

The proposed method in this thesis has some limitations and of course, there are a lot of possibilities for future improvements. The following part discusses these points and completes this thesis with a brief conclusion.

8.1 Limitations and Future Work

Differential Rendering Reflective and refractive effects are built upon the original differential instant radiosity technique. In certain cases, this method produces wrong double shadows and inconsistent color bleeding as discussed by Knecht et al. [22]. Furthermore, all real objects that contribute in the lighting computation must be pre-modeled. All these problems are already addressed in a follow up from Knecht et al. [23] but are not included in this approach.

Real caustics are only canceled out when they are not blocked by virtual objects. This means that the real caustic is blocked by a virtual object and the real differential part L_r does not contain the caustic contribution, see Figure 7.4a.

Currently, we store the computed radiance for a diffuse surface, a reflected surface, and a refracted surface together in the two differential rendering buffers L_r and L_{rv}. For differential rendering and the back-projection, it would be more manageable to have a separate radiance buffer L_r and L_{rv} for each of the three parts. Hence, it would be possible to correctly handle all missing cases in the back-projection, like the one shown in Figure 4.15. Moreover, with this adaption, we could separately apply the scaling factor α to a real reflected or refracted surface, which accounts for the relative error in the differential rendering effect.

Reflections and Refractions In most scenarios, reflective impostors provide stable results. However, not the whole geometry of the impostor object may be visible from the center of the

reflective or refractive object. In such an event, these areas are missing in the reflection or refraction, see Figure 7.6a. This limitation is solved in the work of Rosen et al. [37].

For refractions, our approach roughly approximates the second refraction direction and does not support total internal reflection. To enhance the visual quality, a future extension could be to integrate a more accurate computation of the second refraction direction with the method of Olivera and Brauwers [30] and consider total internal reflection with the method from Davis and Wyman [7].

During multiple bounces, the impostor refracts the ray only on a single surface. It is possible to store the front and back-faces also for an impostor, which allows computing the refraction on the entry and exit point of the ray. Furthermore, our approach traces only one ray direction when a second-order ray hits a refractive impostor.

Layered cubic environment mapping supports multiple reflections and refractions. Our approach currently generates only three layers, which works well for convex objects but fails for flat surfaces. A possible future improvement would be to dynamically generate these layers with depth-peeling, as mentioned by Umenhoffer [44]. This means that also occluded objects are visible from the reference point, which should eliminate the artifacts from Figure 7.7b.

Caustics We use a simple splatting technique to visualize caustics. Due to the regular sampling pattern of the photon map, the photon density may be too high or too low for some areas. Moreover, unnecessary photons are emitted that correspond to areas in the photon map not covered by the reflective or refractive object. The method of Wyman and Nichols [50] handles these problems with an adaptive photon generation.

Furthermore, the caustic algorithm would benefit from a new GPU shader model that supports tessellation. At the moment, the total number of the emitted photons and their alignment depends on the resolution of the photon map. With DirectX 11 and its tessellation functionality, it is possible to emit photons on demand. This means that the aforementioned adaptive caustic method from Wyman and Nichols [50] would be more intuitive to implement and would work faster on newer hardware architecture. Therefore, one of the next steps could be to migrate the framework to an actual DirectX 11 implementation, which includes the shader model 5 [28].

The caustic algorithm is an image-space method and due to the depth resolution, artifacts may arise in certain cases, e.g. the caustic is applied to a wrong diffuse surface. However, such cases are rare and are simply handled by a manual adaption of the epsilon value, which is used in the depth comparison. Nevertheless, an improved epsilon test would reduce the requirement for this manual adaption. For this reason, this test could also compare the involved surface normals, which then need to be additionally stored in the caustic buffer. Apart from that, caustics are only generated from a spotlight source, so a future work could generate caustics also from virtual point lights.

8.2 Conclusion

The intent of our approach was a perceptually plausible light simulation for reflective and refractive objects, including caustics in a mixed-reality environment. Our method extends differential instant radiosity, which is able to realistically simulate the light distribution in a mixed-reality

scene. We analyzed the additional light paths from reflective and refractive objects. This information was utilized to compute their outgoing radiance and augment the captured scene with virtual content. The differential rendering algorithm was improved with a back-projection method, which allows applying this effect also to reflected and refracted real surfaces. As a consequence, reflections or refractions seamlessly merge with the real scene and this improved the overall impression of a mixed-reality environment.

Although our approach applies the differential-rendering effect whenever possible, artifacts arise due to missing information in the back-projection. Therefore, it is important to have an accurate approximation of the material attributes from the real objects. Furthermore, it would be beneficial to incorporate an enhanced reflectance model that builds upon physical properties. Currently, our implementation uses a simple *Blinn-Phong* BRDF model. An alternative, for example, would be the Cook-Torrance [1] BRDF model, which also considers the Fresnel equations for a diffuse surface.

The intersection computation uses an image-space method with impostors. The impostor algorithm provides convincing results, even though the intersection result is only an approximation. We observed that small errors are barely noticeable for the human eye, especially for refractions. Hence, it would be interesting to perform a user study and evaluate different criteria for a plausible perception in a mixed-reality scene.

With the growing processing power of hardware, GPU ray-tracing, or a hybrid approach combined with a z-buffer based rasterizer, may become a valid alternative to simulate the light distribution in a mixed-reality scene. For example, image-space approximations have problems with concave objects and materials with a higher index of refraction, so a ray-tracing technique would provide more precise results in such cases. However, deformable or animated virtual objects require higher demands on the ray-tracer and on the computing power but are inherently handled by the impostor method on the other hand. Besides, augmented-reality software on mobile devices becomes increasingly popular. The hardware resources for these devices are limited by their size and other factors, like the battery life time. Accordingly, real-time rendering algorithms that produce convincing results but rely on fast approximations remain an alternative for this sector.

In addition, this thesis described a method to simulate caustics in a mixed-reality environment. Our method generates caustics from real or virtual spotlights and this light pattern appears on diffuse surfaces, either real or virtual. Our approach does not rely on physically correct assumptions. This means that the incoming photon energy is only a crude approximation, although the difference between an estimated caustic and a realistic caustic is mostly hard to detect for the human eye. The aim was to generate perceptually plausible caustics, however, an improved caustic algorithm should build upon physically correct assumptions, particularly in a mixed-reality environment that tries to imitate the real lighting conditions.

To our knowledge, the presented method in this thesis is the first rasterization-based approach that covers all these light effects from reflective and refractive objects in its entirety and is also able to fully handle the user interaction in a mixed-reality application. Our framework handles various lighting situations for real and virtual objects and light sources. This offers a practicable possibility that greatly improves the visual quality of a mixed-reality environment.

APPENDIX A

Light Paths for the Back-Projection

This chapter shows all possible light paths that occur for reflective and refractive objects in a mixed-reality environment. These paths help to identify the initial situation and to define the composition of the final output image.

A.1 Light Paths for Reflection

This part contains the light paths for reflective objects. Figure A.2 illustrates the including components. All components, except the camera, can be real or virtual. The camera (E) sees the visible point (p) on the reflective object (S) and also the reflected point (p_{refl}) on object (D).

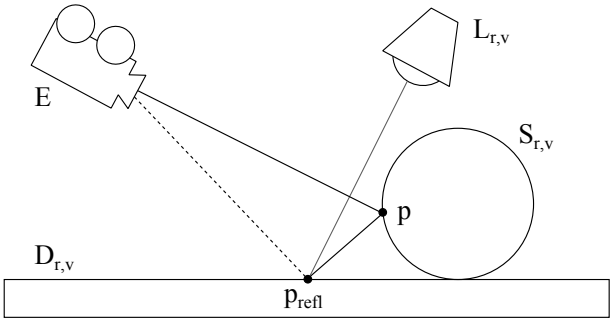


Figure A.1: Light path for reflections. Contributing components are the light source (L), the reflective object (S), the reflected object (D) and the camera (E).

Table A.1 specifies the composition of the final image L_f with differential rendering. All components in the light path, like the light source (L), the reflected surface (D), and the reflective surface (S) can be real (r) or virtual (v). Note that for multiple reflections we set the reflective object (S) to real when all light bounces happen on real surfaces. Consequently this flag is virtual (v) when only one surface is virtual. The function $L_b(x)$ takes a surface location (p or p_{refl})

and returns the corresponding value in the captured camera image, a one in the column indicates that this value is used. The remaining columns L_{rv}, L_r, and L_f define the differential rendering contribution.

Nr	L	D	S	$L_b(p)$	$L_b(p_{refl})$	L_{rv}	L_r	L_f
1	r	r	r	1	0	L_{refl}	L_{refl}	$L_b(p)$
2	r	v	r	0	0	L_{refl}	0	L_{refl}
3	v	r	r	1	0	L_{refl}	0	$L_b(p) + L_{refl}$
4	v	v	r	0	0	L_{refl}	0	L_{refl}
5	r	r	v	0	1	L_{refl}	L_{refl}	$L_b(p_{refl})$
6	r	v	v	0	0	L_{refl}	0	L_{refl}
7	v	r	v	0	1	L_{refl}	0	$L_b(p_{refl}) + L_{refl}$
8	v	v	v	0	0	L_{refl}	0	L_{refl}

Table A.1: Light paths for reflection according to differential rendering.

We observe from Table A.1 that the real-virtual flag of the light source has no influence on the usage of L_b. Moreover, the original value $L_b(p)$ from the camera image is used when the reflected object and the reflective objects are real (row one and three):

$$D_{refl} \quad \wedge \quad S \tag{A.1}$$

, where the flag real (r) maps to true and virtual (v) to false. The back-projected value $L_b(p_{refl})$ is used for L_b when the visible reflected object is real and the reflective object is virtual (row five and seven):

$$D_{refl} \quad \wedge \quad \neg S \tag{A.2}$$

, where the flag real (r) maps to true and virtual (v) to false.

Utilizing the helper functions $r(x)$ and $r(x, y)$, which maps a real surface or a path between two surface points to true, we can rewrite the back-projection as:

$$L'_{bp}(p, p_{refl}) = \begin{cases} L_b(p) & \text{if} \quad r(p, p_{refl}) \wedge r(p_{refl}) \\ L'_b(p_{refl}) & \text{otherwise} \end{cases} \tag{A.3}$$

$$L'_b(x) = \begin{cases} L_b(x) & \text{if} \quad r(x) \\ 0 & \text{otherwise} \end{cases} \tag{A.4}$$

A.2 Light Paths for Reflection and Refraction

This part contains the light paths for reflective and refractive objects. Figure A.2 shows the including components. All components, except the camera, can be real or virtual. The camera (E) sees the visible point (p) on the refractive object (S), the reflected point (p_{refl}) and the refracted point (p_{refr}) on the diffuse objects (D).

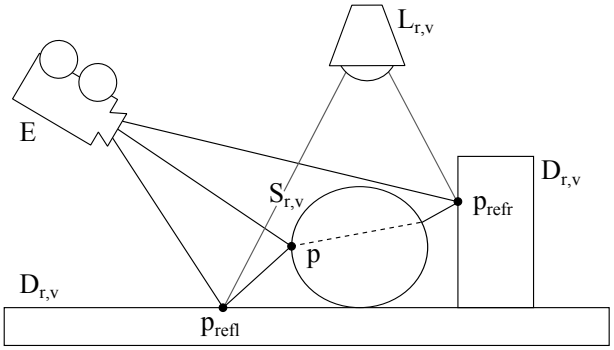


Figure A.2: Light path for reflections and refractions. Contributing components are the light source (L), the reflective object (S), the reflected object (D), the refracted object (D), and the camera (E).

Table A.2 specifies the composition of the final image L_f with differential rendering. All components in the light path, like the light source (L), the reflected surface (D_{refl}), the refracted surface D_{refr}, and the refractive surface (S) can be real (r) or virtual (v). The function ($L_b(x)$) takes surface locations (p, p_{refl}, or p_{refr}) and returns the corresponding value in the captured camera image, a one indicates that this value is used. The remaining columns L_{rv}, L_r, and L_f define the differential rendering contribution. For reasons of clarity we omit the Fresnel factors in the column header. This means that $L_b(p_{refl})$, $L_b(p_{refr})$, L_{rv}, and L_r are implicitly weighted by the Fresnel term and transmittance color filter.

As mentioned before, the real-virtual flag of the light source has no effect on the usage of L_b. From Table A.2 follows that the original value $L_b(p)$ is used for L_b when all participating surfaces are real (row one and five):

$$D_{refl} \quad \wedge \quad D_{refr} \quad \wedge \quad S \tag{A.5}$$

, where the flag real (r) maps to $true$ and virtual (v) to $false$.

The back-projection of the reflected surface point depends on the set of real-virtual combinations that evaluates to one for $L_b(p_{refl})$. Identfiying the different cases results in:

$$\begin{array}{lllllll} D_{refl} & \wedge & \neg D_{refr} & \wedge & S & \vee \\ D_{refl} & \wedge & D_{refr} & \wedge & \neg S & \vee \\ D_{refl} & \wedge & \neg D_{refr} & \wedge & \neg S & \end{array} \tag{A.6}$$

, where the flag real (r) maps to $true$ and virtual (v) to $false$. Simplifying this *disjunctive normal form* utilizing Boolean algebra results in:

$$\begin{array}{lllll} D_{refl} & \wedge & \neg D_{refr} & \vee \\ D_{refl} & \wedge & \neg S & \end{array} \tag{A.7}$$

, so $L_b(p_{refl})$ is used for L_b, when the reflected object is real and one of the refracted or the reflective surfaces is virtual.

Nr	L	D_{refl}	D_{refr}	S	$L_b(p)$	$L_b(p_{refl})$	$L_b(p_{refr})$	L_{rv}	L_r	L_f
1	r	r	r	r	1	0	0	$L_{refl} + L_{refr}$	$L_{refl} + L_{refr}$	$L_b(p)$
2	r	r	v	r	0	1	0	$L_{refl} + L_{refr}$	L_{refl}	$L_b(p_{refl}) + L_{refr}$
3	r	v	r	r	0	0	1	$L_{refl} + L_{refr}$	L_{refr}	$L_b(p_{refr}) + L_{refl}$
4	r	v	v	r	0	0	0	$L_{refl} + L_{refr}$	0	$L_{refl} + L_{refr}$
5	v	r	r	r	1	0	0	$L_{refl} + L_{refr}$	0	$L_b(p) + L_{refl} + L_{refr}$
6	v	r	v	r	0	1	0	$L_{refl} + L_{refr}$	0	$L_b(p_{refl}) + L_{refl} + L_{refr}$
7	v	v	r	r	0	0	1	$L_{refl} + L_{refr}$	0	$L_b(p_{refr}) + L_{refl} + L_{refr}$
8	v	v	v	r	0	0	0	$L_{refl} + L_{refr}$	0	$L_{refl} + L_{refr}$
9	r	r	r	v	0	1	1	$L_{refl} + L_{refr}$	$L_{refl} + L_{refr}$	$L_b(p_{refl}) + L_b(p_{refr})$
10	r	r	v	v	0	1	0	$L_{refl} + L_{refr}$	L_{refl}	$L_b(p_{refl}) + L_{refr}$
11	r	v	r	v	0	0	1	$L_{refl} + L_{refr}$	L_{refr}	$L_b(p_{refr}) + L_{refl}$
12	r	v	v	v	0	0	0	$L_{refl} + L_{refr}$	0	$L_{refl} + L_{refr}$
13	v	r	r	v	0	1	1	$L_{refl} + L_{refr}$	0	$L_b(p_{refl}) + L_b(p_{refr})$ $+L_{refl} + L_{refr}$
14	v	r	v	v	0	1	0	$L_{refl} + L_{refr}$	0	$L_b(p_{refl}) + L_{refl} + L_{refr}$
15	v	v	r	v	0	0	1	$L_{refl} + L_{refr}$	0	$L_b(p_{refr}) + L_{refl} + L_{refr}$
16	v	v	v	v	0	0	0	$L_{refl} + L_{refr}$	0	$L_{refl} + L_{refr}$

Table A.2: Light paths for reflection and refraction according to differential rendering.

For the refracted object, a similar approach results in:

$$\begin{array}{lll} D_{refr} & \wedge & \neg D_{refl} \quad \vee \\ D_{refr} & \wedge & \neg S \end{array} \tag{A.8}$$

, where $L_b(p_{refr})$ is used for L_b, when the refracted object is real and one of the reflected or the reflective surfaces is virtual.

Since our system also supports multiple reflections and refractions, we need to evaluate a reflective or refractive path instead of only considering the refractive object S. Therefore we introduce the function $r(x, y)$ that takes the refractive object as argument x and the diffuse surface hit as argument y, which is either a reflected or refracted surface. It returns true when all reflective or refractive surfaces along this path are real. Otherwise, it returns false when only one light bounce happens on a virtual surface.

In combination with the other helper function $r(x)$, we can rewrite Equation A.5, now accounting for multiple reflections and refractions, as:

$$r(p, p_{refl}) \wedge r(p, p_{refr}) \wedge r(p_{refl}) \wedge r(p_{refr}) \tag{A.9}$$

, rearrange Equation A.7 as:

$$r(p_{refl}) \wedge (\neg r(p, p_{refl}) \vee \neg r(p_{refr})) \tag{A.10}$$

and Equation A.8 as:

$$r(p_{refr}) \wedge (\neg r(p, p_{refr}) \vee \neg r(p_{refl})) \tag{A.11}$$

The usage of the original value in the camera image and the back-projected camera values is mutually exclusive. Either we use $L_b(p)$ or a combination of $L_b(p_{refl})$ and $L_b(p_{refr})$. Formally this is written as:

$$L'_{bp}(p, p_{refl}, p_{refr}) = \begin{cases} L_b(p) & \text{if} \quad \begin{array}{c} r(p, p_{refl}) \wedge r(p, p_{refr}) \wedge \\ r(p_{refl}) \wedge r(p_{refr}) \end{array} \\ L'_b(p_{refl}) + L'_b(p_{refr}) & \text{otherwise} \end{cases} \tag{A.12}$$

$$L'_b(x) = \begin{cases} L_b(x) & \text{if} \quad r(x) \\ 0 & \text{otherwise} \end{cases} \tag{A.13}$$

A.3 Light Paths for Caustics

This part contains the light paths for caustics. Figure A.3 illustrates the including components. All components, except the camera, can be real or virtual. The photon is emmited from the light source (L) and is scattered on the reflective or refractive object (S). The camera (E) sees the photon hit point (p_{hit}) on the diffuse surface object (D).

Table A.3 specifies the contribution to the differential rendering buffers L_{rv} and L_r. All components in the light path, like the light source (L), the diffuse surface (D), and the reflective or refractive surface (S) can be real (r) or virtual (v).

Table A.3 shows that the incoming radiance on a diffuse surface is only added to the real differential buffer L_r when all contributing components are real.

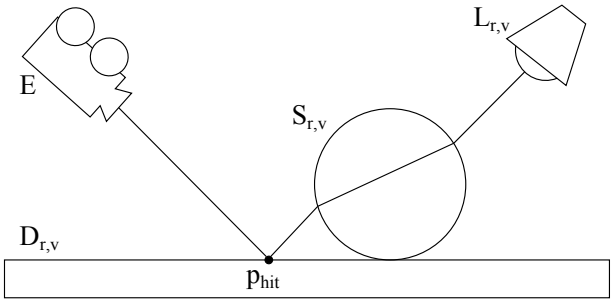


Figure A.3: Light path for caustics. Contributing components are the light source (L), the reflective or refractive object (S), the diffuse object (D) and the camera (E).

Nr	L	D	S	L_{rv}	L_r
1	r	r	r	L_{photon}	L_{photon}
2	r	v	r	L_{photon}	0
3	v	r	r	L_{photon}	0
4	v	v	r	L_{photon}	0
5	r	r	v	L_{photon}	0
6	r	v	v	L_{photon}	0
7	v	r	v	L_{photon}	0
8	v	v	v	L_{photon}	0

Table A.3: Light paths for caustics according to differential rendering.

Bibliography

[1] Akenine-Möller, Tomas, Eric Haines, and Natty Hoffman. *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008.

[2] Appshaker Ltd: www.appshaker.co.uk. Live Augmented Reality for National Geographic Channel / UPC. http://appshaker.co.uk/national-geographic-channel-2/, 2011. Accessed on Jannuary 27, 2013.

[3] Arvo, James. Backward Ray Tracing. In *In ACM SIGGRAPH '86 Course Notes - Developments in Ray Tracing*, pages 259–263. 1986.

[4] Bjorke, Kevin. Image-Based Lighting. In Randima Fernando, editor, *GPU Gems*, pages 307–321. Addison-Wesley, 2004.

[5] Blinn, James F. and Martin E. Newell. Texture and Reflection in Computer Generated Images. *Commun. ACM*, 19(10):542–547, October 1976.

[6] Dachsbacher, Carsten and Marc Stamminger. Reflective Shadow Maps. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, I3D '05, pages 203–231. ACM, New York, NY, USA, 2005.

[7] Davis, Scott T and Chris Wyman. Interactive Refractions with Total Internal Reflection. In *Proceedings of Graphics Interface 2007*, GI '07, pages 185–190. ACM, New York, NY, USA, 2007.

[8] Debevec, Paul. Rendering Synthetic Objects into Real Scenes: Bridging Traditional and Image-Based Graphics with Global Illumination and High Dynamic Range Photography. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 189–198. ACM, New York, NY, USA, 1998.

[9] Dutré, Philip, Kavita Bala, and Philippe Bekaert. *Advanced Global Illumination, 2nd edition*. A K Peters (http://www.akpeters.com/), 2006.

[10] Greene, Ned. Environment Mapping and Other Applications of World Projections. *IEEE Comput. Graph. Appl.*, 6(11):21–29, November 1986.

[11] Gringer. Linear visible spectrum.svg. Wikipedia - The Free Encylopedia, 2008.

[12] Grosch, T. Differential Photon Mapping: Consistent Augmentation of Photographs with Correction of all Light Paths. In *Eurographics 2005 Short Papers, Trinity College, Dublin, Ireland*. 2005.

[13] Heckbert, Paul. Derivation of Refraction Formulas. In Andrew Glassner, editor, *Introduction to Ray Tracing*, pages 288–293. Academic Press, London, 1989.

[14] Heckbert, Paul S. Adaptive Radiosity Textures for Bidirectional Ray Tracing. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '90, pages 145–154. ACM, New York, NY, USA, 1990.

[15] Horn, Martin. Der Blick in den Körper – Erweiterte Realität in der computergestützten Chirurgie. Faculty of Informatic - Munich University of Technology http://www.in.tum.de/forschung/forschungs-highlights/medical-augmented-reality.html, 2009-2013. Accessed on Jannuary 27, 2013.

[16] Hu, Wei and Kaihuai Qin. Interactive Approximate Rendering of Reflections, Refractions, and Caustics. *IEEE Transactions on Visualization and Computer Graphics*, 13(1):46–57, January 2007.

[17] Jensen, Henrik Wann. Global Illumination Using Photon Maps. In *Proceedings of the eurographics workshop on Rendering techniques '96*, pages 21–30. Springer-Verlag, London, UK, UK, 1996.

[18] Kajiya, James T. The Rendering Equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '86, pages 143–150. ACM, New York, NY, USA, 1986.

[19] Kán, Peter and Hannes Kaufmann. High-Quality Reflections, Refractions, and Caustics in Augmented Reality and their Contribution to Visual Coherence. In *Proceedings of International Symposium on Mixed and Augmented Reality (ISMAR)*. ACM Press, 2012. Zur Veröffentlichung angenommen.

[20] Kaufmann, Hannes. *Geometry Education with Augmented Reality*. Ph.D. thesis, Institut für Softwartechnik und Interaktive Systeme, 2004.

[21] Keller, Alexander. Instant Radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 49–56. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1997.

[22] Knecht, Martin, Christoph Traxler, Oliver Mattausch, Werner Purgathofer, and Michael Wimmer. Differential Instant Radiosity for Mixed Reality. In *Proceedings of the 2010 IEEE International Symposium on Mixed and Augmented Reality (ISMAR 2010)*, pages 99–107. October 2010. Best Paper Award!

[23] Knecht, Martin, Christoph Traxler, Oliver Mattausch, and Michael Wimmer. Reciprocal Shading for Mixed Reality. *Computers & Graphics*, 36(7):846–856, November 2012.

[24] Laboratory, Christian Doppler. Studierstube Tracker. In Christian Doppler Laboratory for Handheld Augmented Reality at Graz University of Technology: http://handheldar.icg.tugraz.at/stbtracker.php. Accessed on Jannuary 27, 2013.

[25] McReynolds, Tom and David Blythe. Advanced Graphics Programming Techniques Using OpenGL. SIGGRAPH 98 Course Notes, April 1998.

[26] metaio GmbH. Lego Digital Box Augmented Reality Kiosk. metaio GmbH - http://www.metaio.com/solutions/kiosk/lego/, 2013. Accessed on Jannuary 27, 2013.

[27] Microsoft. Cubic Environment Mapping. In the Programming Guide for Direct3D 9: http://msdn.microsoft.com/en-us/library/windows/desktop/bb2048812012. Accessed on Jannuary 27, 2013.

[28] Microsoft. DirectX Software Development Kit, 2012. Accessed on Jannuary 27, 2013.

[29] Milgram, P. and F. Kishino. A Taxonomy of Mixed Reality Visual Displays. *IEICE Trans. Information Systems*, E77-D(12):1321–1329, December 1994.

[30] Oliveira, Manuel M. and Maicon Brauwers. Real-time Refraction Through Deformable Objects. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, I3D '07, pages 89–96. ACM, New York, NY, USA, 2007.

[31] Pessoa, Saulo, Guilherme Moura, Joao Lima, Veronica Teichrieb, and Judith Kelner. Photorealistic Rendering for Augmented Reality: A Global Illumination and BRDF Solution. In *Proceedings of the 2010 IEEE Virtual Reality Conference*, VR '10, pages 3–10. IEEE Computer Society, Washington, DC, USA, 2010.

[32] Pharr, Matt and Greg Humphreys. *Physically Based Rendering, First Edition: From Theory To Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[33] Pirk, Sören. *GPU-Based Rendering of Reflective and Refractive Objects in Augmented Reality Environments*. Master's thesis, Fachhochschule Oldenburg/Ostfriesland/Wilhelmshaven, 2007.

[34] Policarpo, Fábio, Manuel M. Oliveira, and João L. D. Comba. Real-time Relief Mapping on Arbitrary Polygonal Surfaces. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, I3D '05, pages 155–162. ACM, New York, NY, USA, 2005.

[35] Popescu, Voicu, Chunhui Mei, Jordan Dauble, and Elisha Sacks. Reflected-Scene Impostors for Realistic Reflections at Interactive Rates. *Comput. Graph. Forum*, pages 313–322, 2006.

[36] Ritschel, T., T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz. Imperfect Shadow Maps for Efficient Computation of Indirect Illumination. *ACM Trans. Graph.*, 27(5):129:1–129:8, December 2008.

[37] Rosen, Paul, Voicu Popescu, Kyle Hayward, and Chris Wyman. Nonpinhole Approximations for Interactive Rendering. *IEEE Comput. Graph. Appl.*, 31(6):68–83, November 2011.

[38] Schlick, Christophe. An Inexpensive BRDF Model for Physically-Based Rendering. *Comput. Graph. Forum*, 13(3):233–246, 1994.

[39] Shah, Musawir A., Jaakko Konttinen, and Sumanta Pattanaik. Caustics Mapping: An Image-Space Technique for Real-Time Caustics. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):272–280, March 2007.

[40] State, Andrei, Gentaro Hirota, David T. Chen, William F. Garrett, and Mark A. Livingston. Superior Augmented Reality Registration by Integrating Landmark Tracking and Magnetic Tracking. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 429–438. ACM, New York, NY, USA, 1996.

[41] Szirmay-Kalos, László, Barnabás Aszódi, István Lazányi, and Mátyás Premecz. Approximate Ray-Tracing on the GPU with Distance Impostors. *Computer Graphics Forum*, 24(3):685–704, 2005.

[42] Traxler, Christoph. Reciprocal Shading for Mixed Reality. In The Institute of Computer Graphics and Algorithms at Vienna University of Technology: http://www.cg.tuwien.ac.at/projects/RESHADE/, 2009-2013. Accessed on Jannuary 27, 2013.

[43] Umenhoffer, Tamás, Gustavo Patow, and László Szirmay-Kalos. Caustic Triangles on the GPU. In *Proceedings of Computer Graphics International 2008*. 2008.

[44] Umenhoffer, Tamás, Gustavo Patow, and László Szirmay-Kalos. Robust Multiple Specular Reflections and Refractions. In Hubert Nguyen, editor, *GPU Gems 3*, pages 387–407. Addison-Wesley, 2008.

[45] ViewAR - Augmented Reality Catalog System. Augmented Reality Furniture. http://www.viewar.com, 2013. Accessed on Jannuary 27, 2013.

[46] Wyman, Chris. An Approximate Image-Space Approach for Interactive Refraction. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 1050–1053. ACM, New York, NY, USA, 2005.

[47] Wyman, Chris. Interactive Image-Space Refraction of Nearby Geometry. In *Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, GRAPHITE '05, pages 205–211. ACM, New York, NY, USA, 2005.

[48] Wyman, Chris. Hierarchical Caustic Maps. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games*, I3D '08, pages 163–171. ACM, New York, NY, USA, 2008.

[49] Wyman, Chris and Scott Davis. Interactive Image-Space Techniques for Approximating Caustics. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, I3D '06, pages 153–160. ACM, New York, NY, USA, 2006.

[50] Wyman, Chris and Greg Nichols. Adaptive Caustic Maps Using Deferred Shading. *Comput. Graph. Forum*, 28(2):309–318, 2009.