FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

# Reciprocal Shading For Mixed Reality

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

**Doktor der technischen Wissenschaften**

by

**Martin Knecht**
Registration Number 0326294

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn Michael Wimmer

The dissertation has been reviewed by:

| | |
|---|---|
| (Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn Michael Wimmer) | (Prof. Dr. Mark Billinghurst) |

Wien, 15.10.2013

(Martin Knecht)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

# Reciprocal Shading For Mixed Reality

DISSERTATION

zur Erlangung des akademischen Grades

**Doktor der technischen Wissenschaften**

eingereicht von

**Martin Knecht**
Matrikelnummer 0326294

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn Michael Wimmer

Diese Dissertation haben begutachtet:

| | |
|---|---|
| (Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn Michael Wimmer) | (Prof. Dr. Mark Billinghurst) |

Wien, 15.10.2013

(Martin Knecht)

# Erklärung zur Verfassung der Arbeit

Martin Knecht
Mottaweg 72, 6822 Röns

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

Although it is only my name that is written on the cover of this thesis, there are several people who put a lot of time and hard work into the presented contributions.

First, I would like to thank Michael Wimmer and Christoph Traxler for supervising this thesis and for their incredible support over the last four years.

Furthermore, I would like to thank Werner Purgathofer and Michael Gervautz for their support throughout the RESHADE project and for giving me the chance to work in the office of Imagination GmbH. Moreover, I would like to thank the team at the Imagination for the great work environment, the fun and the adventures we shared. For inspiring discussions and helpful feedback I would like to express my gratefulness to my colleagues at the Institute of Computer Graphics and Algorithms. A special thank goes to Matthias Bernhard for his great help and support concerning the user study and to Reinhold Preiner for the relaxing coffee breaks.

During my Ph.D., I got the chance to do research at the Human Interface Technology Laboratory in Christchurch, New Zealand. I greatly thank Mark Billinghurst for giving me this opportunity and also for being the second reviewer of this thesis. I had an amazing time there and would like to thank Raphael Grasset, Andreas Dünser and the whole team at HITLab NZ for this great experience. Many thanks also go to Jiri Bittner from the Czech Technical University in Prague where I spent three weeks doing research under his supervision.

The students Adam Celarek, Klemens Jahrmann, Martina Rasch and Stefan Spelitz added several valuable features to our software framework – thank you very much. Especially I want to thank Georg Tanzmeister and Christoph Winklhofer, whose master theses contributed a lot to my thesis.

Thanks go to the FFG Austria Research Promotion Agency, who funded the RESHADE project under the „FIT-IT Visual Computing“ program (project nr. 820916).

Finally, for me writing this thesis was like a roller coaster ride with several awesome highs but also some downs. However, it is much easier to go through and recover from tough times when there are amazing people around you. I'm deeply grateful to my family, friends and Lea for their incredible help, support and patience over the last years! Thank you!

# Abstract

Reciprocal shading for mixed reality aims to integrate virtual objects into real environments in a way that they are in the ideal case indistinguishable from real objects. It is therefore an attractive technology for architectural visualizations, product visualizations and for cultural heritage sites, where virtual objects should be seamlessly merged with real ones. Due to the improved performance of recent graphics hardware, real-time global illumination algorithms are feasible for mixed-reality applications, and thus more and more researchers address realistic rendering for mixed reality.

The goal of this thesis is to provide algorithms which improve the visual plausibility of virtual objects in mixed-reality applications. Our contributions are as follows:

First, we present five methods to reconstruct the real surrounding environment. In particular, we present two methods for geometry reconstruction, a method for material estimation at interactive frame rates and two methods to reconstruct the color mapping characteristics of the video see-through camera.

Second, we present two methods to improve the visual appearance of virtual objects. The first, called differential instant radiosity, combines differential rendering with a global illumination method called instant radiosity to simulate reciprocal shading effects such as shadowing and indirect illumination between real and virtual objects. The second method focuses on the visual plausible rendering of reflective and refractive objects. The high-frequency lighting effects caused by these objects are also simulated with our method.

The third part of this thesis presents two user studies which evaluate the influence of the presented rendering methods on human perception. The first user study measured task performance with respect to the rendering mode, and the second user study was set up as a web survey where participants had to choose which of two presented images, showing mixed-reality scenes, they preferred.

# Kurzfassung

Reziproke Schattierung für erweiterte Realitäten zielt darauf ab, virtuelle Objekte so in eine reale Umgebung zu integrieren, dass diese im Idealfall nicht von realen Objekten zu unterscheiden sind. Deshalb ist reziproke Schattierung für erweiterte Realitäten eine attraktive Technologie für Architekturvisualisierungen, Produktvisualisierungen oder für Kulturstätten, bei denen virtuelle Objekte nahtlos in die reale Umgebung eingebettet werden sollen. Aufgrund der erhöhten Leistung heutiger Graphikhardware sind nun auch globale Beleuchtungsmethoden für Echtzeitanwendungen möglich, wodurch das wissenschaftliche Interesse an realistischen Darstellungen für erweiterte Realitäten gestiegen ist.

Das Ziel dieser Dissertation ist es die visuelle Plausibilität virtueller Objekte für Anwendungen im Bereich der erweiterten Realität zu erhöhen. Folgende Beiträge werden in dieser Arbeit präsentiert:

Der erste Abschnitt widmet sich fünf Methoden, die zum Ziel haben die reale Umgebung zu rekonstruieren. Zwei dieser Methoden befassen sich mit der Rekonstruierung der realen Geometrie, eine weitere Methode dient der Schätzung von Materialeigenschaften bei Beibehaltung interaktiver Bildraten. Die verbleibenden beiden Methoden rekonstruieren das Farbabbildungsverhalten der Kamera, über deren Bilder die virtuellen Objekte eingeblendet werden.

Der zweite Abschnitt beschreibt zwei Methoden, um die visuelle Plausibilität virtueller Objekte zu verbessern. Die erste Methode nennt sich „differential instant radiosity“, welche „differential rendering“ und die globale Beleuchtungsmethode „instant radiosity“ kombiniert, um reziprokale Schattierungseffekte, wie Schatten und indirekte Beleuchtung zwischen realen und virtuellen Objekten zu simulieren. Die zweite Methode beschäftigt sich mit der visuell plausiblen Darstellung reflektierender und refraktierender Objekte. Die bei solchen Objekten entstehenden hochfrequenten Lichteffekte werden mit dieser Methode ebenfalls simuliert.

Der dritte Teil der Dissertation präsentiert zwei Benutzerstudien, welche den Einfluss verschiedener Rendering-Methoden messen. Die erste Studie untersuchte den Einfluss der Rendering-Methoden auf die Dauer mit der verschiedene Aufgabenstellungen von Testpersonen erledigt wurden. Die zweite Studie wurde in Form einer Internetumfrage gestaltet bei der Teilnehmer beurteilten, welche von zwei vorgelegten erweiterten Realitätsbildern, sie subjektiv bevorzugen.

# Contents

CHAPTER 1

# Introduction

## 1.1 Motivation

In mixed reality (MR), virtual objects and real objects are merged. Mixed reality is an attractive and exciting way to present virtual content in a real context for various application domains, like architectural visualizations, virtual prototyping, marketing and sales of not yet existing products, and edutainment systems. These kinds of application scenarios demand a believable realistic appearance of virtual objects, providing a perfect illusion for human visual perception. Unfortunately, this requirement is not met in common mixed-reality systems, where the composed images look disturbingly artificial. One major reason for this is that real illumination conditions and the mutual shading effects between virtual and real objects are completely ignored.

Such mutual shading effects are shadowing and color bleeding. If real and virtual objects reciprocally cast shadows and also cause indirect illumination, the virtual objects are photometrically registered with the surrounding real world. In this way, the human visual system (HSV) should perceive the virtual objects as if they were real ones. However, rendering these reciprocal shading effects is a challenging task when real-time or at least interactive frame rates should be reached. We therefore do not aim for physically correct rendering but instead, in this thesis we developed methods that try to render virtual objects in a visually plausible way. In the ideal case, the virtual objects should not be recognized as being virtual anymore.

## 1.2 Challenges

This section gives an overview on the challenges that we are facing when virtual objects should look visually plausible. Azuma [4] defined three characteristics that are important for mixed reality/augmented reality systems: *A combination of real and virtual objects*, *interactive in real time* and *registered in 3-D*. While the first point does not really pose a challenge, the second and third do.

**Combination of real and virtual objects:** With current hardware it is no challenge anymore to render virtual objects over a video stream. However, it is still a challenge if the real and virtual objects should be photometrically registered to each other.

**Interactive in real time:** The limited computation time for each frame is a major challenge in our context. We want visually plausible mixed reality, which implies real-time global illumination computations – in a fraction of a second. With the current hardware available, this limits the rendering quality that can be achieved. Therefore our proposed methods aim for visually plausible and not physically correct renderings. In other words, as long as the results look plausible, we can reduce the quality of the rendering methods in order to reduce computation time.

**Registration in 3-D:** The third characteristic states that virtual objects must be *registered in 3-D space*. This means that they should not float in the real world when the camera/viewpoint is moved. A lot of research therefore focuses on tracking the camera movement to seamlessly position the virtual objects in the real world. Despite the huge improvements over the last years, it is still not possible to track all different kinds of scenarios with one single method. For example, position tracking outdoors using GPS does not work in indoor scenarios. In this thesis, we do not focus on this challenge but rather use existing tracking approaches usable for our restricted scenarios.

**Environment reconstruction:** In order to simulate the *mutual light interaction* between real and virtual objects, it is necessary to have knowledge about the *surrounding environment* in terms of incident light, the geometry of the real objects and their material properties. Furthermore, the environment might change over time and therefore they should be reconstructed during runtime. However, there is no sensor available that is able to capture this data entirely in a dynamic environment, and therefore another challenge is how to reconstruct this information and furthermore how to deal with missing data.

In our thesis we focus on video see-through mixed reality. This means that a camera films the real scene and then virtual objects are rendered on top of the video stream. If necessary, a head-mounted display could then show the enhanced video image. In contrast to that are optical see-through systems where virtual objects are projected onto semi-transparent screens. In this way, they are merged with the real environment seen through the screens. With video see-through mixed reality, the camera always also introduces its very own mapping from real-world radiance values to RGB color triples. This mapping includes lens distortions, noise and a couple of other artifacts – which virtual objects do not show, because they are rendered. However, the perfect rendering of the virtual objects reduces their plausibility in the real world and therefore the camera characteristics have to be reconstructed and simulated as well.

**Evaluation** Since we aim for visually plausible mixed reality, the human visual system is the assessment criteria which decides whether our proposed methods render virtual objects better or worse. This raises the challenge, how to set up experiments that measure the quality of our methods.

## 1.3 Dissertation Thesis

This thesis focuses mainly on the reconstruction of the environment and the mutual shading effects between real and virtual objects to enable visually plausible mixed reality. The main thesis is that it is possible to render virtual objects in a visually plausible way for mixed-reality applications by using methods proposed in this work.

## 1.4 Contributions

This thesis presents several solutions to the previously mentioned challenges. The following sections and the chapters are ordered in the same way, as information gets processed throughout an MR pipeline. The MR system first has to gain knowledge about the surrounding environment and reconstruct it. Then the virtual objects are photometrically registered within the real world by using reciprocal shading effects. Finally, the evaluation section covers results and insights of our evaluations.

### 1.4.1 Reconstruction

Chapter 4 focuses on the first stages in the MR pipeline - the reconstruction of the surrounding environment. In order to calculate correct occlusions and lighting effects between real and virtual objects, their geometry must be known. This means that the 3D shape of a real surface must be reconstructed. We proposed two methods to calculate the geometric shape of real objects using the Microsoft Kinect sensor in[2, 4].

Beside the 3D shape, the material characteristics of a real surface and the surrounding illumination need to be known too. We therefore proposed an interactive BRDF estimation approach [2] which uses the reconstructed geometry from the Microsoft Kinect sensor.

Each camera has a mapping function from real radiance values to device-dependent RGB triples. Our third contribution, described in Chapter 4, focuses on an adaptive method to match the colors of the virtual objects to the color characteristics of a video camera [5].

### 1.4.2 Reciprocal Shading

The contributions presented in Chapter 5 introduce reciprocal shading effects between real and virtual objects. Our methods perform real-time global illumination computations to simulate effects like shadows and indirect illumination between real and virtual objects [3, 4]. Furthermore, virtual light sources may be used to illuminate the real environment. Beside, opaque objects, we also propose a new method to handle real or virtual reflective or refractive objects in a general manner [6]. The presented method produces visually plausible reflections and refractions as well as real-time high-frequency light effects known as caustics.

### 1.4.3 Evaluation

The first part of Chapter 6 is based on Knecht et al. [1]. It proposes a framework for the evaluation of user studies especially in the area of visually plausible mixed reality. Furthermore, we

present a preliminary user study on task performance using different qualities of rendering. A second user study evaluates which rendering methods are preferred by the participants in terms of „How well do the virtual objects fit into the real scene?“.

### 1.4.4 References

This dissertation thesis is based on the following publications:


[1] Martin Knecht, Andreas Dünser, Christoph Traxler, Michael Wimmer, and Raphael Grasset. A framework for perceptual studies in photorealistic augmented reality. In *Proceedings of the 3rd IEEE VR 2011 Workshop on Perceptual Illusions in Virtual Environments*. Singapore, March 2011, pages 27–32.

[2] Martin Knecht, Georg Tanzmeister, Christoph Traxler, and Michael Wimmer. Interactive BRDF estimation for mixed-reality applications. *Journal of WSCG*, 20(1):47–56, June 2012.

[3] Martin Knecht, Christoph Traxler, Oliver Mattausch, Werner Purgathofer, and Michael Wimmer. Differential instant radiosity for mixed reality. In *Proceedings of the 9th IEEE International Symposium on Mixed and Augmented Reality*. In ISMAR '10. IEEE Computer Society, Seoul, South Korea, 2010, pages 99–108.

[4] Martin Knecht, Christoph Traxler, Oliver Mattausch, and Michael Wimmer. Reciprocal shading for mixed reality. *Computers & Graphics*, 36(7):846–856, November 2012.

[5] Martin Knecht, Christoph Traxler, Werner Purgathofer, and Michael Wimmer. Adaptive camera-based color mapping for mixed-reality applications. In *Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality*. In ISMAR '11. IEEE Computer Society, Washington, DC, USA, 2011, pages 165–168.

[6] Martin Knecht, Christoph Traxler, Christoph Winklhofer, and Michael Wimmer. Reflective and refractive objects for mixed reality. *IEEE Transactions on Visualization and Computer Graphics*, 19(4):576–582, April 2013.

CHAPTER 2

# Overview

The purpose of this chapter is to give an overview on how the mixed-reality pipeline is set up, and why it is done that way. Furthermore, it outlines how the thesis is organized.

As mentioned in the introduction, the goal of this thesis is to present methods which allow rendering virtual objects in a visually plausible way. In other words, the virtual objects should be photometrically registered in the real environment and to do so, two lighting effects are required: shadowing and indirect illumination.

## 2.1 Requirements

In order to photometrically register the virtual objects in the real environment, we need a method that is capable of rendering shadows and indirect illumination in a reciprocal manner. This means that the effects should be cast from real objects onto virtual ones and from virtual ones onto real ones. A fundamental solution to this problem was proposed by Debevec [22], called differential rendering (DR). It will be covered in more detail in Section 3.2.5, but in short, it is a method that calculates only the differential influence of virtual objects that are inserted into a real scene. To calculate the differential effect, two global illumination solutions are needed. One solution, $L_{rv}$, takes the real *and* virtual objects into account, and the other solution, $L_r$, takes only the real objects into account. The differential influence $\Delta L = L_{rv} - L_r$ only contains changes in the illumination caused by the virtual objects. If this differential buffer is added to the video image, which is masked to black where virtual objects appear, the virtual objects and all its reciprocal effects between the real and the virtual objects are added to the camera image. Wherever there is no influence of the virtual objects, the camera image will stay the same. Wherever there is a shadow or indirect illumination from a virtual object, the camera image will be darker or respectively brighter and therefore, it will look like the real objects receive shadows or indirect illumination from virtual objects. Note that this also works the other way round and therefore also applies for virtual objects.

However, since this method needs two global illumination solutions, it was not usable for real-time mixed-reality applications at the time of publishing because the frame rates were too

low. Even nowadays, it is computationally demanding to calculate these two global illumination solutions, and methods to reduce computation costs must be applied. Another important point about differential rendering is that it needs the geometry and material characteristics of the real objects in order to calculate the two global illumination solutions. Note that also the surrounding incident illumination influences the appearance of the virtual objects and thus needs to be known too. In short, it is not enough to only have the information available from the virtual objects, which is common in typical mixed-reality applications.

There are basically two ways to get the needed information about the real environment. First, one can model the real objects in a preprocessing step and capture a static environment map in advance. However, this is very time consuming and also limits the dynamic behavior of the real scene. The second option is to reconstruct the information about the real objects and the incident illumination during runtime. This reconstruction step is a research problem unto itself and therefore we split the mixed-reality pipeline into two main parts. The *reconstruction* stage and the *reciprocal shading* stage.

## 2.2 The Mixed-Reality Pipeline

Kruijff et al. [72] introduced a taxonomy of the main perceptual issues in augmented reality. In this taxonomy, they also propose a perceptual pipeline which consists of five stages: *Environment*, *Capturing*, *Augmentation*, *Display Device*, and finally, the *User*. If we embed our mixed-reality pipeline in their perceptual pipeline, the reconstruction stage fits into their capturing stage and the reciprocal shading stage into their augmentation stage. Figure 2.1 gives an overview of the two pipeline stages and the sub tasks that need to be performed. Two cameras are used to capture the real scene. One camera, which is shown in the middle of the scene, uses a fish-eye lens to capture the incident surrounding illumination. The second camera is the observing camera, shown on the right side. The video stream of that camera will be used to embed the virtual objects in the real environment.

### 2.2.1 The Reconstruction Stage

The reconstruction stage deals with all tasks related to gathering and generating information about the surrounding environment. Four types of data sources must be generated in the reconstruction stage: Knowledge about the surrounding illumination, the geometric shapes of the real objects, its material characteristics, and a fourth source, camera characteristics, which we have not discussed yet.

Each camera has its very own mapping function from real-world radiance values to the device-dependent RGB color space. The virtual objects, however, are normally only rendered on top of the video stream and are therefore not exposed to this mapping function. If we want the colors of the virtual objects to appear as if they were seen through the camera, this mapping function must be known. Therefore, the fourth data source is this color-mapping function, which will be applied to the virtual objects later.

The first three types of data sources are needed to compute the two global illumination solutions $L_{rv}$ and $L_r$ in the second pipeline stage. In our case we use the Microsoft Kinect

sensor [90] to gather information about the real environment. The fourth data source will be used to change the color appearance of the virtual objects according to the observing camera before differential rendering is applied. All methods that generate these data sources are presented in Chapter 4.

Note that if the real scene is premodelled, the subtasks position and normal estimation as well as the material estimation can be skipped.

### 2.2.2 The Reciprocal Shading Stage

The reciprocal shading stage takes the gathered information from the reconstruction stage to calculate the two global illumination solutions in real time. In Chapter 5, the first method focuses on a combination of differential rendering [22] with instant radiosity [60] to obtain these two global illumination solutions, $L_{rv}$ and $L_r$, at real-time frame rates. It also applies the color-mapping function and performs differential rendering to generate the final image, as shown in Figure 2.1. The second presented method adds support for reflective or refractive objects in mixed-reality applications.

An important subtask of the reciprocal shading stage is to generate a so-called geometry buffer (G-Buffer). This buffer stores all information necessary to shade a pixel. Thus it contains the 3D position, the normal, and the material parameters of each point visible from the observing camera. The preprocessing for the reciprocal shading covers a couple of other subtasks, such as virtual point light creation (see Section 3.2.2) or imperfect shadow map creation (see Section 3.2.3). The results after the indirect illumination shading task are the two global illumination buffers $L_{rv}$ and $L_r$. The postprocessing task adds direct illumination, applies the color-mapping function, and calculates the differential influence using the two buffers. In the last step of the pipeline, the differential influence is added to the masked camera image to produce the final output.

Please note that although all tasks are mentioned in the pipeline (see Figure 2.1), they do not necessarily work seamlessly together. For example, if the material parameters are estimated, it is not possible to also perform the color-mapping function estimation for the camera because the material estimation does not take this color-mapping function into account.

## 2.3 Further Reading

The next chapter serves as an additional resource of background information and related work. It is organized according to the follow-up chapters and introduces different reconstruction and real-time global illumination rendering methods that are related to our work. Furthermore, the bidirectional reflectance distribution function [100], the rendering equation [53], instant radiosity [60], imperfect shadow maps [117], and finally, the differential rendering method [22] are presented. The last section deals with related work of studies on perception and photorealism. References for further reading and in-depth reviews are presented at the end of each section.

Chapter 4 presents all methods which belong to the reconstruction stage in the mixed-reality pipeline. Chapter 5 presents the two mentioned methods to add virtual objects into a real environment in a visually plausible way. Chapter 6 covers two user studies that we performed to

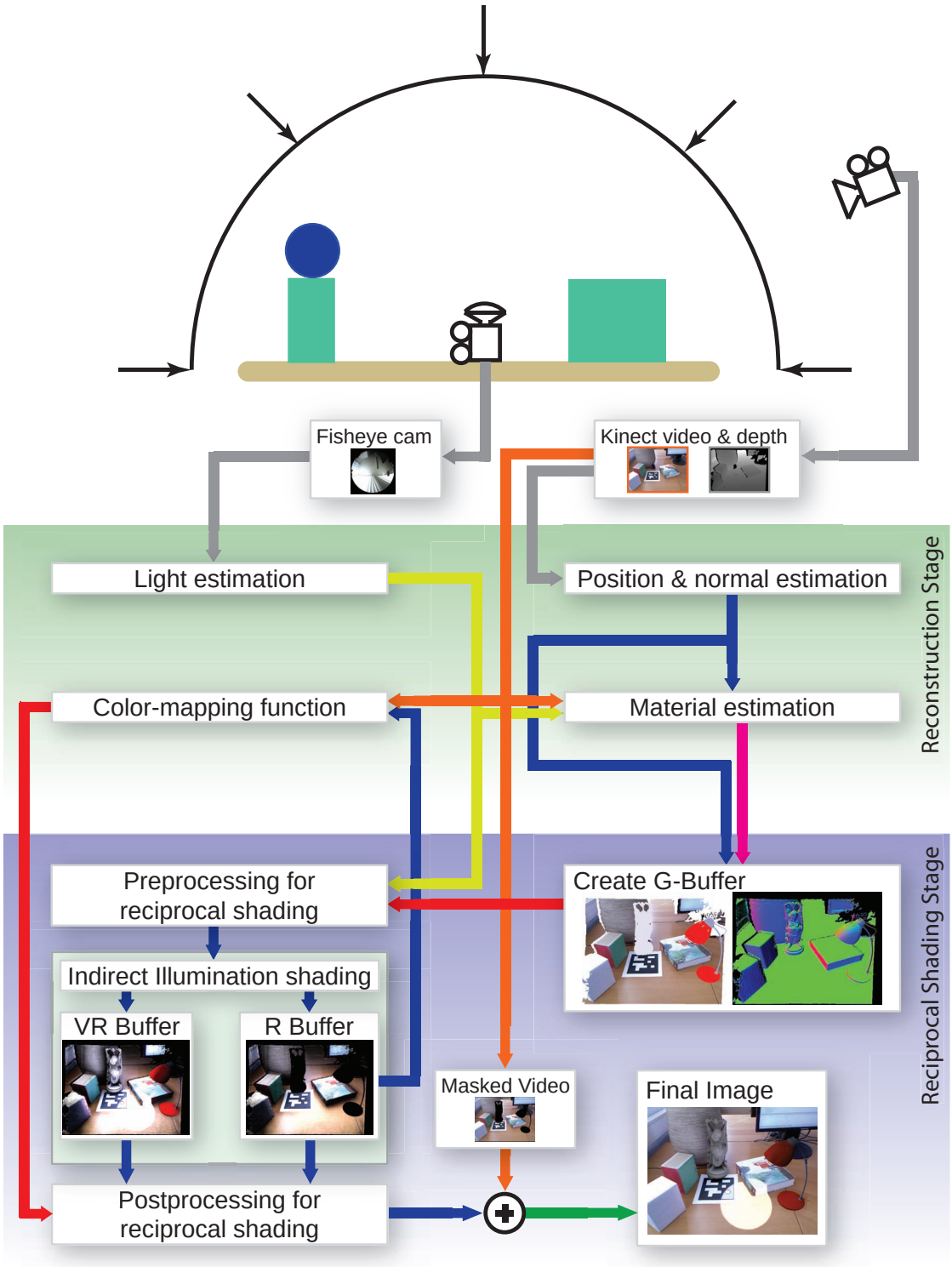



**Figure 2.1:** This figure shows the main pipeline stages *reconstruction* and *reciprocal shading* with its sub tasks.

find out whether our methods have an influence on the perceived quality or task performance. The thesis concludes with Chapter 7 and also gives an outlook on future research directions in visually plausible mixed reality.

CHAPTER 3

# Background and Related Work

This chapter gives an overview on the related work of the proposed methods and also serves as a resource for background information. In analogy of the organization of the thesis, the related work chapter is divided into three main sections. Section 3.1 describes methods related to image based lighting, real-time geometry reconstruction, BRDF estimation, and finally camera characterization. Section 3.2 first concentrates on methods for real-time global illumination computation. Then methods are presented for merging real and virtual content. Afterwards, related methods for reflections and refractions as well as caustics are presented. The evaluation Section 3.3 describes several user studies that where performed in the area of visual perception with special focus on shadows and indirect illumination, as these are the main effects simulated with our methods.

## 3.1 Reconstruction

In visually plausible mixed-reality applications it is necessary to have knowledge about the real surrounding environment. In this way, it is possible to simulate mutual light interaction between real and virtual objects. One way to have the real scene's geometry is to simply pre-model it. However, this is a time consuming task and limits dynamic effects in the real scene. Therefore, real-time reconstruction techniques are the preferred way to gather information about the surrounding environment.

### 3.1.1 Light estimation and image based lighting

Most approaches that deal with illumination in mixed-reality applications use an environment map to simulate the incident illumination. For dynamic environment maps there are basically two types of methods to acquire it: outside-in and inside-out methods. Outside-in methods use a camera to take photos or a video stream of a chrome sphere. This chrome sphere reflects the surrounding scene and can be used as an environment map [22, 1, 45, 134]. The inside-out methods use a camera to capture the surrounding illumination. Ritschel and Grosch [116] used

a high dynamic range video camera to capture the surrounding illumination. Sato et al. [123] as well as Korn et al. [71] used a stereo vision inside-out approach to calculate the environmental radiance distribution and to reconstruct the environment. In most cases environment maps are in low dynamic range (LDR). However, direct sunlight is orders of magnitude brighter than indoor light sources. Therefore, it is preferable to use environment maps which have a high dynamic range (HDR) to capture all details in bright and dark areas. In this way, the illuminated objects will contain details which are otherwise lost.

Once the environment is captured, a fast method is needed to extract light sources from the environment map. The idea is to place light sources around the scene which imitate the incident illumination situation. Several methods exist to detect light sources in the environment map efficiently by some importance sampling [12, 20, 43, 21].

Beside the inside-out and outside-in methods, there are also probe-less approaches to estimate the surrounding illumination situation. In contrast to the previously mentioned methods, these do not require a disturbing camera or chrome sphere placed in the scene. Madsen and Nielsen [86] proposed a method confined to outdoor scenarios, where they estimated the surrounding illumination by taking the position and time, when an image was taken, as well as shadows of objects in the image into account. A new method, applicable for indoor scenarios, was proposed by Gruber et al. [37] in 2012. To estimate the surrounding illumination situation only the RGB color image and a depth image are needed of the real scene. With this approach they are able to achieve interactive frame rates and impressive light estimation results.

### 3.1.2 Geometry reconstruction

Geometry reconstruction is a large and very active research topic. It starts from data acquisition to semantic analysis of the reconstructed geometry. We therefore focus on related methods, which are applicable for MR systems. However, a comprehensive summary of various reconstruction methods can be found in [114].

Reisner-Kollmann [114] distinguishes between three main methods to gather geometry information from a data acquisition point of view: *Photogrammetry*, *Laser scans* and *Range images*.

*Photogrammetric* methods reconstruct geometric information from a set of images. This type of reconstruction is often used for large scale scenes, like complete cities. In these cases a plane flies over the city and an attached camera takes several images of the city. Using photogrammetric reconstruction, the buildings and streets can be reconstructed to a certain degree. For MR applications of course smaller reconstruction volumes are of interest. A photogrammetric based method was presented in 2010 by Newcombe and Davison [95]. Their method was able to reconstruct an indoor scenery of a table with only a single moving camera by using a structure from motion approach.

Another way to capture real geometry are *Laser-scanners*. These devices are able to produce a high density point cloud at a very high accuracy. However, depending on the scanned object, it is often necessary to take several scans and merge them together in a post-processing step. Even a single scan takes too long to achieve real-time frame rates.

An alternative to Laser-scanners are *Range cameras*. These cameras produce a depth image using structured light patterns or time of flight (TOF) methods. They have the benefit that they

are faster than laser scanners and therefore allow to capture dynamic scenes. TOF cameras send a light pulse and measure the time until the light gets reflected, whereas structured light patterns are projected into the scene for the other class of cameras. Cameras using structed light patterns normally have a higher lateral resolution than TOF based cameras.

The Microsoft Kinect sensor belongs to the class of structured light pattern cameras. It was released in the year 2010 and since then a large number of researchers focused on various ways to use this very cheap commodity device. The Kinect sensor is capable to deliver a color and a depth range image stream at a resolution of 640x480 pixels at 30 Hz. It uses a structured infrared light pattern to reconstruct the surrounding environment. The high frame rate enables the reconstruction of dynamic scenes. However, the depth range images are far from perfect. First, they suffer from temporal and spatial noise. Second, due to the nature of the system the accuracy decreases for objects farther away from the device. Another important point is that the infrared emitter and the infrared sensor are located in a certain distance horizontally to each other. Due to this offset, between the emitter and the sensor, some areas seen by the sensor will not receive any structured light patterns due to occluding objects. Therefore, no depth values can be obtained in these areas. Furthermore, in areas where there are depth discontinuities, on specular surfaces or in very bright situations, such as direct sunlight in outdoor scenarios, the infrared sensor is not able to see the structured light pattern and thus cannot resolve any depth values. Because of the mentioned limitations, it is not sufficient to use the raw depth range image of the input stream to reconstruct the geometry. Methods to improve and filter the results are necessary.

Such a method, called KinectFusion, was proposed by Shahram et al. [50] and Newcombe et al. [96]. It performs camera pose tracking and environment reconstruction in parallel on the GPU. The incoming depth values from the Kinect are converted into so called Truncated Signed Distance Functions (TSDFs) that are represented in a voxel volume. Voxel *outside* the geometry have positive and voxel *inside* the geometry have negative distance values. Therefore, the zero crossings in the voxel volume are points, where the surface geometry lies. In this way, the truncated signed distance values in the voxel volume are used to reconstruct a high quality 3D surface. Furthermore, the data in the volume is used to estimate the new pose of the camera for the next frame. Over time more and more data gets added into the volume and therefore, the reconstruction gets more accurate (see Figure 3.1). However, the voxel volume needs a lot of memory to be able to represent small details and therefore, the real size of the reconstructed volume is limited. Whelan et al. [145] propose a solution to this problem by moving the TSDF volume. They are able to handle the large point cloud by continuously triangulating it. In this way, scenes with a larger scale than the TSDF volume can be reconstructed. Zeng et al. [155] use an octree data structure to minimize the memory usage and therefore, are also able to reconstruct scenes with a larger extent. Meister et al. [88] performed an evaluation about the quality of the acquired data from KinectFusion and when it can be used as a ground truth reconstruction method.

Another method from Lieberknecht et al. [82] also uses the Kinect for tracking and reconstruction. In contrast to KinectFusion their method creates a polygonal mesh during runtime.

The described methods deliver a complete reconstruction once all surface parts were visible to the sensor. However, often it is sufficient to only have the geometry available which is

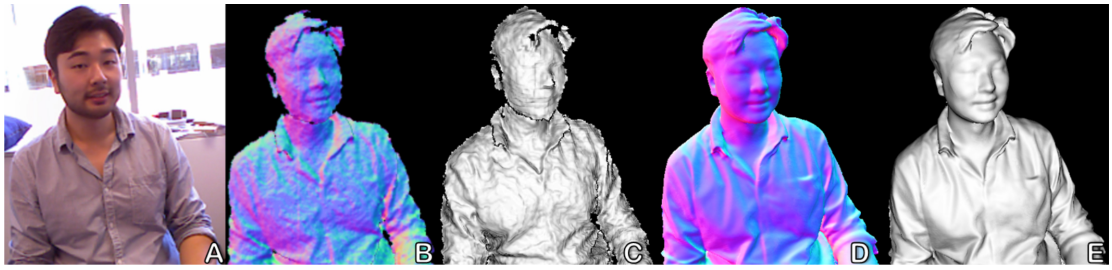



**Figure 3.1:** Image A shows the RGB input stream. Images B and C show the extracted normals and 3D mesh from a raw Kinect depth image. Images D and E illustrate the results using multiple frames with KinectFusion. ©ACM.

currently in the view frustum. Thus intelligent filtering methods which enhance the depth map quality would be enough. Lensing and Broll [77] introduced such a method. Instead of building up a volume representation over time, they enhance the quality of the raw depth map from the Kinect sensor. They first reduce undefined regions and then smooth the surface while preserving edges. To our knowledge, some parts of their algorithm are done on the CPU and they are able to reach an update rate of 11 frames per second.

A method from Kainz et al. [52] called OmniKinect uses multiple Kinects in parallel to reconstruct a given volume during runtime.

As mentioned previously, reflective or refractive objects introduce errors in the depth map. In the worst case no information at all can be reproduced. A method to detect these areas and still reconstruct a rough estimation of the transparent objects' shape is given in Alt et al. [3].

For further reading Han et al. [40] published a comprehensive review on Kinect based vision algorithms.

### 3.1.3 BRDF Estimation

In the previous section we presented methods that dealt with geometry reconstruction - respectively 3D shapes. This section gives an overview on how to estimate material characteristics of a surface. These material characteristics are described by the so called Bidirectional Reflectance Distribution Function (BRDF) introduced by Nicodemus [100]. This function describes how incident light gets scattered at a surface point $p$ over a hemisphere that is oriented with respect to its normal. In its most simple form the BRDF takes an incoming ($\omega_i$) and outgoing ($\omega_o$) light direction as parameters. However, more parameters are possible if for example the scatter point $p$ also influences the BRDF. Figure 3.2 illustrates a possible BRDF of a glossy material. The incoming light gets scattered mainly into the reflection direction.

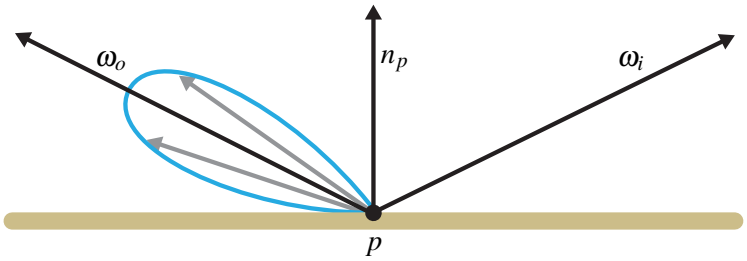



**Figure 3.2:** This figure illustrates a glossy reflection. Light comes from direction $\omega_i$ and gets scattered at point $p$. A BRDF describes how much light will be scattered into direction $\omega_o$

One can measure BRDFs, but this results in huge data sets and is very time consuming. Therefore, in most computer graphics applications analytical models [105, 14, 102] are used instead of measured BRDFs to these days. One of the most common models in use is the Phong illumination model [105], which we also use throughout this thesis. It is defined as follows:

$$f_{r,Phong}(k_d, k_s, k_n, n_p, \omega_i, \omega_o) = k_d + \frac{k_s \langle \omega_o \cdot r \rangle^{+k_n}}{\langle \omega_i \cdot n_p \rangle^{+}} \tag{3.1}$$

where $k_d$ is the diffuse coefficient, $k_s$ the specular coefficient and $k_n$ the specular exponent which defines the size of the specular highlight. $r$ is the reflected vector of $\omega_i$ along the normal $n_p$ of point $p$ and $\langle \cdot \rangle^+$ the dot product clamped to zero. Note that since we only need visually plausible illumination the Phong reflectance model is sufficient. However, for physically based rendering the energy-conserving reflectance model from Lewis [79] should be used.

### Inverse Rendering

In our proposed BRDF estimation method (see Chapter 4) we use a camera image stream to estimate the material properties of the real environment. In this related work section we therefore, focus on image-based methods, which are sometimes synonymously called *Inverse Rendering* methods. These methods try to fit parameters of an underlying BRDF model, like the presented Phong [105] or Ward model [144], to images of a scene.

Yu et al. [153] introduced *Inverse Global Illumination*, where reflectance properties are derived from a sparse set of HDR images considering also indirect illumination. The geometry is pre-modeled and partitioned into surfaces with similar materials. The direct light sources must also be known. An optimization algorithm then calculates diffuse and specular components separately. Although the concept is sound and forms the basis of newer algorithms, it needs a lot of manual pre-processing. Sato et al. [124] presented a method that also performs a reconstruction of the object's geometry from range images, which is then used to estimate diffuse and specular parameters from the same images.

Ritschel and Grosch [116] performed an on-line estimation of diffuse material properties for a known object using two HDR cameras. One camera was used to capture the object and the other one was used to capture the surrounding illumination. Specular material properties were not estimated in their approach. Boivin and Gagalowicz [7] use a single LDR image in addition

to a geometric model including light sources. Starting with a Lambertian model, they iteratively compare renderings with the original image and consider more and more complex reflectance models as long as the difference is too large. Though their solution is scalable with regard to accuracy, it is still time consuming and requires pre-processing. Mercier et al. [89] were the first to present a fully automatic method to recover the shape and reflectance properties of a single object and the position of light sources from a set of calibrated images. For that purpose, the object and light sources are fixed on a turntable, and photographs are taken every 5 degrees. The geometry is approximated by *Shape From Silhouette* (SFS) from Szeliski [135]. The method is very accurate and does not need any pre-processing, but the special setup makes it unsuitable for mixed reality. Xu and Wallace [152] used a depth sensor and a stereo intensity image to acquire an object's reflectance properties and parameters for multiple light sources. Although using a depth map comes close to our approach (see Chapter 4), their method is restricted to a single object. Furthermore, calculating light source parameters from intensity images introduces inaccuracies for flat surfaces.

Li et al. [80] use a photometric stereo approach to estimate parameters for the Ward model [144]. In contrast to other methods they only use a small local window for the estimation instead of a full image and therefore avoid a time consuming global optimization.

Zheng et al. [157] presented a solution that is similar to that of Mercier et al. [89]. One big difference is that they use measured lighting conditions instead of deriving this information from the images, which minimizes the estimation error. They then apply the highlight removal algorithm from Ortiz and Torres [103] before clustering images into regions with similar diffuse materials using K-Means. The parameters of the Ward model are then obtained for each cluster by non-linear optimization. Their algorithm is very robust, since after estimating specular factors, diffuse factors are re-estimated in order to compensate for errors caused by wrong clustering or inaccurate geometry. A follow up method proposed by Zheng et al. [156] is able to deal with weakly and highly specular objects. In contrast to the previous method they do not cancel out specularities but rather simulate these by combining different specular base functions.

Like Mercier's method, the approach is based on a controlled setup, which does not meet the requirements of mixed reality applications. This especially concerns reconstruction by shape from shading (SFS) and measurement of the light source. Their estimation pipeline however is very efficient and so we based our work presented in Chapter 4 on it. For example we also use an adaptation of the highlight removal technique from Ortiz and Torres [103] and we also use K-Means [83] for clustering. We therefore describe related work for K-Means algorithms in the next section.

Generally speaking, all these previous image-based BRDF estimation methods work off-line and have running times ranging from a couple of minutes to several hours. Furthermore, they are restricted to static scenes. Mixed reality applications are highly interactive and dynamic according to Azuma's definition [4]. Hence our motivation was to design and develop a method that runs at interactive frame rates and can thus handle highly dynamic scenes. A comprehensive overview on BRDF models for realistic image synthesis can be found in the technical report of Montes and Urena [120].

#### K-Means Implementations

Several efficient implementations of the K-Means [83] algorithm on the GPU already exist. Almost all of them use a hybrid GPU/CPU approach, where the new cluster centers in each iteration are either entirely or at least partially calculated on the CPU [47, 81, 154, 146, 30]. In all of the mentioned papers CUDA is used to perform the calculations on the GPU.

Dhanasekaran and Rubin [23] proposed a method, where the whole K-Means algorithm is done entirely on the GPU, eliminating the need of continuously copying data via the PCIe bus. Fang et al. [29] too perform the whole K-Means algorithm on the GPU, which leads to low memory bandwidth. They use bitmaps to count the elements related to a given cluster. Our approach is similar to their method. However, in contrast to their work we do not use CUDA, but rather utilize mipmaps to calculate the center of each cluster using DirectX.

### 3.1.4 Reconstructing the Camera Color Mapping Characteristics

Beside the need to reconstruct the surrounding environment, there is also the need to reconstruct the behavior of the video see-through camera device. It turns out that it is not enough to just simulate the reciprocal shading effects between real and virtual objects to make the virtual objects indistinguishable from real objects. As long as they do not look like as if they were seen through the camera itself, they will be recognized as being virtual. We therefore have to reconstruct the camera characteristics. More specific, we concentrate on the color mapping characteristics of the camera.

There are three research fields that are more or less related to our work. First of all, there are many methods for the characterization or modeling of cameras. Second, color harmonization is a topic quite recently applied to AR/MR. The last topic sees the problem not from a color space point of view between different devices, but rather as a color transfer problem between two images.

#### Characterization or modeling of cameras.

Klein and Murray [63] introduced a new compositing method for video see-through AR that simulates the most visible artifacts of small cameras. Effects of the imaging process considered are distortions, chromatic aberrations, blur, Bayer masking, noise, sharpening, and color-space compression. In this way, the appearance of virtual objects better matches those of real ones as captured by the camera. However, as mentioned in the introduction of their paper, they do not attempt to achieve an accurate color matching, which requires at least an estimation of the real scene's lighting conditions.

Color management by colorimetric characterization of cameras is another topic closely related to our work. Light that hits a sensor element in a camera is mapped to RGB values, forming the device-dependent color space. The transformation of this color space into a device-independent one, usually CIEXYZ (CIE tristimulus values), is called colorimetric characterization and is often described by an ICC profile. This is then used for color management, the mapping between device-dependent color spaces (e.g. camera to monitor). Obtaining a colori-

metric characterization is not trivial. It requires many measurements of a broad range of samples in a controlled setup and is computationally intense.

There are mainly two methods to get a characterization: Using polynomial regression [46] or neural networks. A study where these two methods were compared was done by Cheung et al. [11]. They concluded that polynomial regression is the better alternative to characterize a camera. Polynomial regression has the advantage that the sample points can be distributed non-uniformly [56]. A recently proposed method by Bianco et al. [6] uses a genetic algorithm to characterize a camera.

Color management is so far only applied to single images, mainly in digital photography and print. Fairchild [28] mentions that colors in video are purely device dependent (RGB to RGB) and presents some theoretical thoughts about how color management could work for it. His observation is also true for the image stream of video-see-through MR applications.

### Color harmonization

Another topic more remotely related to our work is color harmonization. Here colors are adjusted according to aesthetic or artistic principles to achieve a higher visual harmony [13]. Sawant et al. [125] adapted this method to video by optimizing hue values in time and space. Gruber et al. [36] applied it to video see-through augmented reality. They also introduced constraints for color harmonization to preserve certain colors. In classic AR systems, where virtual objects are rendered independently from real lighting conditions and the real scene geometry, this is an efficient approach to obtain a better match between real and virtual objects. However it would be counter-productive in our system, which considers real lighting conditions and simulates the mutual shading effects between real and virtual objects. Color harmonization would alienate the global illumination solution and cause disturbing effects due to automatically adjusted camera parameters. It was our intention to preserve video see-through images and use them as reference for adaptive color mapping. This is also what users would expect from such a system, since they are familiar with the characteristics and quality of their cameras. In that sense, users are also able to achieve the fidelity they require by purchasing the adequate camera.

### Color transfer between Images

To let the colors of virtual objects appear as seen through the camera we can also see the problem as transferring colors from one image onto another. So, what we want is to transfer the colors of the camera image onto the image that contains the virtual objects. Reinhard et al. [111] introduced the color transfer method which transfers colors from a source image onto a target image using a statistical analysis of both images. They used the $l\alpha\beta$ color space from Ruderman et al. [121] since it has uncorrelated color channels for images containing natural scenes. Reinhard and Pouli [112] investigated different color spaces and their applicability for color transfer methods. While the $l\alpha\beta$ color space has decorrelated color channels for natural scenes, this does not necessarily hold up for other types of images. Therefore, they used images containing natural daylight, manmade daylight, indoor, and night scenes to test the applicability of each color space for color transfer. Their results showed that the CIELab color space with illuminant E performs best in average, over all types of images, for color transfer. Xiao and Ma [151] how-

ever, proposed a method that works in a correlated RGB color space by calculating the mean RGB values for the source and the target images as well as the covariance matrices between the color channels. By applying translation, rotation, and scaling transformations a given RGB triple can be transformed into a desired new RGB triple.

Beside this color transfer methods, there are also methods which use the histograms of an image to transfer colors from one image to another [108, 94, 38]. However, these methods are based on histograms and are computationally more expensive than the color transfer method based on statistics from Reinhard et al. [111].

## 3.2 Reciprocal Shading

The second main chapter focuses on reciprocal shading effects. In this section we summarize related work and background information to compose real and virtual objects in a visually plausible manner. Therefore, the next two sections will introduce the rendering equation from Kajiya [53] and instant radiosity from Keller [60]. Afterwards, we will cover methods to improve the performance for instant radiosity based real-time global illumination algorithms. Once the global illumination (GI) solutions are computed the virtual objects need to be merged with the real scene. Differential rendering [22] is a commonly used method for this and used throughout this thesis. The last section will introduce work related to reflective or refractive objects in mixed reality applications.

### 3.2.1 The Rendering Equation

The rendering equation (RE) was introduced by Kajiya [53] in 1986. It describes the light transport in a scene by calculating how much light $L_o$ is leaving into direction $\omega_o$ from a point $p$. We use the more common form of the RE, which is expressed as follows:

$$L_o(p,\omega_o) = L_e(p,\omega_o) + \int_\Omega f_r(p,\omega_i,\omega_o) L_i(p,\omega_i) cos\theta d\omega_i \tag{3.2}$$

where $L_e(p,\omega_o)$ is the light emitted from point $p$ in direction $\omega_o$ - the self-illumination. The integral covers the entire hemisphere $\Omega$ which is oriented along the normal $n$ of point $p$. The BRDF $f_r(p,\omega_i,\omega_o)$ takes point $p$ and the incoming $\omega_i$ as well as the outgoing $\omega_o$ directions to simulate the material characteristics. $L_i(p,\omega_i)$ is the light that arrives at point $p$ from direction $\omega_i$. Finally, $\theta$ represents the angle between the incoming direction $\omega_i$ and the normal $n$ at point $p$. Figure 3.3 gives a geometric explanation of the rendering equation. It also shows the recursion with $L_i(p,\omega_i) = L_o(p',-\omega_i)$, where $p'$ is the nearest point in direction $\omega_i$. This is the reason why in most cases the rendering equation cannot be solved analytically.

Many methods which try to solve the RE in a numerical way can be summarized as Monte Carlo global illumination methods. They sample the integral to approximate it and recursively create light paths from the light sources to the eyes [17]. Well known examples for these methods are path tracing from Kajiya [53] or metropolis light transport proposed by Veach and Guibas [142]. Instant radiosity, which was introduced by Keller [60] in 1997, is another method that belongs to the Monte Carlo global illumination methods and is used throughout this thesis.

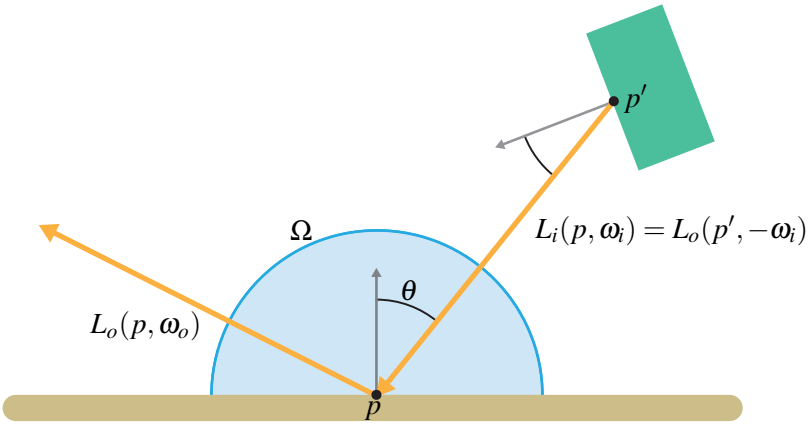



**Figure 3.3:** This figure gives a graphical illustration of the rendering equation.

### 3.2.2 Instant Radiosity

The idea behind instant radiosity is to place so-called virtual point lights (VPLs) in the scene to approximate global illumination. Instant radiosity is particularly suitable for real-time global illumination (RTGI) on current graphics hardware, as it does not need any complex pre-computations or data structures of the scene. The approach is especially suitable for low-frequency lighting effects, which is typical for diffuse indirect illumination. High-frequency lighting effects, such as caustics, would require a large amount of VPLs and thus increases computation time. In Chapter 5, we propose a combination of instant radiosity and other methods [107, 148, 149] to have such high-frequency effects in mixed-reality scenarios.

According to Segovia [129] the light paths $\bar{x} = \{x_0, x_1, x_2, ..., x_k\}$ (see Figure 3.4) from the light source to the camera, produced by instant radiosity, can be split into three parts:

- The first segment of the light path $\bar{x_c} = \{x_0, x_1\}$ is the ray starting from the camera sensor element $x_0$ towards the surface point $x_1$ which is projected on the camera sensor element.
- The second light path segment starts at surface point $x_1$ and goes to point $x_2$ where a virtual point light is placed, which illuminates $x_1$.
- The remaining part $\bar{x_s}$ of the light path $\bar{x}$ starts with $x_2$ and ends at the light source $x_k$. This path can be of arbitrary length. However, a length of 0 is also possible, meaning that $x_2$ is placed on the light source itself. An example for this are VPLs that are used to simulate the surrounding illumination as described in Section 3.1.1.

To create N VPLs first N random paths $\bar{x_s} = \{x_2, x_k\}$, starting from the light source $x_k$ are created. The VPLs are then placed at the end points $x_2$. Then each VPL illuminates all points $x_1$ which are visible to the camera. Because the last step can be done in parallel and no further scene data structures are needed, instant radiosity is very suitable for the execution on GPUs.

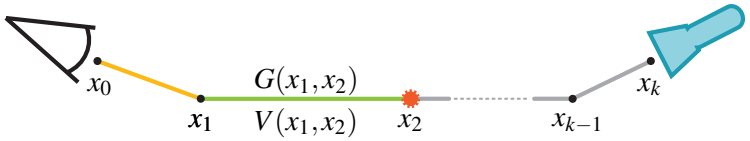



**Figure 3.4:** This figure illustrates the three parts of a path created with instant radiosity. $x_2$ is the point that gets illuminated by the VPL placed at point $x_3$. From $x_3$ towards the light source a path of arbitrary length can be used.

When VPLs are used to approximate the GI solution the rendering equation can be rewritten as follows [17]:

$$L_o(x_1 \to x_0) = \sum_{i=1}^{N} f_r(x_1, x_1 \to x_2^i, x_1 \to x_0) V(x_1, x_2^i) G(x_1, x_2^i) f_r(x_2^i, x_2^i \to x_3^i, x_2^i \to x_1) \phi_i \quad (3.3)$$

where the integral from Equation 3.2 gets substituted by the sum over N VPLs and the directions are written in the form of start and end points ($x_s \to x_e$). $x_2^i$ is the location of the $i$th VPL, $\phi_i$ the flux of the $i$th VPL, and $V(x_1, x_2^i)$ the visibility term between $x_1$ and $x_2^i$. The geometry term $G(x_1, x_2^i)$ is defined as follows:

$$G(x,y) = \frac{\max(0, cos\theta_x)\max(0, cos\theta_y)}{||x-y||^2} \quad (3.4)$$

where $\theta_x$ is the angle between the surface normal and the direction vector pointing at $y$. The same applies for $\theta_y$ but with the normal of point $y$ and the direction vector towards $x$. The geometry term has a singularity where the distance between points $x$ and $y$ gets zero. A common solution to this problem is to clamp the influence of each VPL with the cost of loosing energy and therefore introducing a bias. However, there are solutions to compensate this energy loss such as from Novák et al. [101].

Although instant radiosity is very suitable for the graphics card, there are performance bottlenecks due to visibility calculation and illumination computation for each pixel illuminated by a VPL. The next section will therefore introduce real-time global illumination methods and algorithms to improve the performance for visibility tests and shading.

For further reading we refer to the STAR of Dachsbacher et al. [17], which is an in-depth review on many-light methods.

### 3.2.3 Real-time many-light methods

Instant radiosity has two major computational bottlenecks - visibility calculation and shading costs. Dachsbacher and Stamminger [18] extended standard shadow maps to so-called reflective shadow maps (RSM). These RSMs store enough information so that each pixel can be treated as a light source. However, since this is too computationally intensive they adaptively sample the reflective shadow map to create VPLs. In this way they were able to calculate indirect illumination. However, the indirect illumination computation did not contain any visibility calculation

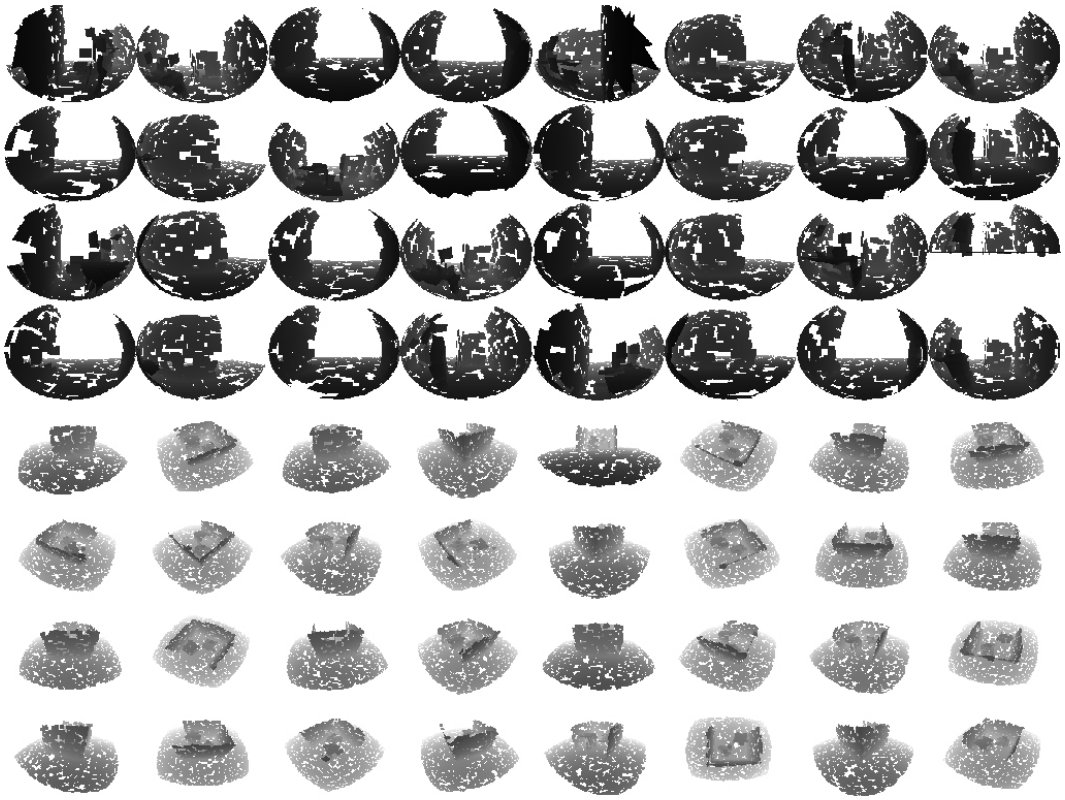


**Figure 3.5:** This figure shows 64 imperfect shadow maps in one large texture buffer. Each of these ISMs will be used for determining visibility for indirect illumination. Note that the splats are already aligned to the tangent of its corresponding surface (see Chapter 5). The top half of the ISMs are used for a spot light while the bottom half is assigned to the environment light source.

since generating shadow maps for each VPL is too costly. A first solution to this limitation was proposed by Laine et al. [73]. They developed a real-time instant radiosity method that caches the shadow map for each VPL over several frames. In this way, only a few shadow maps need to be recreated every frame, thus achieving real-time frame rates. A limitation of this method is however, that moving objects cannot influence indirect visibility calculation. In 2008, Ritschel et al. [117] introduced the concept of imperfect shadow maps (ISMs). The idea is to represent the scene as a sparse point cloud and use this point cloud to generate a shadow map for every VPL. Using this approach it is possible to create hundreds of shadow maps per frame, which are all stored in one large texture buffer. This method enables to have completely dynamic scenes, with the downside that the visibility calculations are imperfect. However for low frequency indirect light this is in most cases sufficient. Figure 3.5 shows 64 imperfect shadow maps in one large texture buffer. Another possibility for visibility calculation is to use a voxelized representation of the geometry as proposed by Thiedemann et al. [138].

To overcome the second bottleneck, one of the first steps is to avoid unnecessary shading

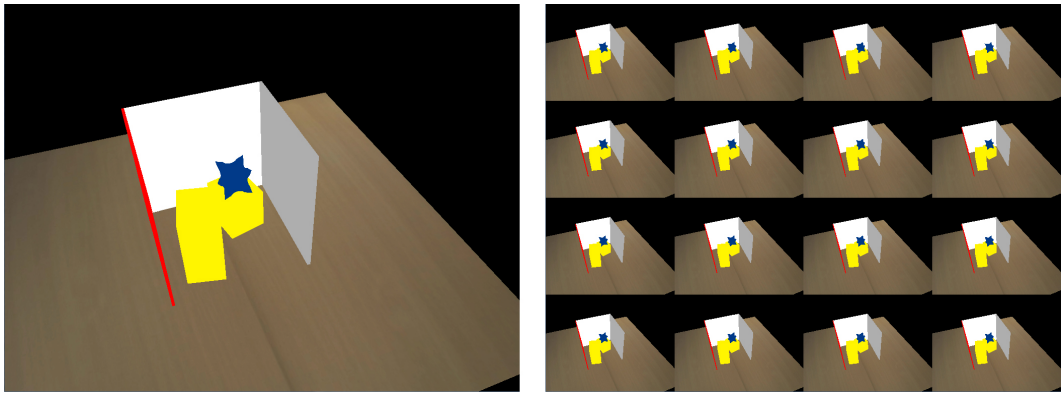


**Figure 3.6:** The color information of the G-Buffer is shown in the left image. To decrease shading costs, the complete G-Buffer gets split into small tiles (in this case $4 \times 4$ tiles) as shown in the image on the right. A pixel is assigned to a tile by taking the pixel position modulo 4 and the result corresponds to the tile id (horizontally as well as vertically).

costs by using a deferred rendering system. In such a system a so-called geometry buffer (G-Buffer) is rendered first. It stores all necessary information, like position, normals, and material parameters of all points visible to the camera. In a second step, the G-Buffer is used to shade each pixel. In this way, only the visible stored points are shaded with a light source, respectively VPL, and no pixels which would be overdrawn later (as would happen with forward rendering) are shaded. To further reduce shading costs several methods were developed. Segovia et al. [128] introduced an interleaved sampling pattern for deferred shading, which we used in our methods as well. In this method the G-Buffer gets split into several smaller tiles as shown in Figure 3.6. Each patch is assigned to a subset of VPLs and thus each VPL illuminates only as much points as are in one tile. After the illumination computation the split G-Buffer gets merged again. To further reduce artifacts the merged illumination buffer must be filtered in a geometry aware way taking the normal and the depth discontinuities between neighboring pixels into account.

Dachsbacher and Stamminger [18] also reduce the influence area of each VPL. Points where the influence of a VPL is below a certain threshold are not shaded by the VPL at all. Using this method, they are able to simulate caustic effects to a certain degree. Nichols and Wyman [99] exploit the low-frequency nature of indirect light. Their idea is that on a large flat surface it is not always necessary to shade each pixel. It is sufficient to shade only certain points of interest, where the illumination changes drastically, and use intelligent interpolation schemes between those points. A similar approach for a mixed reality scenario was presented by Lensing and Broll [78]. A recent method also from Lensing and Broll [76] uses a more advanced interpolation scheme where not the indirect illumination but the flux, the position, and the surface normal of each VPL are averaged to get a new averaged VPL, which is then used to shade a point. In this way, they are able to get convincing results with only little indirect lighting error while shading costs are reduced drastically. Ritschel et al. [115] use CUDA to perform fast final gathering on a hierarchical scene representation of splats. The images look very impressive but at the time of publishing the frame rates were too low for mixed-reality applications.

Beside these acceleration approaches, there are methods that do not use many-light methods directly. Kaplanyan [57, 58] uses a light propagating volume, where radiance is propagated from source voxels based on spherical harmonics coefficients. The method does not include visibility but delivers a good GI approximation at very high frame rates. Another voxel based method was proposed by Crassin et al. [15]. This method creates an octree on the fly and allows for visibility and incoming energy computation using approximative voxel cone tracing. The method produces impressive images and is capable of rendering diffuse and glossy materials at interactive frame rates. McGuire and Luebke [87] extend photon mapping in a way that photon volumes are splat in screen-space to compute indirect illumination. Wang et al. [143] are able to simulate several illumination effects by clustering the scene into coherent shading clusters on the GPU. Final gathering is then performed but performance is still too low for mixed-reality applications. Nichols et al. [97] perform screen-space hierarchical radiosity using the multi-resolution splatting technique from Nichols et al. [99], which greatly reduces shading costs. Ritschel et al. [119] introduce a method similar to screen-space ambient occlusion called screen-space directional occlusion (SSDO). It samples the neighboring screen-space pixels and uses these to calculate indirect illumination.

These are just a few RTGI methods and there exist several other methods to compute global illumination at interactive to real time frame rates. For a comprehensive overview, we refer to the STAR of Ritschel et al. [118].

### 3.2.4 Merging Real and Virtual Scenes

Nakamae et al. [91] were the first to concentrate on merging real and virtual scenes. They had a simple geometry for the real scene and information about the date and time when the background picture was taken. From this information, they could place the sun to calculate shadows cast from virtual objects. Fournier et al. [33] used a progressive radiosity algorithm to compute global illumination for mixed reality scenes. Depending on whether the object of a patch belongs to a virtual or real object they changed the calculation behavior. Drettakis et al. [25] extended this method to dynamic virtual objects. Another approach for mutual light interaction between real and virtual objects was presented by Pessoa et al. [104]. They use an environment map for each object in an augmented reality scene. While performance scaling is an issue when more objects are placed in the scene, they get very impressive results for simple scenes and are able to simulate many illumination effects. The most common method called differential rendering (DR) for adding virtual objects into real scenes was introduced by Debevec and is presented in the next section.

### 3.2.5 Differential Rendering

Debevec [22] introduced differential rendering (DR), which is based on the work of Fournier et al. [33]. Mathematically DR can be expressed in two simple equations:

$$\Delta L = L_{rv} - L_r \tag{3.5}$$

$$L_{final} = L'_{cam} + \Delta L \tag{3.6}$$

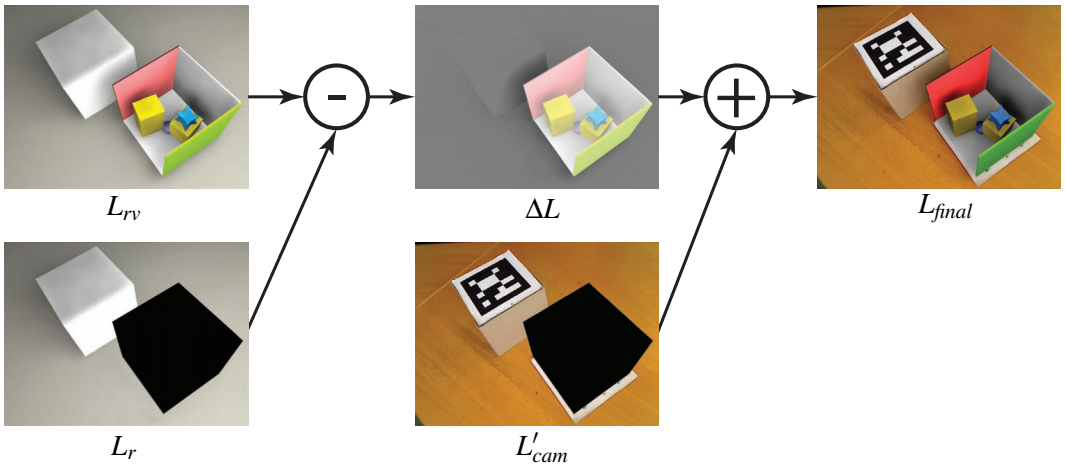



**Figure 3.7:** The two global illumination solutions $L_{rv}$ and $L_r$ are subtracted from each other, leading to the difference buffer $\Delta L$. This buffer gets added to the masked video image $L'_{cam}$ to compose the final images. Note that for better illustration, zero in the difference buffer is mapped to grey. Therefore, areas darker than grey will cause a shadow effect and areas brighter than grey will cause indirect color bleeding on real objects.

where $L_{rv}$ and $L_r$ are the two global illumination buffers. While $L_{rv}$ stores the global illumination solution, taking the real *and* the virtual objects into account, $L_r$ contains a global illumination solution which takes only the real objects into account. $\Delta L$ is the difference buffer, which gets added to the masked camera image $L'_{cam}$. Pixels, where virtual objects are visible, are masked out. Figure 3.7 illustrates these steps. In areas, where virtual objects cast shadows on real objects the difference buffer $\Delta L$ contains negative values. In contrast, in areas where virtual objects cause indirect illumination on real objects the values are positive. Note that these effects (casting shadows and indirect illumination) happen from virtual on real objects and from real on virtual objects. Another advantage is that differential rendering greatly reduces the error that is introduced by BRDF estimation at the cost of having to compute two global illumination solutions. A limitation of DR is that the geometry of the real scene must be known.

### 3.2.6 Reflective and Refractive Objects For Mixed Reality

The first approach to handle reflective and refractive objects in a mixed reality setting was introduced by State et al. [132] in 1996. They used an outside-in approach (see Section 3.1.1) where a chrome or glass sphere was placed at a known position. Then the image of the sphere was extracted from the video feed and remapped onto reflective or refractive objects.

Generally, differential rendering [22] is also able to render reflective and refractive objects into images of a real scene. However, the reflected/refracted objects visible in reflections, respectively refractions, belong to the rendered representation of the real objects, which means that the effects of differential rendering are not applied to reflective or refractive objects. In Chapter 5 we present a solution to overcome this limitation.

Grosch [34] extended differential rendering with photon mapping, which makes it possible to consider additional light paths caused by reflection and refraction. His so-called differential photon map contains photons with positive and negative energy, which in combination yield caustic patterns on real surfaces. Reflection and refraction of real objects through virtual ones are simulated by image back-projection. Although this approach produces impressive results, it is an off-line method applied to single images.

A recent method by Karsch et al. [59] is able to add reflective and refractive objects including caustics to images. However, the method is mainly designed for easy editing of the images and therefore does not fulfill Azumas [4] requirement for interactivity.

Reflective and refractive objects also appeared in a master thesis by Pirk [106]. Refractions are rendered on the GPU by using the image of the see-through camera as background texture or a static cube map, where appropriate lookup functions approximate distortion effects. Pessoa et al. [104] also use environment maps to render reflections and refractions for mixed reality applications. They combine this with spatial BRDFs including Fresnel factors and sophisticated texture sampling on the GPU. In both approaches [106, 104], only virtual objects can have refractive materials – neither caustics, nor light attenuation are considered. The static environment maps used in [106] are an even more severe limitation for the highly dynamic nature of mixed reality scenarios. Uranishi et al. [141] used a box marker to estimate the reflectance and roughness of the floor to simulate its reflection properties. Another method proposed by Tawara and Ono [136] showed caustics cast by virtual water in an augmented reality setup. However, the method does not take surrounding illumination into account.

A recent rendering method was proposed by Kan and Kaufmann [55] in 2012. They use a real-time ray-tracing framework instead of a rasterization approach, as we do, to render reflections, refractions and caustics in mixed reality scenarios. They are able to produce high-quality results, but the frame rate is still low for completely dynamic scenes with caustics enabled.

## 3.3 Evaluation

What does it take to make virtual objects look photorealistic and even better, make them indistinguishable from real objects? Ferwerda [31] introduced three different varieties of realism and pointed out that an image is just a representation of a scene. This representation describes selected properties and we should not confuse this with the real scene. The three varieties are:

*Physical realism*, where the visual stimulus of a scene is the same as the scene itself would provide. Physical realism is hard to achieve due to the lack of appropriate display devices that can recreate the exact frequency spectrum.

*Photo-realism*, where the visual response is the same as invoked by a photograph of the scene. This kind of realism should be targeted in photorealistic MR systems based on video see-through output devices. If the virtual objects are represented using the same kind of photorealistic mapping function, they would be indistinguishable from real objects.

*Functional realism*, provides the same visual information as the real scene. That means that the image itself can be rather abstract, but the information retrieved from it is the same. A construction manual of a cupboard will contain abstract drawings but usually no photographs for example.

### 3.3.1 Studies on photorealism

Having Ferwerda's [31] three varieties of realism helps to focus on what kind of realism we want to achieve in photorealistic MR. However, it is still not fully understood what photo-realism actually means in a perceptual context. Therefore, Hattenberger et al. [41] conducted experiments to find out which rendering algorithm creates the most photorealistic images. They used a real scene and added a virtual cow in the middle of it. Several different rendering algorithms were used to calculate the final results. Observers had to choose between two images compared to a photograph of the scene and decide which one looks more real. Results showed that observers preferred light simulations that took indirect illumination into account and furthermore, that noisier images were preferred to more smooth ones (with some exceptions). Although the authors state that the results cannot be generalized because they belong to this particular scene, the results indicate that there are also other important factors in photorealistic MR that influence the perception of the scene.

Elhelw et al. [27] tried a different approach. They used an eye-tracking system to find the gaze points in images. From that they derived which image features were important for the participants (five of those had background knowledge in graphics and tissue appearance, eleven did not have such knowledge) to decide if the image looks real or not. They found light reflections/specular highlights, 3D surface details, and depth visibilities to be very important image features. For their user study, they used different sets of images from clinical bronchoscopy. These images look quite abstract in shape and texture. However, it would be very interesting to test this method on other images that are related to MR applications.

These are two examples of user-studies that tried to find answers on what makes an image photorealistic, without altering specific image features. We propose to divide the known image features in a MR setup into two main categories: The *visual cues* described in Section 3.3.2 and the *augmentation style* described in Section 3.3.3. While visual cues have a local nature, augmentation style can be seen as global feature in an image.

### 3.3.2 Visual Cues

Visual cues are very important for the human visual system (HVS) as they help to organize and perceive the surrounding environment. Visual cues can deliver depth information and let us recognize inter-object relationships.

In MR visual cues can be exploited to embed virtual objects into the real scene. We split visual cues into *inter-object spatial cues* and *depth cues*.

**Inter-object spatial cues:** *Shadows* belong to the strongest spatial cues available. They define a spatial relationship between the shadow caster and the shadow receiver. The influence of shadows was studied in several experiments (see Section 3.3.4). Furthermore, Rademacher et al. [110] found that the characteristics of soft-shadows changed the perceived realism in images.

Like shadows *indirect illumination* between objects defines a spatial relationship. Although inter-reflections are not as strong a cue as shadows are, their influence is still significant [85].

**Depth cues:** Beside spatial cues such as shadows or indirect illumination, cues that serve as a source for depth information are of particular interest as these allow reconstructing our surrounding environment. Drascic and Milgram [24] as well as Cutting [16] presented a list of depth cues that can be divided into four main groups: Pictorial depth cues, kinetic depth cues, physiological depth cues, and binocular disparity cues.

*Pictorial depth cues* are features that give information about the object's position in a still image. Such cues can be occlusion, linear perspective, relative size, texture perspective, or aerial atmospheric perspective.

*Kinetic depth cues* provide information through change of the viewpoint or moving objects. Relative motion parallax and motion perspective (falling raindrops - near vs. far) are two examples. Another cue is the so-called kinetic depth effect. Imagine a point cloud that rotates around an axis. The structure of the point cloud is easily recognized. However, if the cloud stops rotating every point falls back into the screen plane and the structure is not visible anymore.

*Physiological depth cues* are evaluated by the HVS according to the convergence and accommodation of the eyes.

*Binocular Disparity* is another depth cue that is similar to the motion parallax depth cue. The HVS automatically transforms the disparity seen due to our two eyes into depth perception. Obviously this cue only exists when a stereo rendering setup is used in experiments.

### 3.3.3 Augmentation Style

Beside visual cues that should be provided by the rendering system, it is also important that the augmentation style of virtual objects is similar to the visual response of the scene. Kruijff [72] mentioned several areas where perceptual issues may arise.

**Illumination:** Virtual objects that are rendered into the captured image of the real world must be illuminated correctly. This is often done by using a chrome sphere to capture the incident illumination at the point where the objects will be placed. This method belongs to the outside-in approaches. Debevec [22] introduced a way to use several images, with different exposure times, to create a high dynamic range (HDR) environment map. However, this process is time consuming and only leads to a static environment map. Inside-out methods instead use a camera with a fish-eye lens to capture the surrounding hemisphere. These methods allow for dynamic environments. Unfortunately, there are only a few HDR cameras on the market. So the source for the incident illumination is only of low dynamic range. Once the environment map is acquired, image based lighting methods can be used to illuminate the virtual objects.

**Color and Contrast:** Currently most cameras offer only a limited color gamut and contrast. These limitations lead to wrong color and contrast representations. A special problem due to this mapping arises, when two different cameras are used; one for video see-through and one to capture the surrounding illumination. Both map the high dynamic range illumination into a low dynamic range, *but* with different mapping functions and into a different device-dependent *RGB* color space resulting in wrong colors in the final composed image.

**Tone-mapping:** The ideal setup for a photorealistic mixed-reality system would consist of two equal HDR cameras for video see-through and environment capturing. Using these two cameras with the same configuration would make the virtual objects look correctly illuminated and there would be fewer errors from the capturing stage. Then the whole rendering process could be performed in HDR and ideally the resulting images would be presented on a HDR display. As we do not have a HDR display, our framework uses a tone-mapping operator developed by Reinhard et al. [113], which can be implemented directly on the graphics hardware, or uses either adaptive camera color mapping methods as presented in Chapter 4.

**Camera Artifacts:** Computer generated images normally look absolutely clean/perfect and do not suffer from artifacts like noise or blurred edges. However, since we embed the virtual objects into a captured video frame, we need to add these artifacts to the virtual objects; otherwise they will immediately be recognized as not being real. Klein and Murray [63] developed a method that imitates a couple of artifacts such as Bayer pattern approximation, motion blur or chromatic aberration. Fischer et al. [32] could improve visual fidelity by removing aliasing artifacts and adding synthetic noise to the rendered objects. These artifacts greatly increase the appearance of the virtual objects.

### 3.3.4 Further studies on perception

There is already a lot of research which studies the influence of shadows and indirect illumination in AR and VR applications. Hubona et al. [49] experimented with positioning and resizing tasks under varying conditions. They found significant differences for all independent variables. Sugano et al. [133] studied how shadows influence the presence of virtual objects in an augmented scene. The experiments showed that the shadows increased the presence of the virtual objects. Madison et al. [85] generated several different images of a plane and a cube. With different visual cues enabled and disabled, the participants had to tell whether the cube was touching the plane or not. Similar to that work, Hu et al. [48] generated several different images of a plane and a large box using a Monte-Carlo path tracer. Their results showed that stereo vision is a very strong cue followed by shadows and indirect illumination. Furthermore, shadows combined with indirect illumination are similarly as strong as stereo vision. In a recent user study, Lee et al. [75] studied how different qualities of visual realism influence search tasks. They created three virtual models with increasing degree of visual realism of a real campus. Then they let participants perform search tasks in a virtual environment using the three models and additional info labels, to find out, whether the visual quality of the models influenced the search performance. Furthermore, they were interested if results from user studies in virtual environments are also valid in real-world scenarios. Therefore, they also tested their search tasks using an outdoor AR setup on the real campus where they augmented the real scene with additional info labels. The participants had to perform 16 search tasks of different difficulties and their results showed that out of that only 4 showed a significant effect regarding visual quality. However, the authors also mention that these 4 significant effects could happen due to difficult outdoor lighting conditions and differences between the real and the virtual model.

Kan and Kaufmann [55] also performed a user study to measure the perceived quality of different rendering features like reflections and refractions, anti-aliasing, caustics, and depth of field. First, they showed the participants a video sequence with all effects enabled. The participants were asked how realistic the video looked like on a scale from -3 (not realistic at all) to +3 (completely realistic). Furthermore, they asked the people which objects they thought are real or virtual. Their results showed that in average 40.1% mistakenly marked virtual objects as being real and 36% of real objects were marked as being virtual. In the next phase, the participants saw two video sequences with different effects activated and they had to judge which one looks more realistic, again on a scale from -3 (first video looks more realistic) to +3 (the second video looks more realistic). The results show that anti-aliasing has the highest influence on perceived realism, followed by reflection and refraction effects. Depth of field was the third strongest effect. Caustics were evaluated as having the least impact.

Besides researching the influence of shadows and indirect illumination, some studies investigate thresholds in environmental illumination. Nakano et al. [92] studied how much the resolution of an environment map could be decreased until the increasing error is noticeable. Lopez-Moreno et al. [84] studied how much the illumination direction for an object could differ until human observers noticed the error. For their study they presented the participants eight random objects with random shapes and different textures. All objects except for one were illuminated from the same direction. Then they measured at which threshold angle the participants recognized the differently illuminated object as being illuminated from another direction. The results showed that the maximum threshold angle was even larger in real scenes than in synthetic ones. However, only static environments were used for these experiments and it would be interesting how the thresholds work in dynamic setups.

Similar to the preliminary study presented in Chapter 6, Thompson et al. [139] tried to find out if improved rendering methods also improve distance judgment. The experimental setup and distances to estimate are different to our user study. However, their results are similar to ours (see Section 6.3.3).

For further reading on user evaluations in mixed reality we refer to the work of Bai and Blackwell [5] as well as to Dünser and Billinghurst [26].

CHAPTER 4

# Reconstruction

## 4.1 Introduction

As outlined in Chapter 1, it is necessary to have knowledge about the surrounding environment in order to be able to simulate mutual lighting effects between real and virtual objects. Besides the geometry itself, i.e., the 3D surface of the real objects, it is also necessary to have knowledge about the material characteristics of these surfaces. The video see-through camera adds another layer of image features/artifacts affecting how real objects appear on the display. We therefore also have to reconstruct this behavior. This chapter deals with techniques to perform this reconstruction.

Section 4.2 briefly discusses how we estimate the surrounding illumination at real objects. Section 4.3 outlines two methods for better geometry reconstruction than just using the raw depth information from the Microsoft Kinect sensor. In Section 4.4 we describe our method for interactive BRDF estimation also using the Microsoft Kinect sensor. Finally, Section 4.5 presents two methods for adaptive color mapping of the video see-through camera so that virtual objects have similar colors as the real objects.

## 4.2 Light Estimation

In most cases the real environment is illuminated by some surrounding light, like direct sun light or artificial light sources. These light sources cause shadows and indirect illumination effects on real objects, and of course, these effects should also be visible for virtual objects. Therefore, it is necessary to capture the incident illumination from the real environment. As described in Chapter 3, there are basically two ways to do that. Either use a chrome sphere in the real scene to extract environment data or use a camera with a fish-eye lens attached. We always used a camera with a fish-eye lens in our methods.

Once a video frame is captured, it gets importance sampled using hierarchical warping from Clarberg et al. [12]. This algorithm works very well with mipmap levels of the luminance

probability map and thus allows us to perform importance sampling directly on the GPU. The resulting sample points are then reprojected onto a hemisphere. The next step is to place virtual point lights at the sample positions. In this way, the incident light at the real scene can be simulated in the successive global illumination computation.

## 4.3 Geometry Reconstruction

The differential instant radiosity rendering method, which we describe in Chapter 5, uses a deferred rendering system to minimize the amount of pixels to shade. A so-called geometry buffer (G-Buffer) contains all the necessary information, from the point of view of the camera, to shade each pixel. Thus, it contains either the depth or 3D position, the normal, and material properties. The geometry reconstruction step has to deliver two maps which store the 3D surface points and their normals of the real scene so that they can be inserted into the G-Buffer.

We present two methods for surface geometry estimation from the data stream available from the Microsoft Kinect sensor. Both methods heavily use the GPU and exploit temporal coherence (TC) between adjacent frames in a similar way as done by Scherzer et al. [126]. The main difference between our two methods is that in the first, published in Knecht et al. [66], we filter the normals and use the raw depth map to calculate the 3D surface points, while we already filter the depth map in the second method, published in Knecht et al. [68]. Both methods are very fast compared to the normal estimation of the Point Cloud Library (PCL) [122], but only the second method achieves a similar normal estimation quality.

An alternative to these reconstruction methods is of course to manually model the real scene by hand. However, this is a time-consuming task and also limits dynamic scenes because moving objects must be tracked explicitly.

### 4.3.1 Microsoft Kinect Sensor

For both methods the Microsoft Kinect sensor [90] is used to acquire depth maps. The Kinect device is a very low-cost video and depth sensor. Figure 4.1 shows the two input streams, an *RGB* color image and the depth image. Both streams are delivered in a resolution of $640 \times 480$ @ 30 Hz, and surrounding objects can be captured in a range approximately between 0.45 and 10 meters. The high frame rates make this sensor suitable for dynamic scenes. However, the sensor uses a structured light pattern to measure the depth values. Therefore, if the projected light pattern is not visible, no depth values can be acquired. This especially happens at depth discontinuities, at transparent objects, or in outdoor scenarios, which are very bright due to direct sunlight (see Section 4.3.4 for more details). Another problem for geometry reconstruction is that the input of the depth image is very noisy. In a naive approach, the 3D surface points would be calculated from the depth map of the Kinect sensor. Then, in a second step, the normals would be calculated by looking up the neighboring 3D surface point positions in the according buffer. However, the resulting normal estimations would have very low quality, and they would not be feasible for illumination computation. Therefore, filtering methods are needed.


(a) (b)

**Figure 4.1:** Image (a) shows the *RGB* color input stream and image (b) shows the depth values of the Microsoft Kinect sensor.

### 4.3.2 Raw depth, raw position and filtered normal estimation

The basic idea behind our first method is to reuse information about the normals of previous frames to create better normals in the current frame. Furthermore, normal estimations from previous frames should be subsampled depending whether the normals belong to a flat area or to edges. In this way, flat areas should have smooth normal estimates, while edges are still preserved.

The method is based on two render passes. The first pass performs subsampling and averaging of the normals from the previous frame. Furthermore, a curvature coefficient is calculated. The subsampling causes a smoothing on the normals of the previous frame. Let $(i, j)$ be the row and the column of a given pixel in the previous frame. The average normal is then calculated by averaging over $(i-1, j)$, $(i+1, j)$, $(i, j-1)$ and $(i, j+1)$. Note that if no normal is available at a given pixel location, it will be discarded from the calculation. The curvature coefficient is calculated as follows:

$$curv_H(i,j) = N_{i-1,j} \cdot N_{i+1,j} \tag{4.1}$$

$$curv_V(i,j) = N_{i,j-1} \cdot N_{i,j+1} \tag{4.2}$$

$$curv(i,j) = \min\left[curv_H(i,j), curv_V(i,j)\right]^{128} \tag{4.3}$$

where the dot is the dot product operator. Note that the curvature coefficient goes to zero at sharp edges and to one at flat areas. The average normal and the curvature coefficient of the last frame are rendered to a render target with half the dimension of the rendering window.

The second rendering pass consists of two steps. In the first one, a new normal is calculated from the depth map delivered by the Microsoft Kinect sensor. We look up the 3D position $p_{i,j}$ at the current pixel $(i, j)$ and two neighboring positions in horizontal $(i, j+4)$ and vertical $(i+4, j)$ direction. A distance value of four pixels showed good smoothing characteristics while edges

were still preserved. From these values, we can set up a surface normal as follows:

$$d_{i+4,j} = \frac{p_{i+4,j} - p_{i,j}}{|p_{i+4,j} - p_{i,j}|} \tag{4.4}$$

$$d_{i,j+4} = \frac{p_{i,j+4} - p_{i,j}}{|p_{i,j+4} - p_{i,j}|} \tag{4.5}$$

$$normal_{i,j} = d_{i+4,j} \times d_{i,j+4} \tag{4.6}$$

In the second step, the information calculated by the first rendering pass is used to calculate an old average normal. First, the lookup coordinates are calculated by using reprojection. In this way, the camera movement from one frame to another can be compensated. The curvature coefficient at the current pixel steers the mipmap level for the lookup of the previous normal. The new and the previous normal vectors are linearly combined depending on a confidence value calculated as follows:

$$c_N = |N_p \cdot N| \tag{4.7}$$

$$c = c_B * c_N + (1 - c_N) \tag{4.8}$$

where $N_p$ is the previous averaged normal and $N$ is the new normal. $c_N$ is the confidence coefficient based on the similarity of the previous and the new normal. The resulting confidence is a linear blend between a base confidence $c_B$ and 1, steered by $c_N$. To deal with disocclusions occurring during camera movement, we set the confidence value $c$ to zero if the depth difference between the old frame and the new frame is larger than 0.1 meters. In this way, normals of dynamic objects get updated faster.

### 4.3.3 Filtered depth, filtered position and filtered normal estimation

The previously presented approach has the drawback that only the normals are filtered and not the 3D surface point positions as well. Therefore, the idea for the second method is to reuse available information as early as possible. To do so, we reuse the depth values of the previous frame to improve the acquired depth map from the Kinect sensor of the current frame. In this way, the depth values already have a high quality and thus also the 3D surface points and their normal estimations will have a higher quality. This is a similar approach as presented by Newcombe et al. [96], but in contrast to their voxel-based approach, our method works entirely in screen space and performs better in dynamic scenes. The following steps are performed to get better 3D surface point position and normal estimations:

- Warp depth map from previous frame into new camera view using forward projection
- Merge the new depth data with the warped depth
- Estimate normals based on new depth map

For the warping step, a triangle mesh with $640 \times 480$ vertices is created. The vertices store texture coordinates $(u, v)$ for lookup into the depth/weight buffer. The camera pose $T_{current}$ is

obtained by marker-based optical tracking using Studierstube Tracker [127], and the pose of the last frame $T_{prev}$ is simply cached.

First, for each vertex, the according depth and weight values from the depth/weight buffer of the previous frame are looked up. Then the $(u, v, depth_{prev})$ triple must be back-projected to homogenized view-space coordinates $p$ using the inverse projection function $\phi^{-1}$.

$$p = \phi^{-1}(u, v, depth_{prev}) \tag{4.9}$$

These positions are in the view space of the previous frame ($T_{prev}$) and therefore need to be transformed into the viewing coordinate system of the current frame ($T_{current}$). Equation 4.10 calculates the warping transformation $T_{warp}$ and Equation 4.11 warps a given point $p$ from the old viewing coordinate system into the current viewing coordinate system.

$$T_{warp} = T_{prev}^{-1} T_{current} \tag{4.10}$$

$$\hat{p} = T_{warp}(p) \tag{4.11}$$

Finally, the depth value is calculated by applying the projection function $\phi$ on the warped point $\hat{p}$ as shown in Equation 4.12.

$$depth = \phi(\hat{p}).z \tag{4.12}$$

Note that parts of the warped depth mesh may overlap, but visibility will be correctly resolved in the z-buffer with the depth test enabled. However, certain vertices or triangles may have invalid depth values or be degenerated (edge length larger than a certain threshold) and are therefore, discarded.

The render target for the warping step stores two float values per pixel. One for the warped depth value and one for a weighting factor that gets applied in the merging step. The weight values, which were looked up from the depth/weight buffer of the previous frame, are simply copied into the render target.

The second step is to merge the warped depth values with the newly available depth values from the Kinect. Our approach for merging the new depth values and weighting values is similar to Newcombe et al. [96]. If both depth values are valid, but their difference exceeds a certain threshold, which indicates that they belong to different surfaces, only the new depth value will be written into the render target. After the merging step a depth map is available that has lower noise and less holes than the raw depth input of the Kinect sensor.

In the last step, the normals are estimated as follows:

$$N(u, v) = (V(u+1, v) - V(u, v)) \times (V(u, v+1) - V(u, v))$$

where $(u, v)$ are the screen-space coordinates and $V(u, v)$ the world space position stored in the position map at $(u, v)$.

### 4.3.4 Limitations

Some limitations of our method are imposed by the Microsoft Kinect sensor, which is a structural light scanner. In general, depth values cannot be calculated when the light pattern is not

recognized by the system. This happens when objects are very glossy, reflective, transparent, or absorb too much of the infrared light. The infrared pattern also vanishes in direct sunlight, making the estimation methods unsuitable for outdoor mixed reality. Furthermore, the border of curved objects is also often missing from the depth map because the projected light pattern is too distorted there. The Kinect also suffers from constant measurement errors along the image plane. For our second presented method, this implies that the Kinect should be moving during depth map acquisition. Otherwise these constant measurement errors will also appear in the reconstructed geometry.

A further drawback of these screen-space approaches is that areas which just got disoccluded from the previous frame will have low-quality depth values, i.e., due to interpolation at a depth discontinuity, and therefore, also wrong normal estimations. For these areas it takes some frames until the correct normal is estimated again.

The reconstruction methods reuse data from previous frames and therefore, temporal artifacts appear when interacting with the scene. Although outliers are rejected during the reprojection phase, they cannot be filtered out completely.

In the differential instant radiosity method, described in Chapter 5, spotlight sources also use the data of the geometry reconstruction to calculate shadow maps. However, since we reduce the reconstruction problem to screen-space rather than to a volume-based approach, like in Newcombe et al. [96], the system is not aware of any geometry that is not visible to the camera. While this results in fast execution, the user might observe artifacts when interacting with virtual spot light sources. Imagine that the spot light illuminates a scene from approximately 90 degrees of the observer's view. Then shadow calculations may lead to wrong results since a wall facing the spotlight is not necessarily visible to the camera.

### 4.3.5 Results

Figure 4.2 shows a comparison of two depth maps and four different normal estimation methods. Image 4.2(a) shows the raw depth map and image 4.2(b) how the result of a normal estimation on the raw depth map would look like. It is obvious that lighting computations will have a low quality if image 4.2(b) will be used. Image 4.2(c) shows the normal map calculated with the Point Cloud Library (PCL) and a smoothing factor of 10. The average computation time is 223 milliseconds. The normal map which is computed with our first proposed method (raw depth, raw position and filtered normal) takes about 0.57 milliseconds to compute and is shown in image 4.2(d). The PCL based normal map has a lot of holes, visible as black areas. In our first method these holes are filled with the normals of the neighboring pixels. Even though these normals are not correct from a reconstruction point of view, they reduce visible artifacts a lot. Furthermore, note that our method produces sharper edges. The normal map shown in image 4.2(f) uses our second method to estimate 3D surface points and normals (filtered depth, filtered position and filtered normal). The normals are calculated in 1.8 milliseconds and have lower noise and fewer holes on flat areas and therefore, the final illumination results are of higher quality. The corresponding filtered depth map is shown in image 4.2(e).

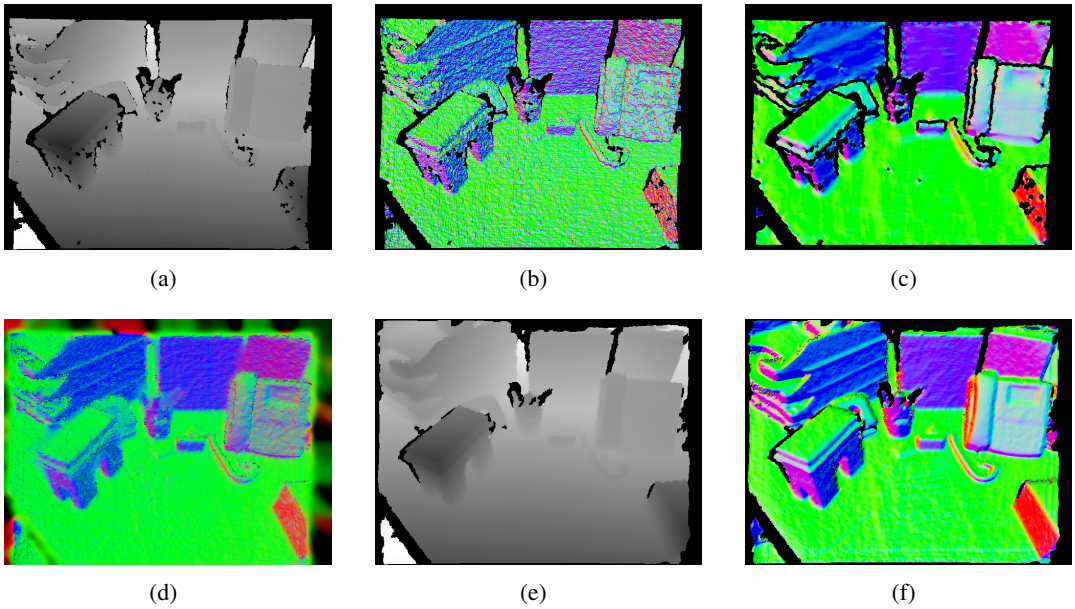


**Figure 4.2:** Image (a) shows the raw depth values captured with the Kinect sensor. Using these values as source to estimate normals results in image (b) which has very poor quality. The PCL [122] creates smoother normal estimations shown in (c) but at high computation costs (223 milliseconds). The normal estimation method we presented first is shown in image (d). The estimation quality is higher than the raw normal estimation (b) but still suffers from noise. Compared to the PCL solution the normal estimation only takes 0.5 milliseconds. Our second normal estimation approach uses the filtered depth buffer (e) and estimates normals from it, as shown in image (f). The results are of higher quality and also fewer holes are visible. The computation time for image (f) is 1.8 milliseconds.

### 4.3.6 Conclusion

In this section we presented two methods to reconstruct the 3D surface using the Microsoft Kinect sensor. Compared to the normal-estimation method of the Point Cloud Library, the two methods perform normal estimation multiple times faster. Both methods exploit temporal coherence between adjacent frames, which reduces the noise of the input data. However, artifacts due to real dynamic objects or interacting hands may appear. The first presented method closes holes but has lower normal estimation quality, while the second presented method does not close all holes but the estimated normals are of higher quality.

## 4.4 BRDF Estimation

In the previous section we presented two methods for geometry reconstruction. However, beside the geometry of the real scene and the real lighting conditions, the BRDFs of real objects are also needed to simulate mutual shading effects. Acquiring this data in a pre-processing step would

diminish the dynamic and interactive nature of mixed-reality systems, and would also make it necessary to track the previously modeled movable real objects. We therefore present a BRDF estimation method, introduced by Knecht et al. [66], which runs at interactive frame rates. It is based on real-time reconstruction using the structural light scanner provided by Microsoft's Kinect sensor [90]. The real lighting conditions are captured by a camera with a fish-eye lens from which light sources are derived.

Our contribution is best characterized by the unique features of our BRDF estimation approach, which are:

- It runs at interactive frame rates.
- It does not need any pre-processing.
- It utilizes a novel K-Means implementation executed on the GPU.

### 4.4.1 Overview

Estimating material characteristics for mixed reality applications is a challenging task, due to several constraints. The most important challenge is the time constraint, since the applications have to be interactive. Furthermore, the observed scenes usually exhibit a certain degree of dynamics, and materials that have just appeared in the camera frustum need to be estimated immediately. As described in Chapter 3, several methods for BRDF estimation exist, but all of them are designed for offline purposes. They all try to get a very accurate BRDF estimation. In our case this goal must be lowered to achieve interactive frame rates. The resulting diffuse and specular reflectance maps are used in a differential instant radiosity (DIR) system (see Chapter 5), where the goal is to get visually plausible images instead of physically correct ones. Mapping this idea to our BRDF estimation method, our goal is to find BRDFs that emphasize the same visual cues to the user as the real materials would do.

Our BRDF estimation algorithm is mainly influenced by the ideas of Zheng et al. [157]. Their method was designed to work offline and thus had different requirements. As an adaption we modified their approach where necessary and made extensive use of the GPU to obtain interactive frame rates.

Figure 4.3 illustrates the separate steps which get executed in the pipeline. The reconstruction process (blue box) was already described in the previous sections. The BRDF estimation (green box) is described in the next section where, after some intermediate steps, the final diffuse and specular reflectance parameters are estimated. Finally, the output of our method is passed to the DIR rendering system. There the geometry data, including the material characteristics, will be directly rendered into a G-Buffer, which stores the 3D position, the normal, the color, and the material parameters needed, to shade each pixel.

### 4.4.2 BRDF Estimation

The main steps for BRDF estimation are *Highlight removal*, *Diffuse estimation*, *Clustering*, and *Specular estimation*. The first step is necessary to get a proper clustering (step three) of the

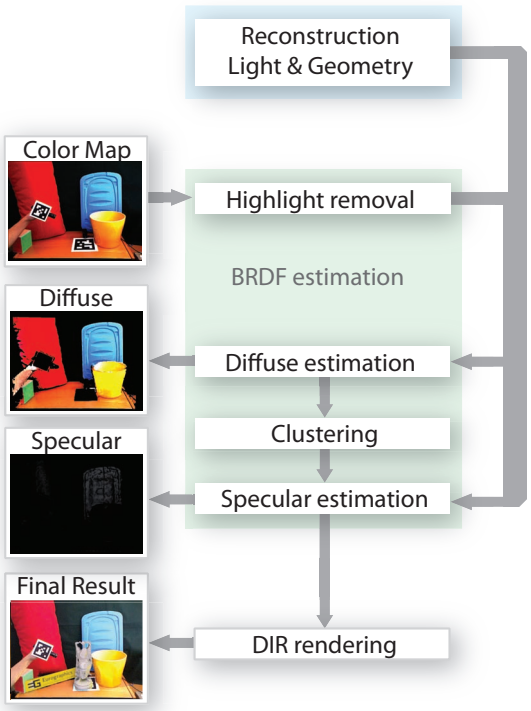



**Figure 4.3:** This figure shows the main steps in the BRDF estimation pipeline. The inputs of the BRDF estimation method are the color map, a subset of the estimated light sources (see Section 4.2), and the reconstructed geometry (see Section 4.3). Steps belonging to BRDF estimation are shown in the green box.

different materials because highlights would result in a separate cluster. In the second step, inverse diffuse shading is applied, and afterwards this buffer gets clustered. Zheng et al. [157] also removes highlights in the input image and applies inverse diffuse shading afterwards. However, in their approach, the resulting buffer was just used for clustering. In contrast, we also use this buffer as a diffuse reflectance map to keep the computation time low. In step three each cluster is now handled as one material and on each one the specular intensity $k_s$ and the specular power $k_n$ are estimated.

### Highlight Removal

To estimate specular reflectance values, similar colors need to be clustered since they are assumed to belong to the same material. However, specular highlights would form a separate cluster due to saturation, which is not desired. Our highlight removal step is based on the work of Ortiz and Torres [103] and first transforms the camera color image into the Hue Saturation Intensity (HSI) color space. Highlights should be detected at pixels where the color has high brightness but low saturation. As thresholds we set the minimum brightness to 0.9 and the max-

(a) (b) (c)

**Figure 4.4:** Image (a) shows the RGB input of the Kinect sensor. Image (b) shows the highlight mask. In a second step, the masked pixels are filled as shown in image (c).

imum saturation to 0.1. In a first pass, the highlight detection result is written into a binary mask with a one where the brightness and saturation criteria are met and a zero otherwise. Then a morphological dilation with a disk (radius of 4 pixels) is performed. While Ortiz and Torres [103] perform a *Morphological Vectorial Opening by Reconstruction*, we use a rather simplistic reconstruction method. For each pixel that is masked as a highlight, a new color has to be found that ideally matches surrounding colors. We do this by iterating through neighboring pixels in an increasing circular manner until a pixel is found that is not masked as belonging to a highlight. Then the color of the found pixel is used to substitute the color of the masked pixel. In this way, all highlights can be canceled out. Note that due to this highlight removal process, bright and weakly saturated objects may get misinterpreted as highlights. The results of the highlight removal operation are shown in Figure 4.4.

### Diffuse Reflectance Estimation

After highlight removal, we estimate the diffuse parameters at pixel $(x,y)$ with the following formula for the diffuse reflectance estimation $k_d$:

$$k_d = \frac{I}{\sum_{i=1}^{N} I_i \langle \omega_i \cdot n(x,y) \rangle^+} \tag{4.13}$$

where $I$ is the input intensity, $I_i$ the intensity of the $i$th light source, $\langle \omega_i \cdot n(x,y) \rangle^+$ the dot product between the surface normal $n(x,y)$ and the direction $\omega_i$ towards the $i$th light source clamped to zero, and $N$ is the number of lights that are used for the BRDF estimation. Zheng et al. [157] estimate the diffuse parameters at a later stage because they used multiple RGB samples per vertex. We use the resulting buffer as diffuse reflectance map and as input for the clustering. The estimated diffuse reflectance map is shown in Figure 4.5. In the ideal case the different objects would have completely flat colors. However, this is not the case due to several simplifications that introduce consecutive errors in the pipeline. First, the environmental light is represented by only a few virtual point lights. Second, no shadows or indirect illumination are taken into account and third, the normal estimation is not totally correct.

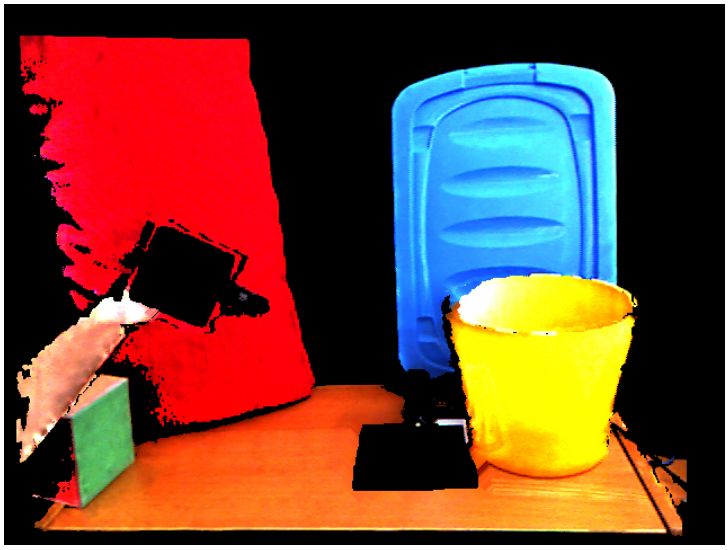


**Figure 4.5:** This image shows the estimated diffuse material component $k_d$. In the ideal case the objects would look perfectly flat and no variations due to different normals and thus illumination would be visible.

### Clustering

Pixels with similar RGB colors in the diffuse reflectance map are assumed to have the same material and therefore need to be clustered. A novel K-Means implementation that is executed on the GPU performs the clustering. K-Means was introduced by Stuart P. Lloyd [83] and it consists of the following steps:

**Step 1: Initialize cluster centers:** The resulting clusters heavily depend on the initial values chosen for the cluster centers. Thus, if bad initial cluster centers are chosen, it might take many iterations until convergence. For each frame, we therefore use one to two different initial cluster centers. The first set uses the cluster centers from the previous frame, and if the stopping criteria are met (see step 4), the next iteration is not executed anymore. However, if they are not met, the second set is executed with random cluster center values.

**Step 2: Assign element to nearest cluster:** Step two is adapted slightly so that step 3 can be executed on the GPU. Instead of just outputting the nearest cluster id and the minimum distance, we need to render each color pixel into multiple render targets. The idea is that each cluster has its own render target and pixels rendered into a given render target only belong to a certain cluster. We used eight simultaneous render targets and can handle six clusters each time a screen-space pass gets executed. The following information is stored on a per-cluster basis for each pixel:

- The *RGB* color value
- The minimum distance to the nearest cluster center
- A binary flag that defines to which cluster the pixel belongs

The *RGB* color and minimum distance can be stored in one texture buffer with four floating point values. For the binary flags of all six clusters, we used two textures where every cluster gets assigned to one color channel. Depending on the cluster id assigned to a given pixel color, the color and distance information is only written into the textures assigned to the specific cluster. All other render target values are set to zero.

**Step 3: Calculate new cluster centers:** In step three, we need to calculate the average *RGB* value for each cluster, which is then used as a new cluster center. For a texture $T$ with a size of $2^n \times 2^n$, there are $n$ mipmap levels that can be created. The smallest mipmap level with a size of $1 \times 1$ stores the average value of all data in texture T. However, we only want the average *RGB* color of those pixels that belong to a given cluster and ignore those that were set to zero.

The cluster center can therefore be calculated using a combination of the two lowest mipmap levels from the color texture and the binary flag texture as follows:

$$cluster_c(T_{RGBD}, T^*) = \frac{avg(T_{RGBD})}{\sum_{i=0}^{n} \sum_{j=0}^{n} T^*_{i,j}} \tag{4.14}$$

where $T_{RGBD}$ is a cluster specific texture containing the *RGB* color values and the distance value. $T^*$ is the binary texture for the cluster filled with ones where pixels are assigned to that cluster and zeros otherwise.

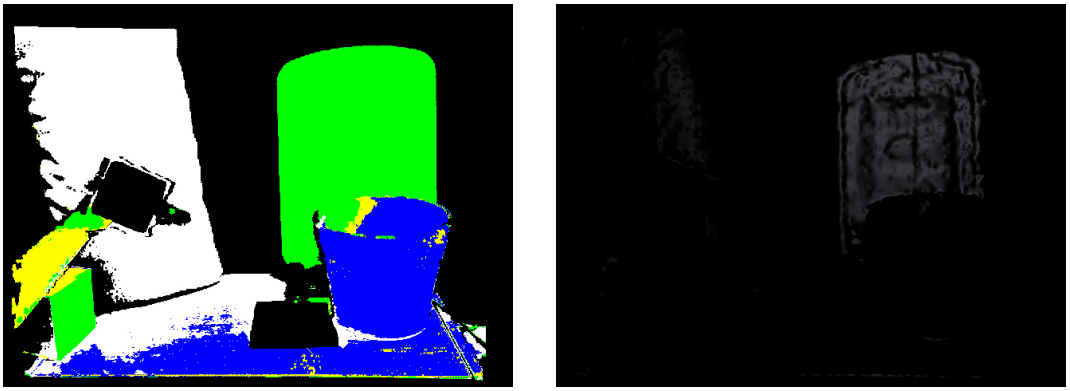
**Step 4: Repeat steps 2 & 3:** In the original K-means method [83], the second and third steps are repeated until no data element changes the cluster anymore. This stopping criterion is too conservative for our needs. We need a fast clustering algorithm and thus have lowered the stopping criteria: First, only a maximum number of 20 iterations are performed defining the upper bound for the computation time. Second, if the variance change from one iteration to another drops below $10^{-4}$, no further iterations are executed. By exploiting temporal coherence, a low variance solution may be available after the first iteration and no new cluster center set needs to be processed. Note that the variance is calculated in a similar way to the cluster centers by exploiting mipmapping. As the squared distances for each pixel to the cluster centers are already calculated in the shader, the variance can be calculated nearly for free in step 2.

If the first run with the old cluster centers as initial values does not converge, the second run with random cluster centers gets executed. Then the cluster centers with the lower variance values are used for BRDF estimation. However, always using just the previous cluster centers could lead to a local minimum for clustering and there would be no way out to maybe find the global one. For this reason, in every fifth frame, the second iteration with random cluster centers will be executed anyway. Figure 4.6(a) shows the resulting clusters after K-Means is applied on the diffuse reflectance map.

### Specular Reflectance Estimation

One of the last steps needed in the BRDF estimation pipeline is the estimation of the specular intensity $k_s$ and specular power $k_n$ values per cluster. We assume white highlights and thus $k_s$ is reduced to a scalar value.

The parameters are estimated similar as proposed by Zheng et al. [157]. However, there are two main differences in our method. First, the solver works partly on the GPU and thus gains



(a) (b)

**Figure 4.6:** Image (a) shows the resulting clusters after K-Means is applied. Image (b) shows the result of the specular component estimation.

more speed than just a plain CPU implementation. Second, the positions of the light sources are not chosen to be fixed variables. The reason for this is that the positions are evaluated using importance sampling and thus can vary over time and furthermore, need not to be at the exact position where a small light source is placed. However, the position of a light source highly influences the position of the specular reflection and therefore, small variations of the initial positions are allowed for the solver.

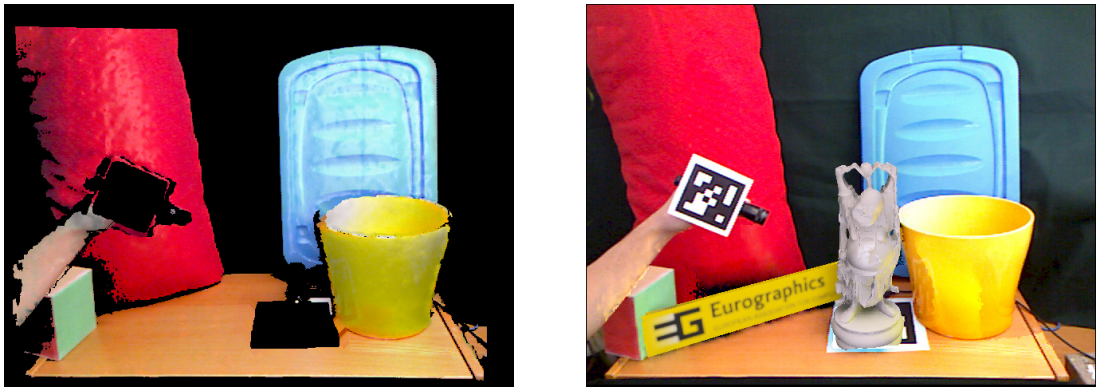
For the non-linear optimization, a Nelder-Mead algorithm [93] was used with the following objective function evaluated on the GPU:

$$F_j = \sum_i \left[ I_i - \sum_{l=1}^{N} I_l k_d \langle n \cdot \omega_l \rangle^+ + I_l k_s \langle \omega_o \cdot r_l \rangle^{+k_n} \right]^2 \tag{4.15}$$

where $i$ iterates over all pixel intensities $I_i$ which are related to cluster $j$. $I_l$ is the intensity of the $l$th light source and $k_d$ is the diffuse intensity vector of a cluster, which is set to the cluster center color. Note that for the specular component estimation, $k_d$ is fixed and only the light source positions as well as $k_s$ and $k_n$ can be varied by the solver. $n$ is the normal vector of the surface, $\omega_l$ the direction vector pointing towards the $l$th light source, and $r_l$ the reflection vector of the $l$th light source. $\omega_o$ is a view vector pointing towards the camera. The result of the specular reflectance estimation is shown in Figure 4.6(b). Figure 4.7(a) shows a simple Phong rendering using the estimated $k_d$, $k_s$ and $k_n$ Phong reflectance parameters. In this case, the same VPLs are used for illumination that are also used to estimate the BRDFs. In image 4.7(b) a rendering using DIR with an additional virtual pocket lamp is shown. Note the yellow indirect illumination on the real desk and on the virtual Buddha.

### 4.4.3 Limitations

Since bright pixels are assumed to be highlights due to specular reflections, bright and weakly saturated objects may be misinterpreted as highlights. Furthermore, shadows are not considered




(a) (b)

**Figure 4.7:** Image (a) shows a simple Phong rendering with the VPLs used for BRDF estimation. In image (b) a virtual pocket lamp illuminates the real scene. Note the yellow color bleeding on the real desk and on the virtual Buddha.

directly in the current implementation. Pixels with a brightness value below a certain threshold are simply discarded.

The K-Means clustering approach uses a variance value to decide whether further iterations are needed or not. However, there is no estimation of the optimal amount of clusters right now. This number must be specified by the user in advance and highly depends on the materials available in the scene.

Although temporal coherence is exploited at several stages in the pipeline, we do not continuously integrate already-seen geometry data as done by Ritschel and Grosch [116]. This would be helpful, as a given point in the scene could be viewed under different viewing angles, leading to a better BRDF estimation but could also lead to problems with moving objects.

Due to the real-time constraints several simplifications are introduced. The environmental illumination is approximated using only a few virtual point lights, and the clustering also introduces some errors. All these simplifications lower the quality of the final BRDF estimation. Therefore, user studies have to reveal whether the approximations are sufficient for visually plausible mixed-reality applications or not. However, evaluations in an interactive experiment setup might be a problem, due to the low frame rates.

### 4.4.4 Results

Our computer system consists of an Intel Core 2 Quad CPU at 2.83 GHz and an NVIDIA GeForce GTX 580 with 1.5 GB of dedicated video memory. The software is running on Windows 7 64-bit and implemented in C# and DirectX 10. Cameras used in this scenario are the Microsoft Kinect sensor and an IDS uEye camera to capture the environment map. Furthermore, the method described in Section 4.3.2 was used for normal estimation.

### K-Means clustering

We compared the proposed K-Means clustering implementation against the OpenCV library [9]. In the test setup a data set of $640 \times 480$ 3-vector elements needed to be clustered. We ran both algorithms 5 times, each time with different initial cluster centers. The iteration count of each run was set to 20. Note that no temporal coherence was exploited in order to get comparable results. The measured times include all the 5 runs and do not include the setup of the random data elements. Table 4.1 shows the execution times in seconds for 6 and 12 clusters.

| Clusters | OpenCV | GPU K-Means |
|---|---|---|
| 6 | 3.94s | 0.33s |
| 12 | 7.07s | 0.44s |

**Table 4.1:** Shows a comparison between the K-Means implementation from OpenCV [9] and our GPU implementation. Both algorithms ran 5 times with 20 iterations. Timings show the total execution of the 5 runs in seconds.

Table 4.2 shows the average iteration count needed for the first 50 frames to get below the variance change threshold. The columns show the average number of iterations for 6 clusters (6C) and for 12 clusters (12C). The rows show if the cluster centers from the previous frame were used (frame$^{-1}$) or if the cluster centers where chosen randomly (random). We set the maximum iteration count to 30, which was never reached during this test.

| Initials | Avg. Iter., 6C | Avg. Iter., 12C |
|---|---|---|
| random | 9.00 | 11.47 |
| frame$^{-1}$ | 7.53 | 6.98 |

**Table 4.2:** Shows the average iteration count when reusing the cluster centers from the previous frame or randomly taking new ones.

### Performance Analysis

The BRDF estimation pipeline has several steps. Table 4.3 gives an overview on the time spent for each one. In this setup, 6 clusters and 4 VPLs were used to approximate the BRDFs.

| Stage | Time in ms |
|---|---|
| Normal Estimation | 0.57ms |
| Highlight Removal | 0.94ms |
| Diffuse estimation | 0.23ms |
| K-Means | 39.08ms |
| Specular estimation | 315.76ms |
| Total time: | 356.58ms |

**Table 4.3:** Shows the time spent on each pipeline stage.

It clearly shows that the specular estimation step consumes by far most of the time. However, if it is possible not only to use a hybrid CPU / GPU version for the optimization but a complete GPU solution, the performance should increase a lot.

Two main parameters can be tweaked to get a better performance for a given scenario. One parameter is the number of materials that are estimated every frame. The second parameter is the number of virtual point lights that are used to approximate the surrounding illumination. Table 4.4 shows the impact of different cluster and VPL settings with a fixed maximum iteration count for the specular estimation set to 50.

| VPLs | 6 Cluster (fps) | 12 Cluster (fps) |
|---|---|---|
| 1 | 3.82 | 2.41 |
| 8 | 2.23 | 1.30 |
| 128 | 1.70 | 1.01 |
| 256 | 0.99 | 0.59 |

**Table 4.4:** Shows the average fps with different VPL and cluster settings.

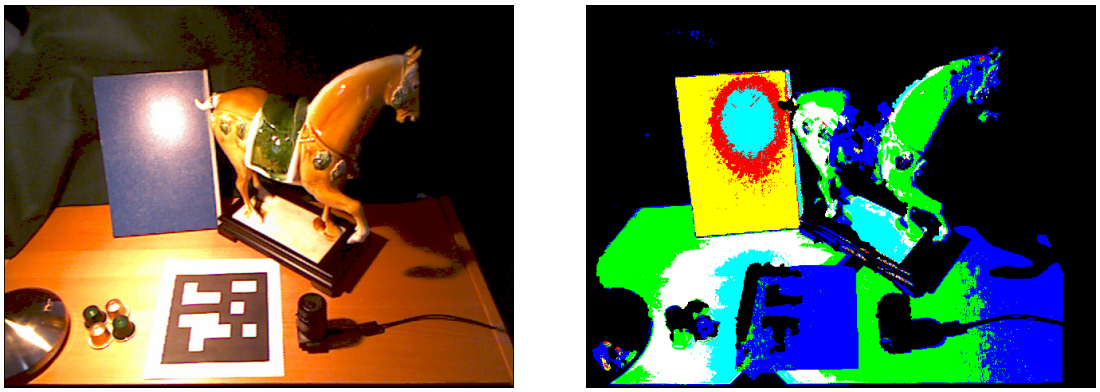
We also investigated how the maximum iteration count for the specular estimation solver reduces the total error. Interestingly, the change of the error was extremely small regardless how large the value was set. We think that this has to do with the large amount of pixels that are available in a cluster. Compared to that the area of a specular highlight is relatively small and thus correct estimations will only have a small impact on the total error.

Furthermore, it turned out that the solver has difficulties in finding appropriate values in certain cases. Sometimes there is simply no highlight due to a given VPL. We therefore, introduced a threshold value for a maximum error. If the error is too large, we set the specular intensity $k_s$ to zero. Another problem could be that the solver just has one single point of view per frame whereas Zheng et al. [157] used several photographs to perform a specular estimation.

### Critical Scenario

A critical scenario is shown in Figure 4.8(a). It shows a blue notebook and a horse made from porcelain placed on a wooden box. The light is mainly coming from the direction of a table lamp partially visible on the left side of the image, causing specular highlights on the blue notebook envelope and the wooden box. The number of clusters used in this scenario is six, and four virtual point lights are used to estimate the surrounding illumination. The average frame rate is 2.3 fps.

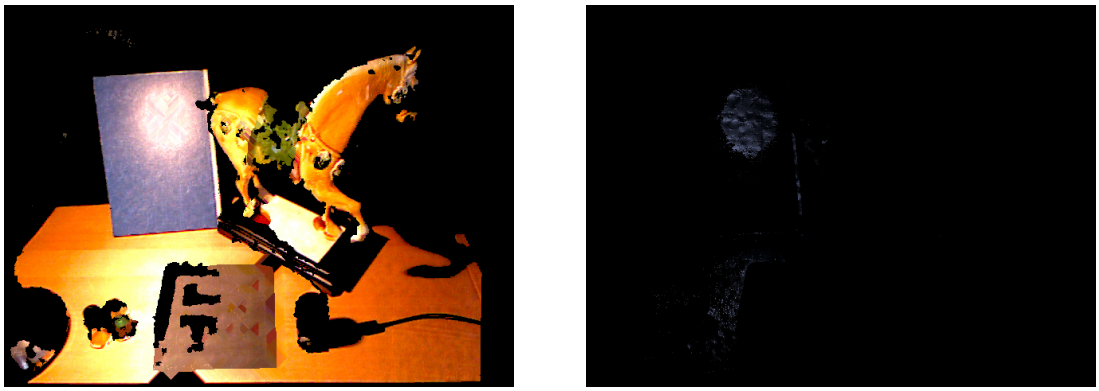
In this scenario, the clustering is not as distinct regarding the objects compared to the first scenario. Due to the highlights caused by the spotlight, the clustering step creates different clusters for the same material as shown in Figure 4.8(b). Furthermore, the Kinect sensor has troubles finding depth values at the white borders of the tracking marker, resulting in holes in the estimations (see Figure 4.9). Figure 4.10 shows the resulting diffuse (a) and specular (b) estimations. The Phong-shaded result is shown in Figure 4.11.



(a) (b)

**Figure 4.8:** Image (a) shows the video input captured by the Kinect sensor. The clustering result is shown in image (b).

**Figure 4.9:** This figure shows the depth values acquired by the Kinect sensor. Note that it failed to measure depth at the white borders of the tracking marker, the black fish-eye lens camera, and the bottom part of the table lamp.



(a) (b)

**Figure 4.10:** This figure shows the scene rendered with just the diffuse estimation (a) and specular estimation (b).

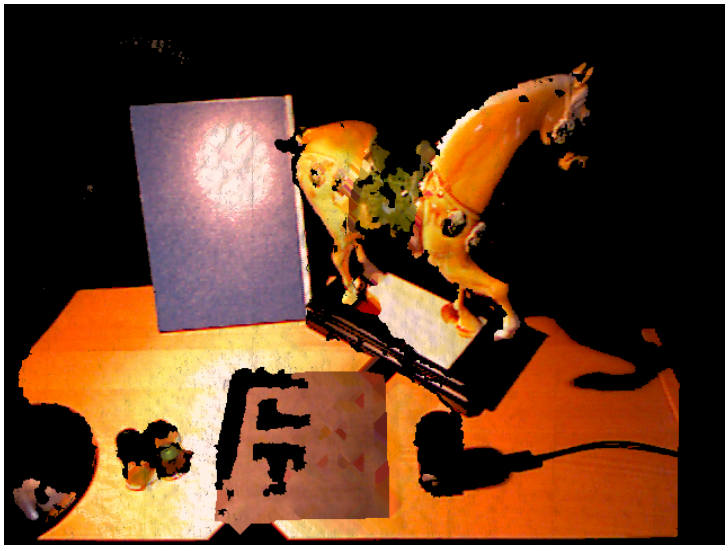


**Figure 4.11:** This figure shows the Phong-shaded result combining the diffuse and specular estimations.

### 4.4.5 Conclusion

We introduced a method to estimate the BRDFs of an augmented scene at interactive frame rates. The method does not need any precomputation, which makes it suitable for mixed-reality applications. The Microsoft Kinect sensor serves as a data input source to reconstruct the surrounding environment. We proposed an adapted K-Means implementation that is specially tailored towards BRDF estimation and fast execution on the GPU. Temporal coherence is exploited, which allows us to find clusters faster than with a conventional implementation. The Phong parameter estimation is performed using a hybrid CPU / GPU variation of the Nelder-Mead optimization algorithm. Although the method works in dynamic environments and is thus suitable for mixed-reality applications, the low frame rates still limits its use in practice.

## 4.5 Adaptive Camera-Based Color Mapping

Since the observing camera, through which the real environment is seen, has its very own transfer function from real-world radiance values to RGB colors, superimposed virtual objects look artificial just because their rendered colors do not match with those of the real objects seen through the camera.

This section presents two approaches that reconstruct the color mapping behavior of the observing camera during runtime to render the virtual objects as if they were seen through the observing camera. The first method was published by Knecht et al. [69] and in this thesis we will denote it as the *sample-based color mapping* approach. The second method published as a Bachelor thesis of Spelitz [131] is an adaption of the method introduced by Reinhard et al. [113], which is a statistics-based method, and therefore, denoted as *statistics-based color mapping* approach.

In contrast to the classical methods of color management, our approaches are adaptive and fulfill the real-time requirements of mixed-reality systems. They work on a frame-by-frame basis and only consider those colors that are currently in the image. This allows switching cameras at any time and promptly reacting to automatic camera adjustments like gain and shutter time or saturation.

### 4.5.1 Overview

Differential rendering from Debevec [22] uses two global illumination solutions of the local scene ($L_{rv}$ and $L_r$). $L_{rv}$ stores the global illumination solution including real and virtual objects, whereas $L_r$ stores a global illumination solution taking only the real objects into account. Then, the difference between both buffers is computed and added to a masked version of the video frame ($L'_{cam}$) (see Chapter 3 for more details). The GI solution buffers ($L_{rv}$ and $L_r$) are normally in high dynamic range and therefore, tone mapping needs to be applied first. The final image $L_{final}$ is then composed as follows:

$$\Delta L = TM(L_{rv}) - TM(L_r) \tag{4.16}$$

$$L_{final} = L'_{cam} + \Delta L \tag{4.17}$$

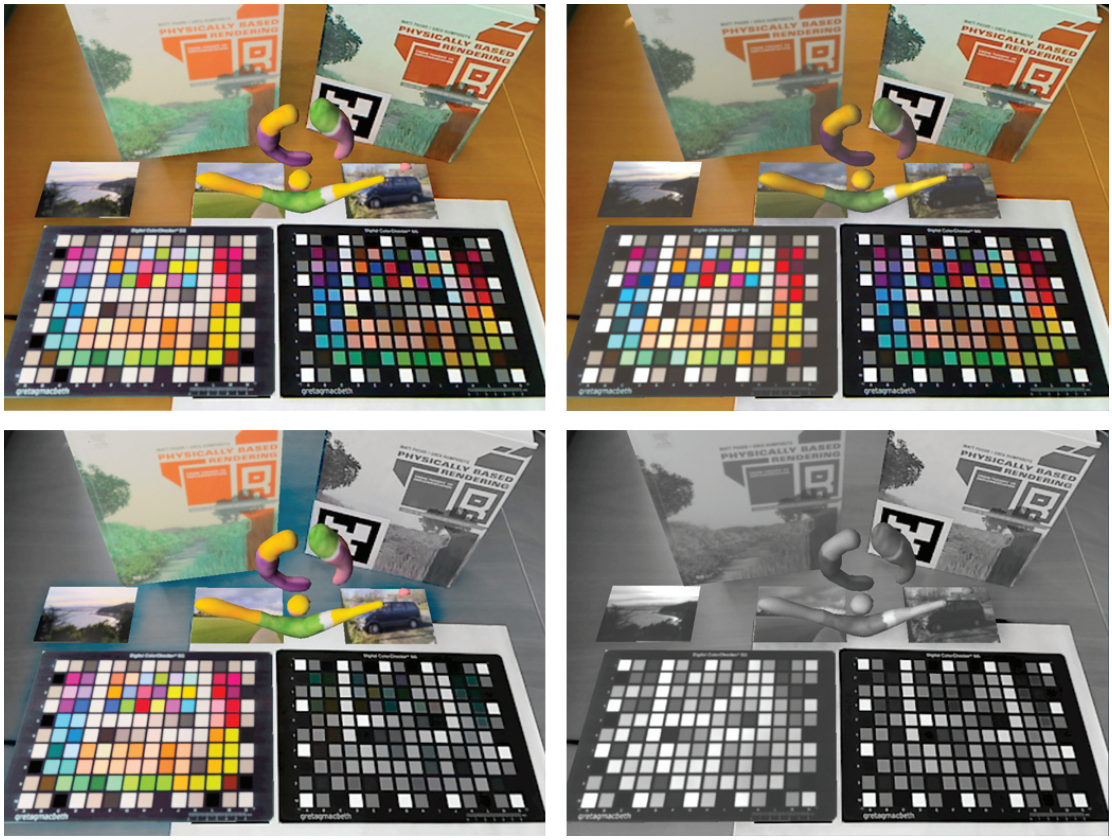


**Figure 4.12:** The left column shows renderings with the tone-mapping operator from Reinhard et al. [113] without any adaption to the characteristics of the observing camera. On the right column our sample-based color mapping method is applied, resulting in better adapted colors as they were seen through the camera. In each image the color chart and the book on the right side are real objects.

where $TM$ is the tone-mapping function and $\Delta L$ the difference image. In Figure 4.12 the two images in the first column use this kind of differential rendering with a simplified version of the tone-mapping operator proposed by Reinhard et al. [113]. However, tone-mapping functions were not designed for use in mixed-reality systems and thus do not take the camera specific behavior into account. In contrast, our presented methods take the camera characteristics into account and thus can also handle changes of its parameters during runtime. The second column in Figure 4.12 shows results from our *sample-based color mapping* method. Please note that our method focuses on color adjustment. Simulation of other camera artifacts, such as lens distortions for a better matching appearance of virtual objects, has been addressed by Klein and Murray [65, 63].

### 4.5.2 Sample-Based Color Mapping

In the sample-based method all computations are done in device-dependent color spaces. The fisheye camera captures the light that is defined through device-independent radiance values $XYZ_{world}$. This camera converts the incident light into RGB values, thus defining the color space that we will denote as reference color space $RGB_{ref}$. All elements that interact with light (materials and light sources) must be defined in that reference color space in order to get correct results. For example, in the current setting we cannot use the color-mapping method together with the BRDF estimation method because the BRDF estimation method does not take the color-mapping function of the Kinect sensor into account. It would be necessary to apply the inverse color-mapping function first and estimate the BRDFs afterwards. Otherwise, the material colors are estimated in the wrong color space, which would lead to different colors after the color-mapping function $\tau_{sa}$ is applied.

The observing camera has a completely independent RGB color space denoted as $RGB_{cam}$. The presented method tries to find an appropriate mapping function $\tau_{sa}$ from $RGB_{ref}$ to $RGB_{cam}$ as shown in Figure 4.13.

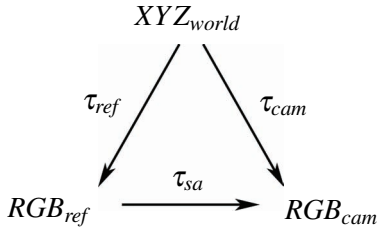



**Figure 4.13:** This figure illustrates the dependencies of the different color spaces. The proposed method tries to find a mapping function $\tau_{sa}$ from $RGB_{ref}$ to $RGB_{cam}$.

In order to find this mapping function, we need to gather information of the $RGB_{ref}$ and $RGB_{cam}$ color spaces that correlate with each other.

The sample-based color mapping approach combines an online camera characterization method with a heuristic to map colors of virtual objects to colors as they would be seen by the observing camera. However, it may also happen that the virtual objects have colors which are not available in the real scene. In such cases, the heuristic will generate samples of colors which are not present in the real scene. To reduce temporal flickering, we also illustrate a way to temporally smooth the color-mapping operator.

In order to allow the $L_{rv}$ and $L_r$ GI solution buffers to be mapped from $RGB_{ref}$ color space to the $RGB_{cam}$ color space, Equation 4.16 must be altered slightly as follows:

$$\Delta L = \tau_{sa}(L_{rv}) - \tau_{sa}(L_r) \tag{4.18}$$

$$L_{final} = L'_{cam} + \Delta L \tag{4.19}$$

Here, $\Delta L$ is the difference image and $L_{final}$ represents the final composed image in $RGB_{cam}$ color space.

Since we have the $L_r$ solution and the $L_{cam}$ image, the idea is to find a proper mapping function $\tau_{sa}$ between these two buffers and then use this mapping for the $L_{rv}$ solution where the virtual objects are visible. Our method can be summarized in five main steps. First, color-sample pairs are created. Second, to overcome issues with missing color samples, a heuristic is applied. Third, a mapping function is computed based on the collected color sample pairs. The fourth step performs temporal smoothing on the computed mapping function to avoid flickering artifacts, and finally, the mapping function is applied to the $L_{rv}$ and $L_r$ GI solution buffers.

### Color sample pair creation

We want good color matching for the virtual objects, but we can only create color sample pairs (denoted as $< C_r, C_{cam} >$) from the $L_r$ and $L_{cam}$ buffers where real objects are visible. Therefore, we need to find color samples in $L_r$ that are similar to the colors of the virtual objects.

In order to do this, we first collect $N$ color samples on virtual objects from the $L_{rv}$ buffer. In a second step, these samples are used to find similar color samples in the $L_r$ buffer. To compare the similarity between two colors, we use the following equations:

$$\Delta h = |h(c_a) - h(c_b)| \tag{4.20}$$

$$h_m = 0.5 - \min(\Delta h, 1 - \Delta h) \tag{4.21}$$

$$match = \frac{h_m}{1 + |c_a - c_b|} \tag{4.22}$$

where $c_a$ and $c_b$ are the two colors to be compared and $h(...)$ converts the color to HSV color space and returns the hue channel. Since hue is a circular value, $hue = 0$ equals $hue = 1$, and thus $h_m$ represents the minimal distance between both hue values. The $match$ function is designed to assign a high value to colors that are very similar in hue, but penalize colors that have an overall high color difference. The denominator is designed to avoid divisions by zero by adding one to the difference of the colors $c_a$ and $c_b$.

The screen-space coordinates $(x_i, y_i)$ of the pixel with the highest color match to the $i$th virtual object's color sample will be used to create a color sample pair $< C_r[i], C_{cam}[i] >$ from the $L_{cam}$ and $L_r$ buffers:

$$C_r[i] = L_r(x_i, y_i) \tag{4.23}$$

$$C_{cam}[i] = L_{cam}(x_i, y_i) \tag{4.24}$$

### Finding missing color samples

In the previous step, we assumed that the virtual objects have similar colors as the real objects. However, in typical AR applications, this will not always be the case, and then no suitable mapping function can be found for missing colors.

For these cases, we propose a heuristic that tries to cover missing colors by creating new color samples before the mapping function $\tau_{sa}$ is computed. The heuristic is based on the assumption that at least one dominant color (ranging from zero to full intensity) is available on

the real objects. So for this dominant color a meaningful mapping can be found. By simply swapping color channels for each color sample pair, the mapping characteristics of one color channel can be transferred to another one.

In other words the volume covered by a mapping function that relies only on the dominant color is very small. If the dominant color is distributed into the other color channels, the mapping function is forced to cover a larger volume and thus other colors can be mapped to $RGB_{cam}$ even though they are not directly available. Note that for these new colors, the exact mapping would be different. However, the main mapping properties are similar to the ones of the dominant color and thus, fewer visual artifacts appear than without any heuristic. For a given color sample pair the heuristic would randomly create a new one as follows (with $RGB \rightarrow GRB$):

$$c'_r.rgb = c_r.grb \tag{4.25}$$

$$c'_{cam}.rgb = c_{cam}.grb \tag{4.26}$$

**Finding a mapping function**

In the previous sections we created a set of $N$ color sample pairs that represent corresponding colors in the $RGB_{ref}$ and the $RGB_{cam}$ color spaces. In order to calculate a mapping function, we use polynomial regression. Polynomial regression is commonly used in camera calibration, or, in a more general sense, for input/output device calibration. It is described in more detail in the book by Kang [56]. For our method we implemented three different types of polynomials:

- $C_{cam}(r) = a_0 + a_1 r, C_{cam}(g) = a_0 + a_1 g, C_{cam}(b) = a_0 + a_1 b$
- $C_{cam}(r,g,b) = a_0 r + a_1 g + a_2 b$
- $C_{cam}(r,g,b) = a_0 + a_1 r + a_2 g + a_3 b$

Note that $a_*$ represent the coefficients calculated by the regression method.

**Temporal smoothing**

The samples are created in screen space and thus, changes in the view point or simply image noise may result in different color samples every frame. This causes temporal flickering artifacts between the adjacent frames. To dampen these sudden color changes, we temporally smooth the resulting mapping function on a per-component level. We use exponential smoothing [126] as follows:

$$a_i = ua_i + (1-u)a'_i \tag{4.27}$$

where $a_i$ are the component values after the regression, $i$ ranges from 0 to the number of used coefficients, and $a'_i$ are the values from the previous frame. The smoothing value $u$ is used to steer the influence of every new result value. In our case a value of 0.3 led to pleasing results while still getting fast adaption to camera parameter changes.

### 4.5.3 Statistics-Based Color Mapping

The method presented above has two drawbacks. First, we used color sample pairs to find a mapping function between the rendered global illumination solution of the real objects and the camera image. Due to tracking errors and camera lens errors it might happen that the two color samples taken from these buffers do not resolve to the same underlying object or color. Therefore, the mapping function $\tau_{sa}$ is prone to errors, which is not desirable. The second problem comes from the used color space. We used the *RGB* color space, which unfortunately turned out to be a bad choice because its color channels are correlated. This means that if the blue channel is high, chances are high that also the red and green channel values are high. This complicates any single-channel color modification as pointed out by Reinhard et al. [111]. Therefore, an uncorrelated color space is preferable to find a proper mapping function.

Please note that the remainder of this section was already published in a Bachelor thesis of Spelitz [131] under our supervision. However, we will briefly outline the method because it was used in one of our user studies (see Chapter 6).

To overcome the drawbacks of the previous method, this approach is based on image statistics rather than color sample pairs. The background to this work was presented by Reinhard et al. [111]. The idea of their method is to transfer color characteristics from one image onto another. Therefore, they perform a statistical analysis, they calculate the mean and standard deviation per color channel, for a source and a target image. Then the color characteristics of the source image are imposed on the target image using simple formulas. Note that they do not use the *RGB* color space because of the correlation between the color channels, but rather used the $l\alpha\beta$ color space [121]. We use this technique to transfer the color characteristics from the camera image to the $L_r$ and $L_{rv}$ buffers. We will denote the color transfer function as the statistics-based color mapping function $\tau_{st}$.

The differential rendering Equation 4.16 then needs to be altered as follows:

$$\Delta L = \tau_{st}(L_{rv}) - \tau_{st}(L_r) \tag{4.28}$$

$$L_{final} = L'_{cam} + \Delta L \tag{4.29}$$

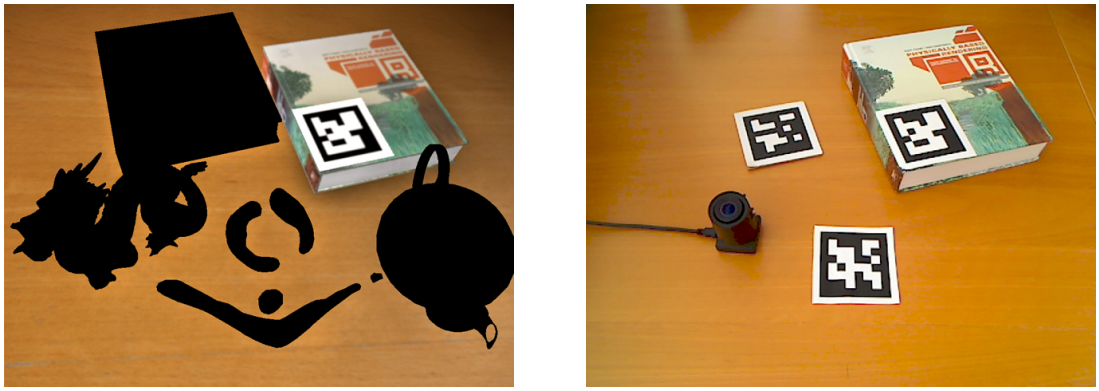
To overcome the aforementioned color-space drawback, the mapping function $\tau_{st}$ internally performs a color space conversion:

$$\tau_{st}(c.rgb) = f^{-1}\big(\tau_{ct}(f(TM(c.rgb)))\big) \tag{4.30}$$

where $f()$ is the color space conversion from *RGB* color space to the CIELab color space and $\tau_{ct}$ the actual color transfer function that we want to find. The statistics-based mapping function $\tau_{ct}$ works in low dynamic range. Therefore, we first perform tone mapping $TM$ on the color sample $c.rgb$.

#### The CIELab Color space

In the original work of Reinhard et al. [111], the $l\alpha\beta$ color space [121] was used because it minimized correlation between color channels for natural scenes. In a follow-up work, Reinhard



(a) (b)

**Figure 4.14:** This figure shows the $L_r$ buffer as the source image (a) and the camera image $L_{cam}$ as the target image (b). Note the black areas in the left image. To correctly calculate the mean and standard deviation these areas have to be excluded.

and Pouli [112] however found out that the CIELab color space with illuminant E performs best on average for different types of scenes. During their research they investigated images of natural day light scenes, manmade day light scenes, indoor scenes, and night scenes. Due to this insight, we also used the CIELab color space.

The CIELab color space is a device-independent color space and its three components are $L$, which represents the lightness from 0 (black) to 100 (white), $a$, which represents the opposing colors red and green, and color channel $b$, which represents the opposing colors blue and yellow.

To convert a color from device-dependent color space *RGB* to CIELab, it must be first transformed to the device-independent color space *XYZ*. However, since the white point of the display device is not known in most cases, an independent transformation is used where white in *RGB* color space is mapped to white in *XYZ* color space. The detailed equations and matrices necessary to transform from *RGB* color space to CIELab color space with illuminant E and vice versa can be found in the Bachelor thesis of Spelitz [131].

### Image statistics

In the color transfer method proposed by Reinhard et al. [111], the mean $\mu_s$, $\mu_t$ and the standard deviations $\sigma_s$, $\sigma_t$ must be calculated on a per color channel basis for the source image and the target image to impose the color characteristics of the source image to the target image. For our purpose the source image is the camera image $L_{cam}$ in CIELab (E) color space. The target image is the tone-mapped $L_r$ buffer also in CIELab (E) color space. In this way, the characteristics of the camera image $L_{cam}$ are applied onto the $L_r$ buffer.

To efficiently calculate the mean $\mu_t$ of the target image $L_{cam}$, we simple create a 1x1 mipmap

on the GPU of the camera buffer. The mean gets calculated as follows:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{4.31}$$

where $n = w \cdot h$ is the number of pixels. Since we calculate the mean using the 1x1 mipmap, we need the images in both dimensions ($w$ = $h$) to the power of two. We do this by simply scaling them accordingly using bilinear filtering.

For the source image, this is however not straight forward. As can be seen in Figure 4.14(a), the source image contains black areas. These black areas may appear because of two reasons. First, the pre-modeled real object representations are not defined in these areas or second, a virtual object appears in these parts. In order to correctly calculate the mean $\mu_s$, we have to compensate for these black areas. This is done in a very similar way as it was done during the center point calculation of our K-Means approach presented in Section 4.4. By taking the image mask into account, the mean $\mu_s$ of the source image can be calculated as follows:

$$\mu_s = \frac{\mu \cdot n}{n - n_{zero}} \tag{4.32}$$

where $n_{zero}$ is the amount of pixels in black areas. The same principle has to be applied for the calculation of the standard deviation. Since the standard deviation is the square root of the variance, we will calculate it as follows:

$$var = \frac{1}{n}\sum_{i=1}^{n} (x_i - \mu)^2 \tag{4.33}$$

Here, mipmapping is exploited again to calculate the variance in an efficient manner. For the $L_r$ buffer the black areas need to be taken into account and compensated as follows:

$$var = \frac{1}{n - n_{zero}}(var \cdot n - n_{zero} \cdot \mu^2) \tag{4.34}$$

Once the mean and standard deviations are calculated for the source and the target image, $\tau_{ct}$ can be calculated as follows [111]:

$$\tau_{ct}(c.Lab) = \langle \tau_L(c.L), \tau_a(c.a), \tau_b(c.b) \rangle \tag{4.35}$$

$$\tau_L(L) = \frac{\sigma_t^L}{\sigma_s^L}(L - \mu_s^L) + \mu_t^L \tag{4.36}$$

$$\tau_a(a) = \frac{\sigma_t^a}{\sigma_s^a}(a - \mu_s^a) + \mu_t^a \tag{4.37}$$

$$\tau_b(b) = \frac{\sigma_t^b}{\sigma_s^b}(b - \mu_s^b) + \mu_t^b \tag{4.38}$$

### 4.5.4 Limitations

The presented methods currently require a representation of the real scene, i.e., the geometry and BRDF estimation of real objects. This implies it does not work for conventional MR applications. Therefore, camera-based adaptive color mapping is tailored for MR systems that consider the real environment for rendering. It is also important to note that these methods do not work straight forward with the BRDF estimation method presented in Section 4.4 because the color mapping of the camera is not considered in the BRDF estimation.

An obvious limitation of the sample-based solution is the heuristic. It is based on the assumption that there is at least one dominating color on real objects and if that is not true, the heuristic will fail and visual artifacts may appear. The sample-based color mapping method also has difficulties when the tracking for real objects is too inaccurate. Then the see-through image will not exactly coincide with the GI solution for real objects ($L_r$) and therefore, wrong color pairs will be mapped to each other.

There are some special cases where both solutions will fail or produce undesired effects: First, in a situation where no real objects are visible because a virtual object completely occludes them, a color characterization cannot be calculated. Second, wrong color adjustment occurs when real objects are captured by the camera for which no representation for rendering exists. This usually happens when hands of interacting users are visible in the camera stream. Finally, if the discrepancy between real and virtual colors is too extreme, for example colorful virtual objects are placed into a gray-scale real environment, a mapping cannot be obtained.

### 4.5.5 Results

The PC used for the test results has an Intel Core2 Quad CPU 9550 at 2.8GHz with 8GB of memory. The graphics card is an NVIDIA Geforce GTX 580 with 1.5GB of dedicated video memory. The operating system was Microsoft Windows 7 64 bit, and the framework was developed in C#. All result images were rendered at a resolution of 1024x768 pixels using the DirectX10 API in conjunction with the SlimDX library.

We set up two different scenarios that should give an impression how our sample-based solution performs compared to simply using the tone-mapping operator from Reinhard et al. [113]. These two scenarios should represent two extreme cases. Scenario A shows a real color checker board (Gretagmacbeth - ColorChecker Digital SG) in which our method can find a large number of different color samples. Scenario B is more challenging, since the only real object visible is a wooden desk with a more or less uniform brownish color.

The implementation supports three different regression modes, which are listed in Section 4.5.2. Figure 4.15 shows a comparison between different polynomials for the regression of the sample-based method and the tone mapping operator by Reinhard et al. [113]. Furthermore, we have adjusted the parameters of the cameras and show the adaption to it by the proposed method. In camera settings A we reduced the saturation and increased the brightness. Setting B is the opposite, here the saturation was increased and the brightness decreased. Note that the implementation is not aware of any camera parameter settings. They were altered during runtime directly with the camera driver software. Furthermore, we used similar real and virtual objects for comparison but there is no need to have equal objects in the scenes. The frame numbers in

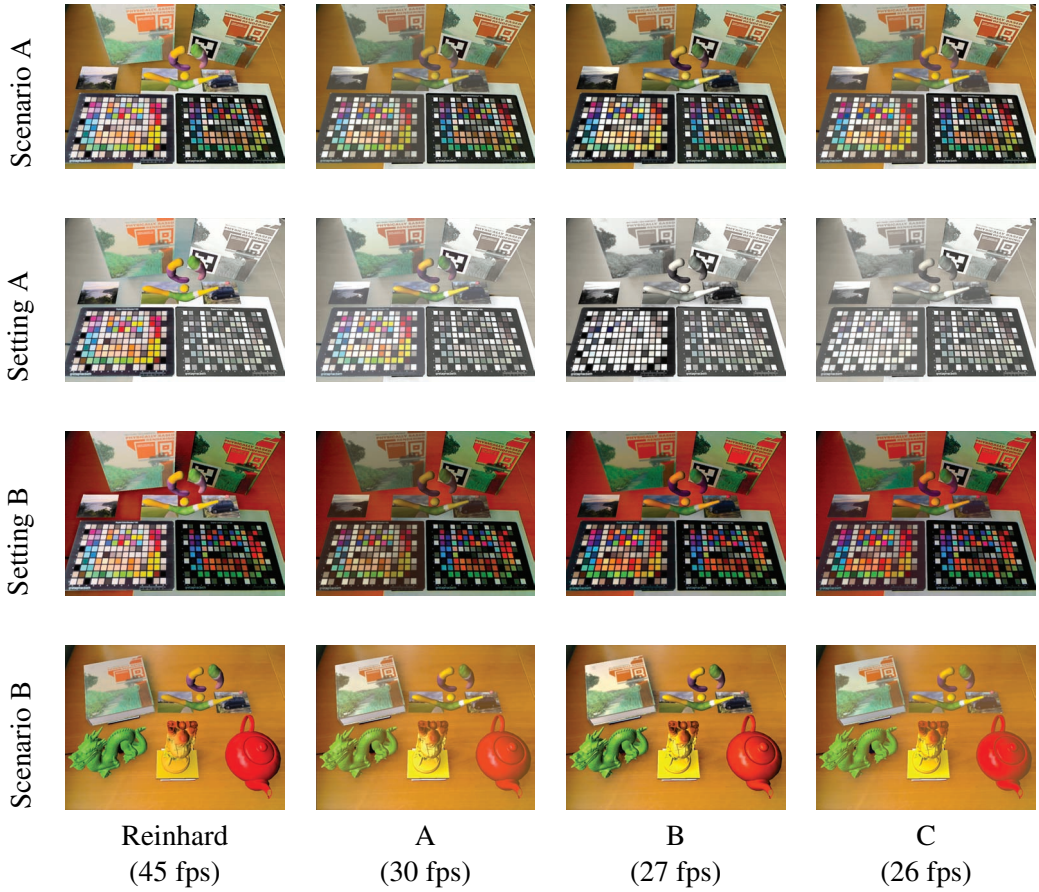



**Figure 4.15:** In these images the different regression modes are compared to each other using the two scenarios as well as two different camera parameter settings. Setting A has low saturation and high brightness. Parameter setting B has high saturation and low brightness. The tone-mapping operator from Reinhard et al. [113] does no approximation to camera colors at all, while the other modes do. Mode C yields the best approximation. The frame numbers in the brackets belong to Scenario A.

the brackets belong to Scenario A. The results show that regression mode C leads to the best results in comparison to the other regression modes.

Figure 4.16 shows a comparison between the tone-mapping operator from Reinhard et al. [113] (first column), our sample-based approach (second column), and the statistics-based approach (third column). The tone-mapping operator is slightly faster but does not adapt to the colors of the see-through image. Both presented solutions have similar visual quality and rendering speed. However, due to the limitations of the sample-based approach, the statistics-based approach is preferred.


**Figure 4.16:** These images show a comparison between the tone-mapping operator of Reinhard et al. [113], the sample-based color mapping approach, and the statistics-based color mapping approach. Five different input images for the see-through camera were used with normal settings (first row), increased contrast (second row), decreased contrast (third row), increased saturation (fourth row), and decreased saturation (last row) created using a graphics editing program based on the image from the first row. The book on the right side and the table including the two BCH markers, are real objects while all others are virtual ones.

### 4.5.6 Conclusion

In this section we presented two adaptive color-mapping methods that can be used in mixed reality applications. In both color-mapping methods, we calculate a color-mapping function that automatically adapts to the internal changes of the camera behavior every frame. In this way, the virtual objects have colors as if seen by the camera. To enhance the quality of the sample-based mapping method, we proposed a heuristic that allows our method to function even if the needed colors are not available in the image. Furthermore, we exploit temporal coherence between adjacent frames to temporally smooth the mapping function and thus get better non-flickering results. However, the sample-based mapping method suffers from low color-mapping quality if tracking errors occur. These limitations were addressed in the statistics-based method by calculating a color-mapping function on the basis of mean and standard deviation values between the rendered and the camera image. The results indicate that the statistics-based method produces slightly better results on average and is therefore preferred.

CHAPTER 5

# Reciprocal Shading for Mixed Reality

## 5.1 Introduction

In this chapter, we will present two methods to add virtual objects in a visually plausible way to a real scene at real-time frame rates. We present a method called *differential instant radiosity*, published in two papers by Knecht et al. [67, 68]. It combines differential rendering from Debevec [22] with instant radiosity from Keller [60]. The method is capable of rendering reciprocal effects, such as shadowing and indirect illumination from virtual on real and from real on virtual objects in an efficient way.

In the original form, differential instant radiosity does not support any reflective or refractive objects. In general, differential rendering is able to render reflective or refractive objects. However, the objects visible in the reflections or refractions are only the virtual representations of the real world objects. This means that no differential effects are applied to such objects, which may look disturbing. The method presented in Section 5.3, which was published in [70], proposes a solution to this limitation.

## 5.2 Reciprocal Shading for Mixed Reality

In this section, we present a global illumination (GI) rendering system that is designed to calculate the mutual influence between real and virtual objects. The aim of the GI solution is to be perceptually plausible without the ambition to be physically correct. In the comprehensive classification of Jacobs and Loscos [51], our approach would be placed in the "Common Illumination and Relighting" category. Besides calculating the influence between real and virtual objects, we are also able to relight the scene by virtual light sources.

There are some previous methods [34, 35] which are able to take indirect illumination into account. However, they need a computationally expensive preprocessing step or are not feasible for real-time applications. An important aspect of our work is an extension of instant radiosity to handle real-world objects. Instant radiosity has the advantage that it does not need any pre-

**Figure 5.1:** This figure shows a mixed reality scenario where a virtual table lamp illuminates the real and virtual objects causing indirect shadows on the real vase from the Buddha. The scene is reconstructed during run-time and rendered at 17.5 fps.

computation and therefore can be easily used for dynamic scenes where object positions change or the illumination is varied through user interaction.

To capture the surrounding environment, we use a fish-eye camera and the Microsoft Kinect sensor to create a representation of the real scene on the fly (see Chapter 4 for more details). Figure 5.1 gives an impression on how our method handles mutual shading effects in mixed-reality scenarios.

In this section, we present the following contributions:

- A modified instant radiosity approach that is usable for differential rendering, while shading is performed only once.
- New higher quality imperfect shadow maps by aligning the splats to the surface normal.
- Reduced temporal flickering artifacts by exploiting temporal coherence.
- Inconsistent color bleeding is avoided due to a special type of virtual point lights (VPLs).
- Double shadowing artifacts are eliminated.

### 5.2.1 Differential Rendering

Mixing real with virtual objects requires the calculation of a global illumination solution that takes both, virtual and real objects, into account. The idea of *differential rendering* [33, 22] is to minimize the influence of the BRDF estimation error by calculating only the "differential" effect of virtual objects, leaving effects of the original real BRDFs in the real scene intact. This requires calculating two global illumination solutions ($L_{rv}$ & $L_r$): one using both, virtual *and* real objects, and one using only the real objects. The final image is created by adding the difference $\Delta L$ between these two solutions to the captured camera image (for a graphical illustration see Chapter 3).

One might be inclined to calculate the difference between $L_{rv}$ and $L_r$ directly and add it to the captured real image. However, since the captured image has only low dynamic range, we first need to apply tone mapping (performed by function $TM$), which requires a complete image as input. Therefore, two complete images, including both direct and indirect illumination, need to be tone-mapped before calculating the difference:

$$\Delta L = TM(L_{rv}) - TM(L_r)$$

We first focus on direct illumination and single bounce global illumination, which in terms of Heckbert's [44] classification of light paths corresponds to $LDE$- and $LDDE$-paths, where L is the light source, D is a (potentially) glossy reflection at an object in the scene, and E is the eye. Furthermore, for specular reflections or refraction the symbol S is used. Figure 5.2 illustrates different light paths according to Heckbert's classification. In Section 5.2.6 we extend the method to be able to calculate multiple-bounce global illumination.

For a mixed-reality setting, the idea is that the illumination of the real scene $L_r$ is calculated by only considering the (outgoing) contribution $L_o$ of those paths where all elements are real, i.e., $L_rD_rD_rE$ paths. The contributions of all other paths, including those with a virtual light source ($L_vDDE$), a bounce on a virtual object ($LD_vDE$) ,or a virtual object to be shaded ($LDD_vE$), only count for the combined solution $L_{rv}$.

The solutions $L_r$ and $L_{rv}$ could be calculated separately using $L_rD_rD_rE$ and general $LDDE$ paths respectively. However, depending on how paths are generated, this might be very inefficient, and might also lead to artifacts if different paths are chosen for the two solutions. Instead, we assume that the paths are already given, and for each path, we decide whether it contributes to $L_r$ or $L_{rv}$ or both.

### 5.2.2 Instant Radiosity

In order to achieve real-time performance, we use instant radiosity to approximate global illumination. Instant radiosity uses virtual point lights (VPLs) to propagate light from the light sources. VPLs are created as endpoints of $LD$ (for the first bounce) or $DD$ (for subsequent bounces) path segments. The VPLs are used to shade every visible pixel on screen using a $DD$ path, so the paths creating them (previous $LD$ or $DD$ segments) are reused multiple times.

Figure 5.3 shows the main steps of our proposed differential instant radiosity method. The renderer is set up as a deferred rendering system and therefore one of the first steps is to render the scene from the camera and store all necessary information like position and normals in the

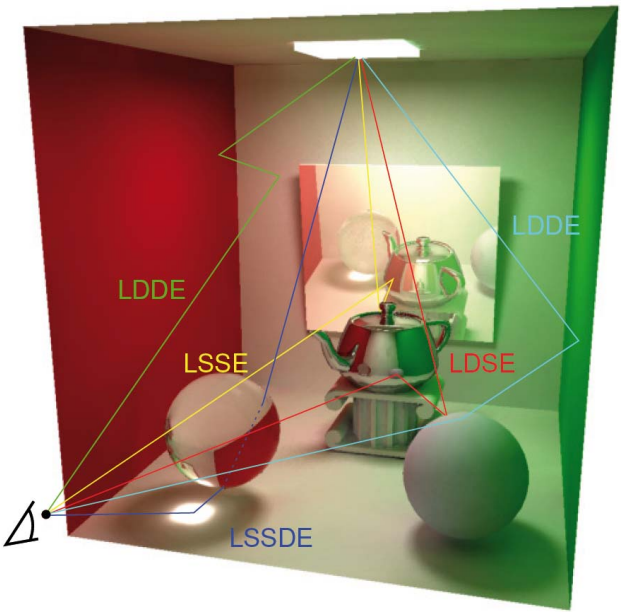



**Figure 5.2:** This figure shows different types of light paths that are annotated using Heckbert's [44] classification method (Image courtesy of Guerrero [39]).

so-called geometry buffer (G-buffer). The G-Buffer is then split as proposed by Segovia et al. [128] to save shading costs. The next step is to create VPLs and store them in a so-called VPL-Buffer. Depending on the primary light source type, the VPLs are created in different ways (see Section 5.2.7).

After two imperfect shadow maps are created, one only for the real objects and one only for the virtual objects, the illumination from the VPLs is calculated. The remaining steps are to perform tone mapping and finally, add the result to the video frame using differential rendering.

### 5.2.3 Differential Instant Radiosity

In instant radiosity, the *DD*-path from a VPL to a surface point (and likewise the *LD*-path from a light source to a surface point for direct illumination) is potentially blocked, and therefore, some of these paths have to be discarded after a visibility test carried out with shadow mapping. In a mixed-reality setting, the type of blocker for these bounces plays an important role when deciding to which GI solution a given light path should be added.

Suppose we have light paths as shown in Figure 5.4. Graph a) shows an $L_r b_v D_r E$ path and graph b) an $L_r D_r b_v D_r E$ path. In this notation, $b_v$ is some virtual blocking geometry. So for $L_r$, both light paths are valid, since virtual objects, including virtual blocking geometries, are not taken into account. However, the result is invalid for $L_{rv}$ since virtual objects are taken into account and therefore, the light path does not exist.

If we substitute the virtual blocking geometry $b_v$ with a real blocking geometry $b_r$, then the light paths cannot exist in both solutions and are simply ignored. So far a simple *visibility test*

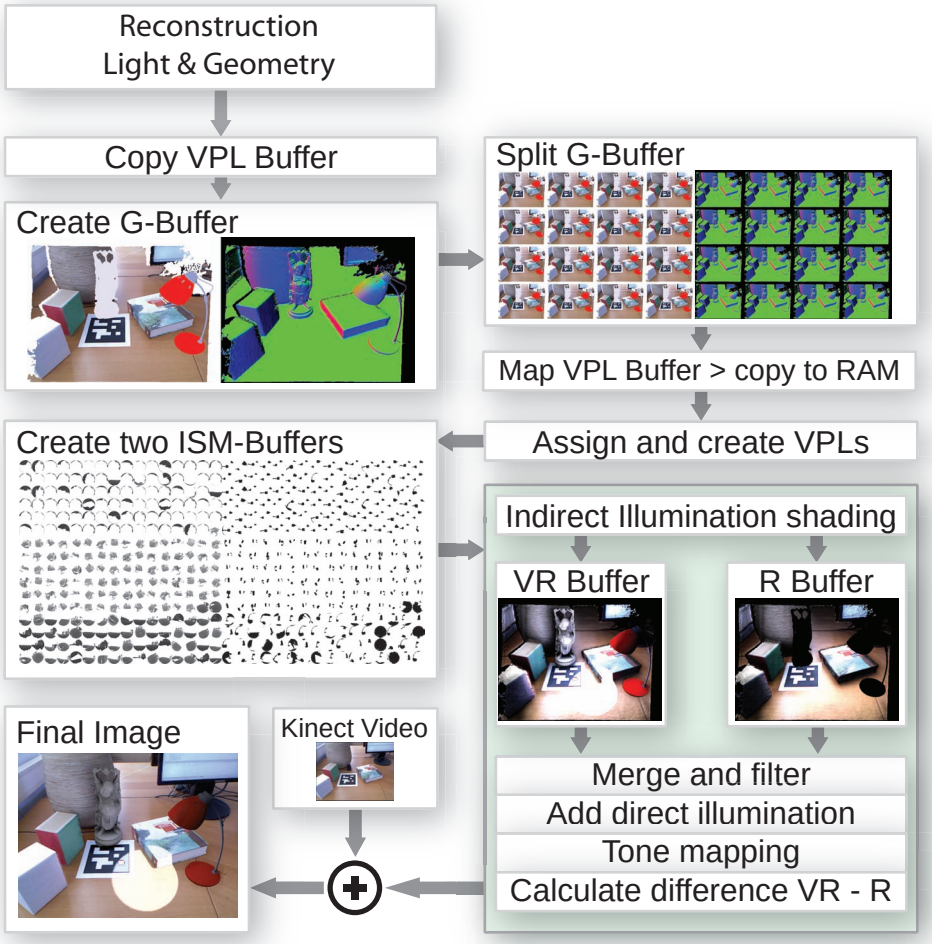



**Figure 5.3:** This figure shows the algorithm outline. The raw depth values from the Kinect sensor are preprocessed to improve the quality of the real scene reconstruction. Furthermore, a G-Buffer is created and afterwards split to reduce shading costs. The primary light sources generate VPLs from which ISMs are created. The VR- and R-Buffers are created in parallel, while the shading computation is performed only once. After some finishing steps like merging, adding direct illumination, and tone mapping, the video frame is added to the difference between the VR- and R-Buffers.

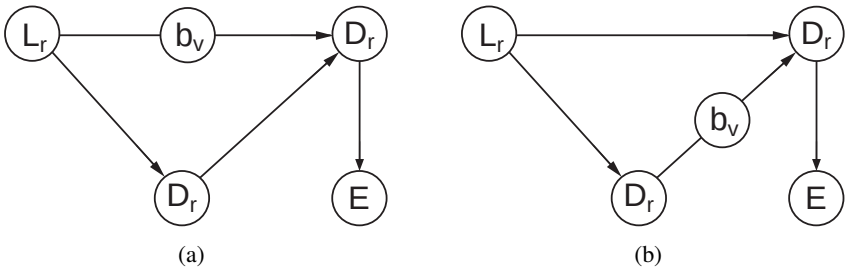



**Figure 5.4:** Graph (a) illustrates a virtual blocker in the $L_rD_r$ segment. The virtual blocker casts a shadow from direct illumination. Graph (b) illustrates a path that has a virtual blocker between the $D_rD_r$ segment. Here the shadow results from single bounce indirect illumination.

using shadow mapping and an additional flag whether the blocking geometry belongs to a real or a virtual object would be sufficient. However, if we have a path setup as shown in Figure 5.5, naive shadow mapping does not deliver enough information to correctly discard both light paths, since only the virtual blocking geometry is visible to the system. In our first differential instant radiosity solution [67], double shadowing artifacts, as shown in Figure 5.6, occurred because the method could only handle either a real or a virtual blocking geometry. Two or more blocking geometry parts behind each other were not supported. In this way, the reflective shadow map of the light source only had information about the front most object – the green virtual cube, which resulted in double shadowing artifacts.

To overcome this limitation, we propose to use two shadow maps for real primary light sources and for the VPLs. Real primary light sources create two reflective shadow maps (RSMs) [18]. One RSM always stores only the real objects and the other RSM only the virtual objects. By having two RSMs, the visibility test needs two shadow map lookups, but double shadowing artifacts are avoided. Note that for performance reasons, two RSMs are only created for real primary light sources like a spot light. If the spot light is set to be virtual, no double shadowing can occur anyway, and using one RSM, as described in Knecht et al. [67], is sufficient.

Since VPLs can be placed on real and on virtual objects, we always need two imperfect shadow maps [117].

So far only blocking geometry was taken into account to decide to which GI solution the light path should be added. However, it also depends on whether the bounces of a light path belong to real or to virtual objects.

Suppose we have a real spot light that illuminates a virtual green cube and a real blue cube placed on a real desk as shown in Figure 5.7.

Here the inserted virtual green cube causes four types of light interaction. First, light that hits the real desk causes color bleeding on the virtual green cube (VPL 1, $L_rD_rD_vE$). Second, light that hits the virtual green cube causes color bleeding on the real desk (VPL 2, $L_rD_vD_rE$). Third, the virtual green cube casts a shadow on the desk illustrated by ray $R$ ($L_rb_vD_rE$). Fourth, the real light arriving along ray $R$ in the real scene causes real color bleeding on the real desk, thus a new VPL is placed on the real blue cube (VPL3 $L_rb_vD_rE$). Since this area lies in the shadow of the virtual green cube the real color bleeding has to be eliminated, which is done by

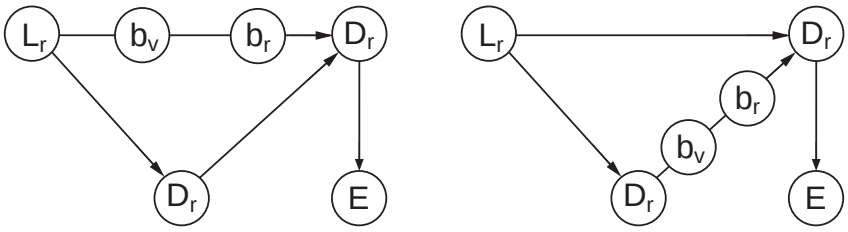



**Figure 5.5:** Graphs (a) and (b) show situations where naive shadow mapping cannot handle correct visibility. In contrast to our first approach [67], the extended method [68] is able to correctly evaluate these paths.

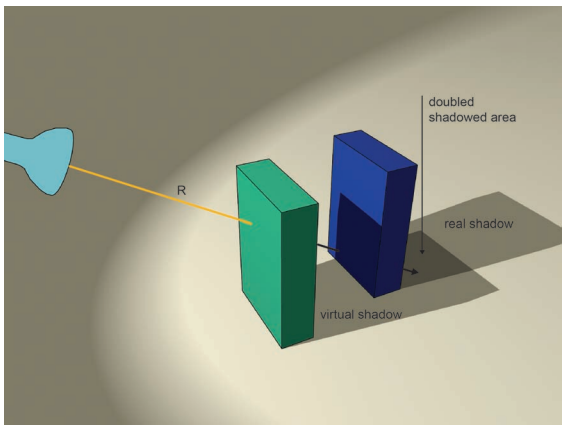



**Figure 5.6:** The spotlight and the blue box are real objects and therefore a real shadow is cast onto the desk. In [67] the reflective shadow map of the light source only had information about the front most object – the green virtual cube, which resulted in double shadowing artifacts.

this special VPL during the differential rendering step.

Suppose we shade a point on the virtual green cube, illuminated by VPL 1. We have to decide if the path contribution should be added to $L_{rv}$ and/or to $L_r$. The spotlight is real, VPL 1 is placed on the real desk, and the shaded point corresponds to a virtual object ($L_r D_r D_v E$). In this case the result gets added to $L_{rv}$, but not to $L_r$, as there is a virtual object in the light path. When shading a point on the real desk, which gets illuminated by VPL 2, the result must again be added to $L_{rv}$, but not to $L_r$, because VPL 2 is placed on a virtual object ($L_r D_v D_r E$).

On the other hand, when a point on the real desk is shadowed by the virtual green cube ($L_r b_v D_r E$), the outgoing radiance must be added to $L_r$ but not to $L_{rv}$.

VPL3 ($L_r b_v D_r E$) is a special type of virtual point light source. VPL3 will only be used to illuminate real objects. For virtual objects, this light source is completely ignored. The placement and special treatment of VPL3 overcomes the artifact of inconsistent color bleeding, a limitation of our first approach [67]. The light caused by VPL3 will be added only to the $L_r$

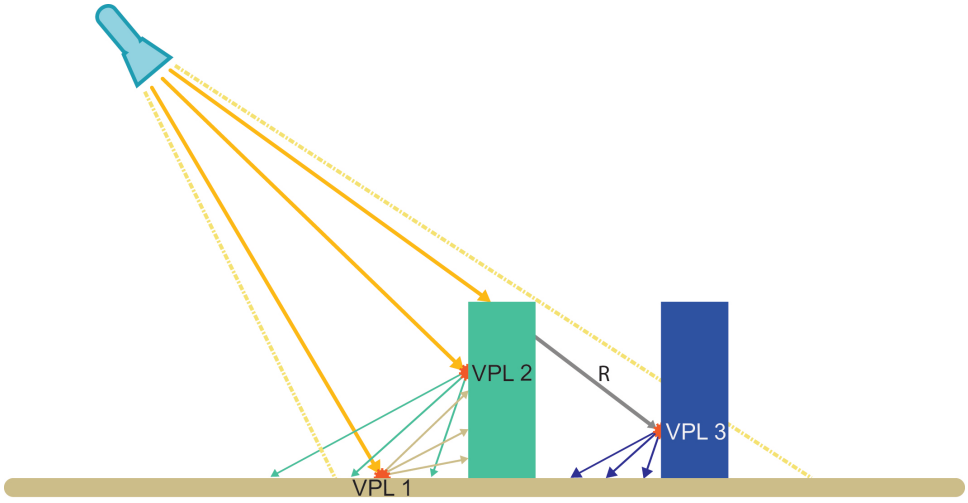



**Figure 5.7:** Illustration of an augmented scene with a real desk, a virtual green cube, a real blue cube, and a real spot light illuminating the scene. VPL 1 will cause color bleeding from the desk onto the green cube – VPL 2 from the green cube onto the desk. Ray *R* illustrates the shadowing caused by the virtual green cube. Light arriving at ray *R* causes real color bleeding from the real blue cube onto the real desk. VPL 3 cancels that color bleeding out. Note that VPL 3 only illuminates real objects.

solution. After differential rendering is applied that light will cancel out the real color bleeding, which is visible in the video frame.

### 5.2.4 Direct Bounce Computation

The direct lighting contribution (*LDE* paths) is calculated for primary light sources, i.e., using a spot light.

Let $r(x)$ be a function that returns 1 if element $x$ is associated to a real object, and 0 if not, where $x$ can be one of the following: a primary light source, a VPL, a surface point, or a light path (see Section 5.2.6).

- For a primary light source $pl$, $r(pl)$ is stored with the light source.
- For a VPL, $r(\text{VPL})$ is set when the VPL is created (see Section 5.2.7).
- For the point to be shaded $p$, $r(p)$ is taken from the object definition.

Depending on visibility, direct light paths *LDE* need to be added to $L_r$ and/or $L_{rv}$. To perform the *visibility test*, we define the visibility test $V_r(pl, p)$ as a shadow map lookup against real shadow casters and $V_v(pl, p)$ as a shadow map lookup against virtual shadow casters for a given primary light source. $L_o$ is the unobstructed radiance from surface point $p$ in viewing direction. For a given surface point $p$ in the G-Buffer, the illumination is calculated as follows:

$$L_{rv}^{pl} = L_o^{pl} V_{rv}(pl,p) \quad (5.1)$$
$$L_r^{pl} = L_o^{pl} r(pl) r(p) V_r(pl,p) \quad (5.2)$$
$$V_{rv}(pl,p) = min(V_r(pl,p), V_v(pl,p)) \quad (5.3)$$

The contributions for *LDE* paths are simply summed up, as follows, for each primary light source:

$$L_{rv} = \sum_{pl} L_{rv}^{pl} \quad (5.4)$$
$$L_r = \sum_{pl} L_r^{pl} \quad (5.5)$$

### 5.2.5 Indirect Bounce Computation

For indirect light bounces, we need to calculate $L_r$ and $L_{rv}$ for *DDE* paths, i.e., for a point $p$ illuminated by the $i^{th}$ VPL, which is placed at the end of light path segment *LD*. To perform the *visibility test* for the $i^{th}$ VPL, the shadow test functions are applied in a similar way as in the previous section, but the imperfect shadow maps related to the $i^{th}$ VPL are used. $L_o$ is the unobstructed radiance, i.e., without taking visibility for the VPL into account. $ir(\text{VPL})$ returns 1 if the VPL should only illuminate real objects and 0 otherwise (see Section 5.2.3). Then, for the $i^{th}$ VPL,

$$L_{rv}^{i} = L_o^{i}(1 - ir(\text{VPL}^i)) V_{rv}(\text{VPL}^i, p) \quad (5.6)$$
$$L_r^{i} = L_o^{i} r(pl) r(\text{VPL}^i) r(p) V_r(\text{VPL}^i, p) \quad (5.7)$$
$$V_{rv}(vpl,p) = min(V_r(vpl,p), V_v(vpl,p)). \quad (5.8)$$

The contributions for *DDE* paths are summed up in a similar way as shown in Equations 5.4 & 5.5.

### 5.2.6 Multiple Bounce Computation

Extending differential instant radiosity to multiple bounces is straightforward. Since in instant radiosity, each light bounce is simulated by one VPL, we have to forward the information of the previous light path to the next VPL. In this way, it is possible to calculate the influence on the $L_{rv}$ and $L_r$ solutions at the last bounce correctly. Note that in this section we only focus on the decision process to which GI buffer the contribution of a VPL should count. The creation of VPLs causing additional light bounces will be described in Section 5.2.7.

There are two flags that are of interest: First, a flag $r(x)$ whether the previous light path only contained real objects and second, $ir(\text{VPL})$, whether only real objects should be illuminated by the given light path. Imagine that light bounces off several times from real surfaces coming from a real primary light source illuminating a real object. In this case the whole light path interacts only with real objects, and the same amount of light will be added to $L_{rv}$ and $L_r$. However, if

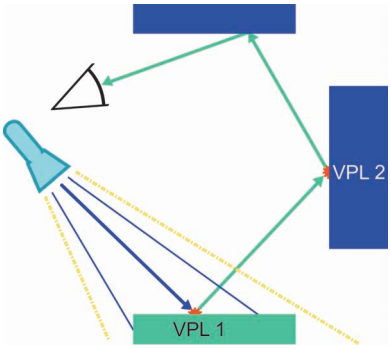



**Figure 5.8:** After encountering the first virtual object (shown in green), the light path is flagged to contain virtual objects (also visualized in green). Note that this information is forwarded to subsequent VPLs.

there is only one bounce on a virtual object, as illustrated by the green object in Figure 5.8, the final influence on $L_{rv}$ and $L_r$ differs. The forwarding of the changed flag $r(x)$ is illustrated by the change of the light path color after VPL 1.

The flag about the primary light source $pl$ will be substituted by a flag that indicates if the previous path $\bar{x}$ included any virtual objects or a virtual primary light source. When a new VPL is created from a previous one, the flag is calculated as follows:

$$\bar{x} = r(\bar{x}_{prev})r(p_{VPL}) \tag{5.9}$$

$$\bar{x_0} = r(pl) \tag{5.10}$$

where $\bar{x}_{prev}$ indicates if the path to the preceding VPL only belonged to real objects, and $p_{VPL}$ indicates whether the new VPL is created on the surface of a real or a virtual object. Once a virtual object was in the light path, the path flag will always be 0. The new illumination equation just uses $\bar{x}$ instead of $pl$.

Furthermore, the flag $ir(\text{VPL})$, which indicates whether only real objects should be illuminated, is directly copied without modification to the next VPL. Note that virtual objects on the path to the next VPL are ignored if the previous VPL has the flag set ($r(p_{VPL}) = 1$), since it must be placed on a real object.

### 5.2.7 Primary Light Source Types and VPL Creation

So far we have assumed that the light paths are already given and thus the VPLs are already placed in the scene. In our system, all VPLs are created by the primary light sources and there are three types currently supported, which will be described in the follow up sections: A spotlight, an environment light, and a special primary light source that performs one light bounce.

In contrast to our first approach [67], three different modes of VPL distribution between the light sources are available. One mode just distributes an equal amount of VPLs over all primary light sources. If a user, however, wants a particular light source to get more VPLs than

the other primary light sources, user specific-weights can be assigned. The third mode takes the summed flux of the VPL buffer from the previous frame on a per primary light source basis and estimates the new amount of VPLs according to the summed flux. In practice we use the distribution mode which gives each primary light source the same amount of VPLs. However, for scenes with glossy surfaces and spot lights that should cause nice glossy reflections, user driven weights are preferred.

### The Spot Light Source

The spot light source behaves like a standard spot light except that it can be set to be a real or virtual light source. If the spot light is set to be virtual, it stores a single reflective shadow map that is rendered from the point of view of the light source. Beside the depth, which is used for standard shadow mapping, it stores the surface color, the material parameter, the normal of the surface, and an importance factor similar to reflective shadow maps (RSM) [18]. When VPLs are created, the importance factor is used to perform importance sampling on the RSM as proposed by Clarberg et al. [12]. After a proper sample position has been found, the information from the RSM is used to create a new VPL in the VPL-Buffer.

However, if the spot light is set to be real, we have to be aware that we need to cancel out inconsistent color bleeding. Figure 5.7 illustrates this effect with VPL 3. In this case a virtual object is in front of the real one and it would not be possible to place VPL 3 there if only one RSM were used. So if the spot light is set to be real, two RSMs are created each frame. The spot light however, has only a certain amount of VPL slots available in the VPL Buffer. Therefore, we need a proper way to distribute the VPLs over the two RSMs. As a metric we use the number of pixels covered by real or respectively virtual objects in the RSMs. Their sum is normalized to get a valid probability value. A pseudo-random Halton value is then used to decide which of the RSMs should be used to create a new VPL. The flag $r(VPL)$ is set with respect to the chosen RSM. Note that since the VPLs have a given flux, but only a subset of VPLs are applied on either of the RSMs, we must compensate their flux according to the distribution of VPLs between the two RSMs.

### The Environment Light Source

The environment light source uses the input image from a fish-eye lens camera to capture the surrounding illumination. It does this by placing virtual point lights on a hemisphere around the scene center. Figure 5.9 shows the virtual point lights placed on the hemisphere. To get a better approximation, the VPLs are first importance sampled according to the illumination intensity using hierarchical warping [12]. Since the environment light source uses the image from the camera, it is set to be a real light source. Note that the environment light is different to the spot light as it uses the VPLs for direct illumination.

### The Multiple Bounce Light Source

The third primary light source is a special type since it performs one additional light bounce. The idea behind it is to use the geometry itself as light sources, i.e., the VPLs already placed in

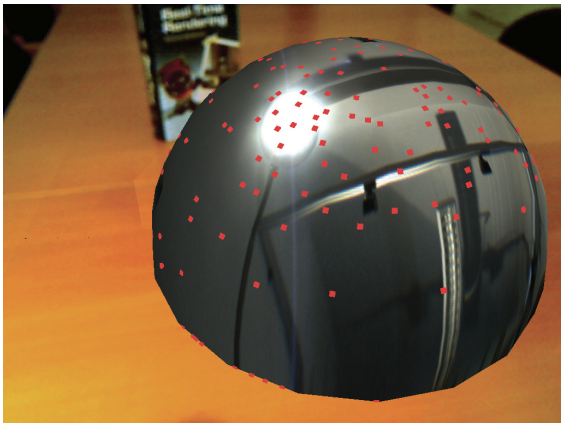


**Figure 5.9:** Illustration of incident illumination from the surrounding environment captured with the fish-eye lens camera. The red dots show the positions of the VPLs.

a previous bounce. For this type of light source, two different approaches were implemented. Both approaches perform three steps, but differ in the way how the last step is performed.

Similar to the other primary light sources, the multiple bounce light source has a given amount of VPL slots available. So in the first step, a source VPL from the VPL-Buffer of the previous frame is selected and linked to a VPL slot. Note that a source VPL will only be used to spawn a new VPL. It gets linked to a VPL slot but not written into it. In this way, whenever a candidate VPL location that is assigned to a given VPL slot is evaluated, the linked source VPL is the light source that illuminates the surface point where the candidate VPL would be placed. The selection process of a source VPL is importance driven. This means that the higher the flux of a VPL of the previous VPL-Buffer is, the more likely it will be selected and linked to a VPL slot and therefore to spawn a new VPL.

The yet to be created VPLs which will cause an additional light bounce will be placed on the surface of objects in the scene. Therefore, we use the point clouds normally used for ISM creation to find possible new VPL locations. In the second step, each of these points is assigned to one of the available VPL slots. In this way, we have for each VPL slot a set of points which are candidates for a new VPL location.

Since only one VPL can be created per VPL slot, but multiple candidates exist, the last step evaluates the suitability of each point as a new VPL location and then selects the one which fits best. In order to select the best VPL location according to a fitness function, the depth test of the GPU is used. Our two multiple bounce methods differ in the way how they evaluate the fitness of each VPL locations.

In our first approach [67], a glossy light bounce from the source VPL onto a candidate VPL location is calculated. For visibility testing, the ISM textures from the previous frame are used, since the source VPL is also from that frame. The maximum outgoing radiance will then be used as a measurement for the fitness of the candidate VPL location. Note, however, that certain light paths may occur more often than in an unbiased bounce schema, which can lead to overestimated indirect illumination. Figure 5.18(d) shows such an example, where a lot of VPLs are created on

the ceiling of the Cornell box because the source VPLs which are created on the ground plane only „see“ the ceiling. Therefore, the red wall gets too bright in the top corner.

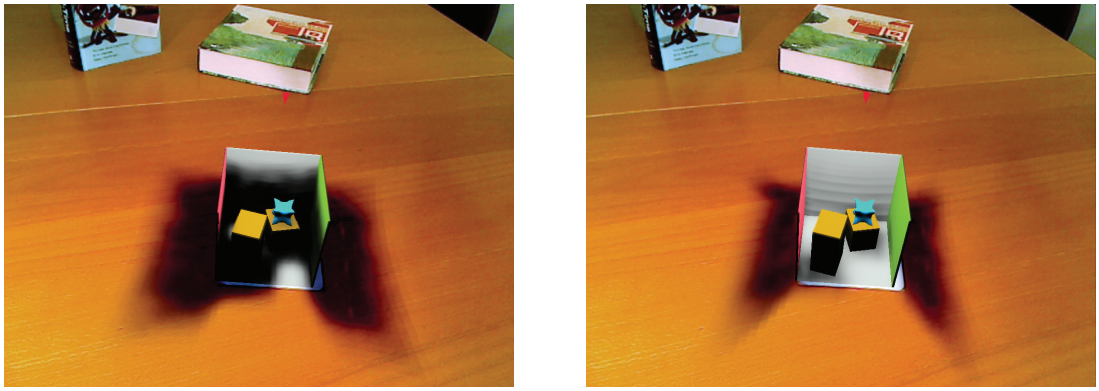
To overcome this issue, we developed an improved, new multiple bounce method. The idea of this new method is to find a VPL location that is located in a certain direction related to the source VPL. This is somehow similar to ray tracing where a ray is shot into a specific direction and the ray hit would be a new VPL location. However, since we cannot shoot rays we want to find surface points that are located as near as possible at the hypothetical ray hit position. Therefore, we only accept candidate VPL locations which are located in a similar direction to the ray direction and furthermore, we choose the candidate VPL location which is closest to the source VPL location to be the fittest.

In order to find such a new VPL location, the evaluation step in the new multiple bounce method first creates a quasi-random direction vector $\omega_o$ for each VPL slot. $\omega_o$ is created in such a way that directions which are more important according to the BRDF characteristics of the source VPL are emphasized.

To create such an importance-driven direction vector $\omega_o$, first, a quasi-random value is used to decide whether a direction vector for the diffuse or the specular lobe should be created – the probabilities depend on the diffuse and specular intensity values. If the diffuse lobe is chosen, we simply create a random uniformly distributed direction $\omega_o$ over the hemisphere. However, if the specular lobe is chosen, $\omega_o$ should mainly point into a direction that is similar to $\omega_{ref}$, the reflected incoming light direction (from a previous light source) at the source VPL. The variation of $\omega_o$ with respect to $\omega_{ref}$ depends on the material characteristics, more precisely, the specular power of the surface at which the source VPL is located. To create such a vector, three steps are necessary. First, we create a uniformly distributed direction $\omega_o$ over the hemisphere with respect to the normal vector of the source VPL. Then $\omega_o$ gets squeezed towards the VPL's normal depending on the specular power of the underlying surface. We do this by using a precalculated texturemap. At this point, $\omega_o$ resembles a specular lobe that is oriented along the VPL's normal. In the last step, $\omega_o$ will be rotated in such a way that it is oriented along $\omega_{ref}$. Note that the candidate VPL location $p$ will be discarded if $\omega_o$ points into the surface, which can happen at very flat reflection vectors $\omega_{ref}$.

Once $\omega_o$ is calculated, the direction $\omega_p$ from the existing source VPL pointing towards a candidate VPL location $p$ is calculated. In the next step, the angle between direction vector $\omega_o$ and $\omega_p$ is calculated. If it is larger than a certain threshold (we set the threshold to five degrees), the candidate VPL location will be discarded. However, if the angle between $\omega_o$ and $\omega_p$ is below the threshold, candidate VPL location $p$ is assumed to be valid. The last step is to calculate the distance from the source VPL towards the valid candidate VPL location $p$ and write that as a depth value into the depth buffer. The depth test is set up like in standard rendering, where only the front most, in this case the nearest candidate VPL location, should survive. Because we use only the nearest candidate VPL location in a specific direction, chances are high that this candidate VPL location is valid. Therefore, we avoid an additional shadow test. However, please note that due to this simplification our method can produce wrong results. Figure 5.19(c) shows an example image rendered with our new multiple bounce method, and Section 5.2.13 shows comparisons of our multiple bounce methods to a ground-truth solution.

So the difference between the two methods is that in the first method, the VPL with the



(a) (b)

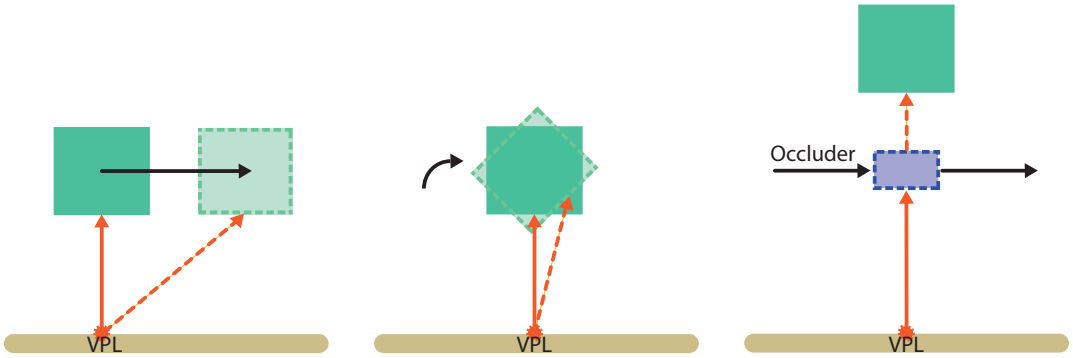
**Figure 5.10:** Image (a) shows the result with standard imperfect shadow maps, when 256 VPLs, shown in red, are placed at the same position on top of the Cornell box. Here, point splats are rendered as billboards into the ISM resulting in a lot of wrong self occlusions. As shown in image (b), our new method, which splats the points aligned to the surface normal into the ISM, removes most of the self-shadowing artifacts.

highest radiance value is chosen, whereas in the new method, the VPL that is closest among those that match a properly importance-sampled direction is chosen.

### 5.2.8 Geometry-Aligned Point Splats for ISMs

In the original ISM approach, the point sprites were splatted into ISMs as billboards. Their size was scaled by the inverse squared distance, so that a splat nearer to the view point would have a larger area of influence. However, this is not a good geometrical approximation of the original geometry and may lead to self-shadowing artifacts. We propose a method where the point splats are aligned to the corresponding surface normal and thus greatly reduce wrong self occlusion.

Instead of using screen-space splats, we define the splat in tangent space of the point sample, and transform all four corner vertices of the splat into the ISM coordinate system. To cover the complete upper hemisphere of a VPL for shadow mapping, Ritschel et al. [117] use parabolic mapping and in their implementation only the center point was subjected to parabolic mapping. In contrast, we use a cosine mapping function and map each corner vertex into the ISM. Therefore, the size of the point splat is implicitly increased or decreased depending on the relative position of the point splat to the VPL. Figure 5.10 shows a comparison of the shadowing results. In this scene 256 VPLs, shown in red, are all placed at the same position on top of the Cornell box. In cases where the VPL direction is normal to the surface normal, a lot of self occlusion occurs with the standard method (a). With our approach the depth values in the ISM are more accurate and thus less self occlusion occurs (b).




(a) (b) (c)

**Figure 5.11:** This figure shows three cases which are taken into consideration when the confidence value is calculated. Movement (a) or rotation (b) of an illuminated object reduces the confidence value and the newly calculated illumination is preferred. Illustration (c) shows how an occluder may cause illumination changes between two frames and therefore also lowers the confidence value.

### 5.2.9 Reducing Temporal Flickering

To get high frame rates, we are forced to keep the number of VPLs as low as possible, while at the same time the image quality should not be decreased too much. For this reason we temporally smooth the illumination results caused by the VPLs, exploiting the temporal coherence between adjacent frames. We do this by storing the illumination buffer from the last frame and reuse the calculated illumination. However, since the objects are moving, and the illumination and the camera may change, we have to calculate how confident an indirect illumination value from the previous frame is. We therefore take three different parameters into account as illustrated in Figure 5.11:

(a) The relative position change from one frame to the next. For performance reasons we divide the position change into two parts. The difference in depth per screen pixel and the difference in 2D screen space.

(b) The difference between the surface normals. If the normal changes, illumination will also change and hence, the previous values are not as confident as the new ones. We calculate the difference using the dot product between the normals.

(c) Global illumination is a global process. So it may happen that a point at a given pixel does not move (no change in position and normal) but the confidence should still be low because, due to other moving objects, the illumination changes a lot. Therefore, the difference of the previously calculated illumination and the current illumination also reduces the confidence. We use the cubed difference so that low differences have little impact and higher differences have more.

The confidence for each pixel is calculated as follows:

$$\varepsilon_{pos} = ||(x_s - x_{prev}, y_s - y_{prev}, d_s - d_{prev})w_p|| \tag{5.11}$$

$$\varepsilon_{normal} = (1 - (n \cdot n_{prev}))w_n \tag{5.12}$$

$$\varepsilon_{ill} = \text{saturate}(||I_{new} - I_{prev}||^3)w_i \tag{5.13}$$

$$confidence = \text{saturate}(1 - \max(\varepsilon_{pos}, \varepsilon_{normal}, \varepsilon_{ill}))c_B \tag{5.14}$$

where $[*]_{prev}$ always directs to values from the previous frame, $(x_s, y_s)$ is the screen position, $d_s$ the screen depth, $n$ the normal of the screen pixel, and $c_B$ is the base confidence. $I_{new}$ is the indirect illumination calculated in this frame. The weighting factors $w_*$ depend heavily on the scene characteristics. For the Cornell box, we have chosen the following weights: $w_p = (10, 10, 10000)$, $w_n = 1.0$, and $w_i = 0.025$.

Note that $w_p$ is a vector so that we are able to weight depth movement independent of screen space movement. If $w_p$ and $w_n$ have smaller values, ghosting artifacts start to show up. Parameter $w_i$ is a little bit more difficult to choose. If it is too low, illumination updates may take too long. On the other hand, if it is too high, the indirect illumination difference between adjacent frames tends to be too high, so the confidence will be low and thus flickering artifacts start to show up again. This is especially important when multiple bounces are calculated where the VPL placement is rather incoherent.

We also tested the confidence-value computation using different equations like multiplication of $\varepsilon_{pos}$, $\varepsilon_{normal}$ and $\varepsilon_{ill}$. However, we got the best results when using the maximum of these values. Furthermore, the saturation function ensures that the confidence value is between zero and one.

The final per-pixel indirect illumination for the current frame is:

$$I = confidence I_{prev} + (1 - confidence)I_{new} \tag{5.15}$$

### 5.2.10 Implementation

The framework is designed as a deferred rendering system, which enables us to decouple scene complexity from illumination computation by using a so-called G-Buffer. The diagram in Figure 5.3 shows an overview of the necessary steps that are performed every frame. We also use single-pass interleaved sampling from Segovia et al. [128] to reduce shading costs.

#### System Overview

The first step is to copy the VPL-Buffer of the previous frame to mappable textures. This is necessary because the VPL-Buffer itself is used as a render target (during VPL creation) and current graphics hardware does not allow for CPU read back of those kind of textures.

The G-Buffer is created afterwards. It stores the color, diffuse intensity, specular intensity, and specular power. Furthermore, it stores the normal at a screen pixel and the movement in screen space compared to the last frame (see Section 5.2.9). To calculate the world-space position in every pixel, the depth is also stored. The G-Buffer also needs an indicator if the current pixel belongs to a real or virtual object. In our implementation the sign of the depth

value is used for this. For real objects, the depth value is positive and for virtual ones it is negative.

Ritschel et al. [117] as well as Laine et al. [73] use interleaved sampling by Segovia et al. [128] to only assign a subset of VPLs to a given pixel. We therefore split the G-Buffer. After splitting, the data in the textures, which were copied in the first step, must be copied to system memory.

After the new VPLs have been created, the imperfect shadow maps, as described in Section 5.2.8, are rendered. Each object in the scene has a corresponding point cloud, and these point clouds are used to create the ISMs. The point clouds themselves are created at loading time by simply distributing a given number of points over the total surface area of all objects. Each point gets a unique id which is used to assign the point to a VPL.

Once the VPL-Buffer is created and the imperfect shadow maps are available, illumination computation can start. In Figure 5.3, these steps are surrounded by a green box. The output of the green box is a difference image that will be applied to the video frame. All other steps in the green box render into two buffers simultaneously, the *VR-Buffer* and the *R-Buffer*. The VR-Buffer stores the complete GI solution including real and virtual objects. The R-Buffer only stores the GI solution computed from the real objects. Note that these buffers are double buffered because simultaneous read and write operations are not possible.

First, indirect illumination is computed for both buffers. This is done by drawing a mesh consisting of quads that corresponds to the split G-Buffer in screen space. The mesh has the same amount of quads as there are VPLs in the VPL-Buffer. Each quad has a unique id that is used to lookup the corresponding VPL in the VPL-Buffer. Note that both buffers are manipulated in parallel, and indirect illumination is accumulated in one single draw call. In this shading process, the flags are used to determine how to calculate the illumination on a per-pixel basis.

Afterwards the split buffers are merged. During the merging step, the results from the previous frames are reused to temporally smooth the indirect illumination calculation and filtered. Then, for each primary light source direct illumination is added. The resulting buffers must be tone mapped using the operator from Reinhard et al. [113] or one of the color mapping methods from Chapter 4 should be applied. The resulting VR-Buffer and R-Buffer contain illumination caused by the VPLs and primary light sources as shown in Figure 5.3. In the last step, the background image is added to the difference between the VR- and R-Buffers. Note that the background image is masked and only added where there are no virtual objects.

#### Kinect Data Integration

While all previously shown figures used premodeled geometry, all figures in Section 5.2.12 use the Kinect sensor to reconstruct the geometry. Therefore, the acquired data of the Kinect sensor (see Chapter 4) must be tightly integrated into the differential rendering system. It must be added into the G-Buffer and should be usable for depth-map creation. The world-space position map and the normal map are directly rendered into the G-Buffer. Furthermore, the G-Buffer must be filled with Phong BRDF estimation parameters of the real scene's materials. For performance reasons, we did not perform any Phong BRDF estimation as described in Chapter 4, so we set the diffuse intensity $(D_I)$, specular intensity $(S_I)$, and specular power $(S_P)$ to values that are at least a usable approximation. The specular component is set to zero and the diffuse intensity

is set to the input image color from the camera stream of the Kinect sensor. Although these *RGB* values are a convolution with the incident light, they can be used to a certain degree for our purposes. Note that the error in the BRDF estimation for real objects only influences the differential effect and not the absolute values in differential rendering. However, the error can still get arbitrarily large.

As described in Section 5.2.3, imperfect shadow maps are used for visibility tests with the VPLs. Therefore, all the objects need to have a point cloud representation, including the data from the Kinect sensor. However, the data from the Kinect sensor is stored in 2D texture maps, and cannot be used straightforwardly as a point cloud. Our main idea to solve this problem is to treat each pixel in the 2D texture map as a vertex.

Therefore, we use a vertex buffer with the doubled size of $1280 \times 960$ vertices to have enough splats for the ISMs. For each vertex lookup, coordinates are stored that are used to lookup the position map. Furthermore, each vertex gets assigned a unique id by the graphics card while rendering. During creation of the ISMs, the vertex buffer is rendered. In the vertex shader, the world-space position is looked up and, based on the unique vertex id, a VPL is chosen to be the target mapping position. According to the chosen VPL and the world-space position, the vertex is mapped onto the ISM. A geometry shader is used to create a splat that aligns it in tangent space to the surface. By using a sufficient splat size, the imperfect shadow maps are tightly filled with depth values from the real scene.

Finally, the filtered Kinect sensor data must be integrated into the reflective shadow maps of the spot light. For that, a $640 \times 480$ vertex buffer is used with an index buffer that connects the vertices together to render small triangles in a similar way as in the depth warping pass for the reconstruction of the real geometry (see Chapter 4). In the geometry shader, the vertex positions of a triangle are looked up using the position map in the same way as it is done for the ISM creation step. However, this time degenerated triangles must be deleted before they are rasterized. They may occur if the positions are invalid. Such cases are easily recognized since those vertices end up at coordinates $(0,0,0)$. So if any vertex in the triangle has a position at the origin, the triangle is discarded. Note that this kind of shadow map creation can lead to artifacts during visibility tests since the vertices have a limited validity for a different point of view than the position of the camera.

### 5.2.11 Limitations

Our method has a couple of limitations that either lower the quality of the final results or limit the range of applications for which the system can be applied.

The presented method inherits all limitations described in the geometry reconstruction chapter, when using the Microsoft Kinect sensor. Furthermore, if the real objects are reconstructed during run time, the color-mapping approaches cannot be applied. As noted above, using the reconstructed geometry for visibility tests for the spotlight may lead to artifacts. Furthermore, temporal coherence may lead to artifacts when the camera is moved. This is especially true for our newly developed multiple bounce approach. While in the first multiple bounce approach, proposed in Knecht et al. [67], the VPL placement introduces a bias, but is rather coherent, our newly developed multiple bounce method implies rather incoherent VPL placement. There-

fore, more flickering artifacts may occur if the camera is moved. Another limitation is that the presented method does not support reflective or refractive objects at all.

In Newcombe et al. [96], the depth data from the Kinect device is used to estimate the new pose of the camera. In comparison we are using BCH markers that are tracked using Studierstube Tracker [127]. It would be interesting to have a markerless tracking system so that disturbing markers can be avoided.

Furthermore, we still need a fish-eye camera in the center of the scene to capture the surrounding illumination. Like with the markers it would be interesting to remove the disturbing camera and use a method such as proposed by Gruber et al. [37], which does not need a camera or chrome sphere to estimate the surrounding illumination.

### 5.2.12 Results

All results were rendered at a resolution of 1024x768 pixels on an Intel Core2 Quad CPU Q9550 at 2.8GHz with 8GB of memory. As graphics card we used a NVIDIA Geforce GTX 580 with 1.5GB of dedicated video memory. The operating system is Microsoft Windows 7 64 bit, and the rendering framework is developed in C#. As graphics API we use DirectX 10 in conjunction with the SlimDX library. Our system uses the Microsoft Kinect sensor for see-through video and depth acquisition and an uEye camera from IDS with a fish-eye lens to acquire the environment map. Our tests took place in an office with some incident light through a window. Furthermore, we have a small pocket lamp to simulate some direct incident light. We use Studierstube Tracker [127] for tracking the Kinect and the position of the pocket lamp. Unless otherwise mentioned, we use 256 virtual point lights and represent the scene using 1024 points per VPL. The imperfect shadow map size for one VPL is 128x128, and we split the G-Buffer into 4x4 tiles.

We also want to mention that the Kinect sensor is too big and heavy for practical usage on an HMD. However, we think that this device could get much smaller in the future and therefore will become feasible in real life use cases.

Figure 5.12 shows a virtual Cornell box and a real white box illuminated by the captured environment. The Cornell box shadows the real box. The image is rendered at 18.0 fps with multiple bounces enabled as described by Knecht et al. [67].

Figure 5.13 shows the same scene with additional light from a real pocket lamp. It points towards the virtual Cornell box causing indirect illumination towards the real box. Note the red color bleeding on the box and the desk. The same illumination effect but reversed is shown in Figure 5.14. Here the pocket lamp illuminates the real box, again causing indirect illumination. Our system is capable of handling these different cases in a general way. Both images are rendered at 18.0 fps.

In our first approach [67], double shadowing and inconsistent color bleeding artifacts occurred as shown in Figure 5.15. The extensions to differential instant radiosity [68] overcome these issues, resulting in images as shown in Figure 5.16. Note that the dark shadow behind the real white box is removed. Furthermore, note the slight darkening of the area near the bright spot of the pocket lamp. Our framework cancels out indirect color bleeding caused by the real white box illuminated by the real pocket lamp. Compared to our first approach [67], the same amount of objects is rendered for shadow-map generation, they are just rendered into different render

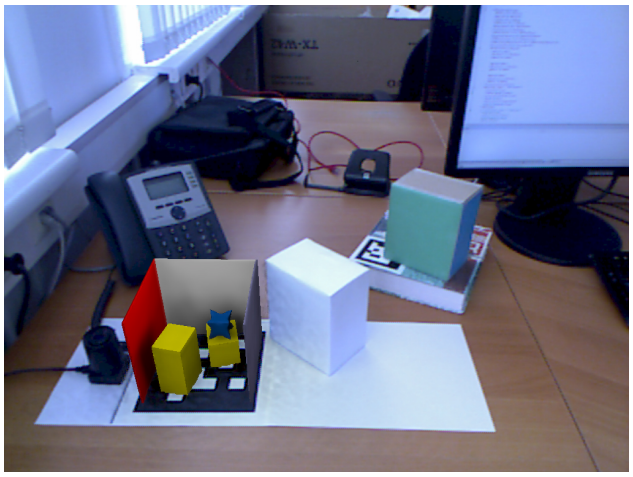


**Figure 5.12:** Virtual object shadows a real box. Image rendered at 18.0 fps.

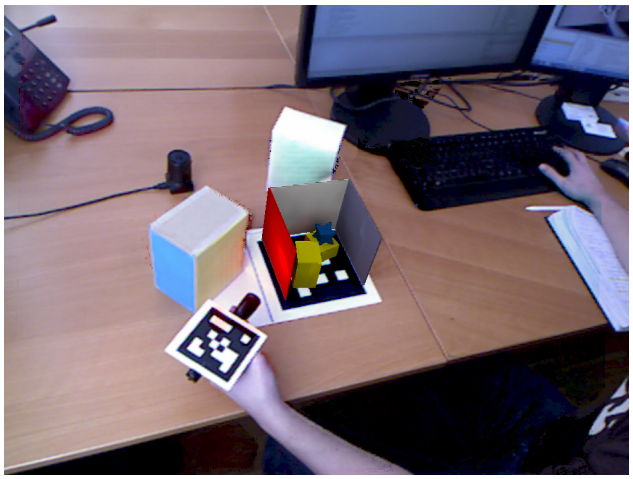


**Figure 5.13:** The real pocket lamp illuminates the virtual Cornell Box and causes red color bleeding on the desk and the real box.

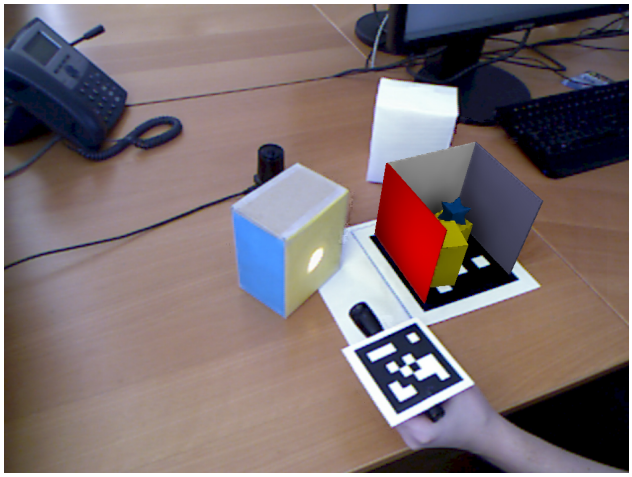


**Figure 5.14:** The real pocket lamp illuminates the real box causing indirect illumination onto the virtual Cornell Box.

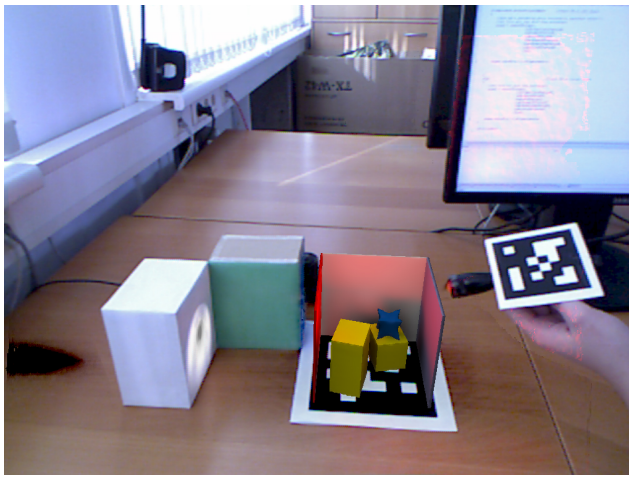


**Figure 5.15:** Double shadowing and inconsistent color bleeding artifacts. The dark spot on the left is due to insufficient visibility tests. Note the slight indirect illumination of the bright spot onto the desk and the green wall of the real box. Our first approach [67] could not avoid these artifacts.

targets. Hence, our new approach causes only a small overhead due to the additional shadow map lookup. In numbers, this means that the average frame rate for Figure 5.15 (with artifacts) is 18.4 fps and for Figure 5.16 (no artifacts) it is 18.2 fps.

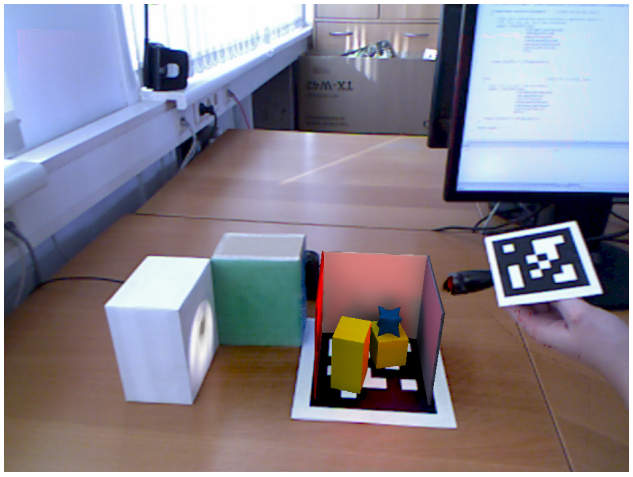
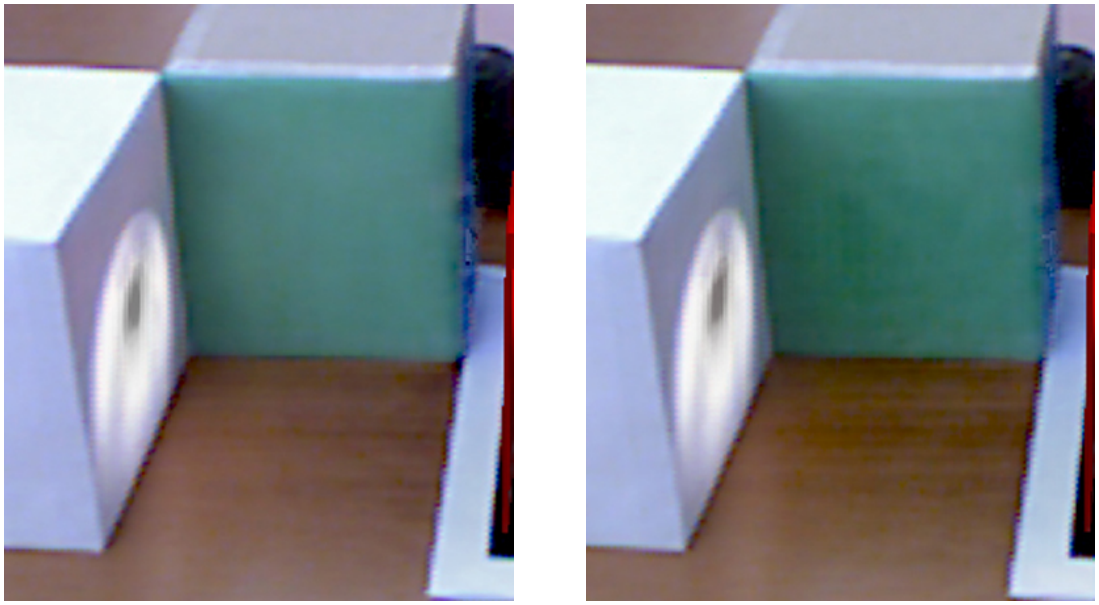


**Figure 5.16:** Our method avoids double shadowing and inconsistent color bleeding artifacts. Compared to Figure 5.15, there is no dark spot in the left part of the image. Furthermore the inconsistent color bleeding on the desk and the real green box is canceled out.



(a) (b)

**Figure 5.17:** (a): Close-up of Figure 5.15 shows incorrect color bleeding on the green box (due to illumination from a spot light which is occluded by a virtual object). (b): Close-up of Figure 5.16 illustrates that our new method correctly accounts for occlusion and cancels out the color bleeding.

### 5.2.13 Ground-Truth Comparison

We also compared our two multiple bounce methods against a ground-truth solution 5.18(a) created and provided by Kan and Kaufmann [54] using a modified path tracer for AR applications. The test scene only contains a plane that resembles the table, the Cornell box, a blue box, and the Standford bunny. Furthermore, the scene contains only diffuse surfaces.

Beside our two multiple bounce methods, we also implemented multiple bounces in a similar way to Ritschel et al. [117] as a reference VPL method. However, in contrast to their method, we do not perform importance sampling on an imperfect reflective shadow map, but rather create a new VPL for each VPL already existing. While their method leads to an unbiased result (except for the imperfect visibility tests), it needs certainly more fill rate to create the imperfect reflective shadow maps, which store similar data as a G-Buffer and therefore, is significantly slower.

Figures 5.18 and 5.19 illustrate all multiple bounce methods. The difference images were all created using the ground-truth image 5.18(a) as basis for comparison, and the resulting error was multiplied by a factor of five.

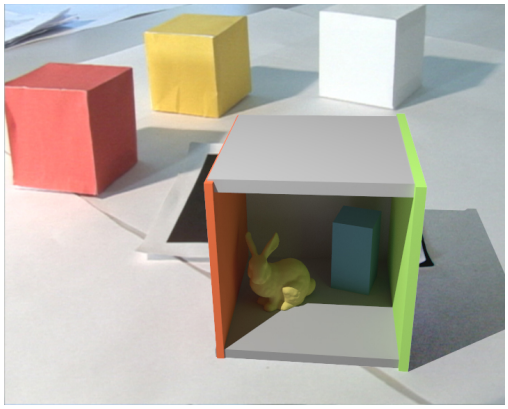
Figure 5.18(b) shows the multiple bounce method from Ritschel et al. [117] with one additional light bounce. Therefore, the error is quite large, as no further bounces are calculated. The image is rendered at a frame rate of 9.8 fps. In comparison our first multiple bounce method, as proposed in Knecht et al. [67], is shown in Figure 5.18(d). In the error image 5.18(e), one can see large errors at the top of the red wall near the ceiling. This is due to the fact that the VPLs placed on the table only *see* the ceiling, and thus a lot of VPLs will be placed there, causing the red wall to get too bright. Compared to the method from Ritschel et al. [117], our method runs at 28 fps and is thus approximately 2.8 times faster.

Figure 5.19(a) shows the method from Ritschel et al. [117] with three additional light bounces. The framerate drops down to 3.6 fps. Note however, how the overall error got smaller due to the additional light bounces (5.19(b) compared to 5.18(c)). Our new multiple bounce method is shown in Figure 5.19(c), which produces nearly the same quality compared to the three bounce method of Ritschel et al. [117], but still runs at 28 fps.

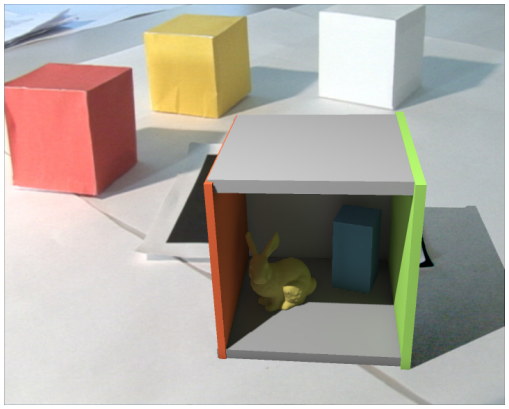
### 5.2.14 Conclusion

In this section, we introduced a method to render mixed-reality scenarios with global illumination at real-time frame rates. The main contribution is a combination of the instant radiosity algorithm with differential rendering. By adding information in various locations of the rendering pipeline, it is possible to distinguish between shading contributions from the real scene and from the combined real and virtual scene. Thus, our method is capable of relighting the real scene and illuminating the virtual objects in a general way by either using real or virtual light sources.

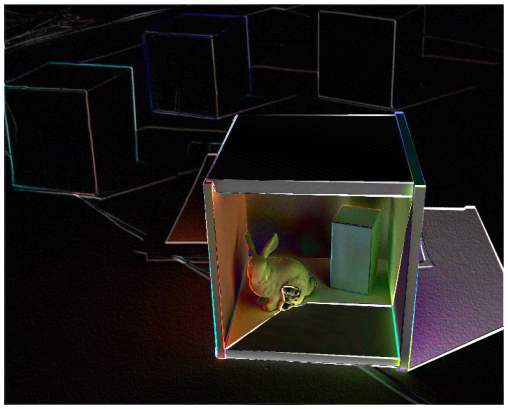
To enhance image quality, we have introduced a number of new techniques. First, we proposed a new way to align the point splats used for imperfect shadow map creation along the surface normal. This greatly reduces artifacts caused by wrong self-occlusions. Second, to remove temporal flickering artifacts between adjacent frames, we reuse the information from the previous frame and smooth indirect illumination computation over time. The results show that our method is able to simulate the mutual influence between real and virtual objects.



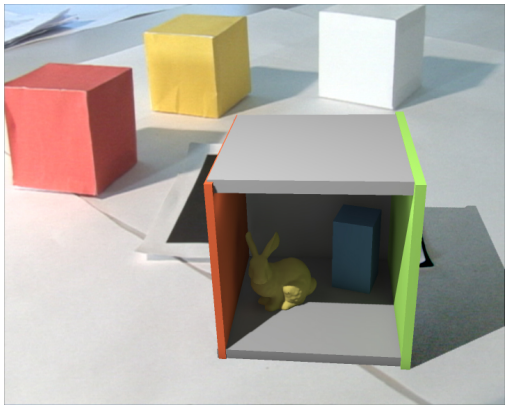
(a)



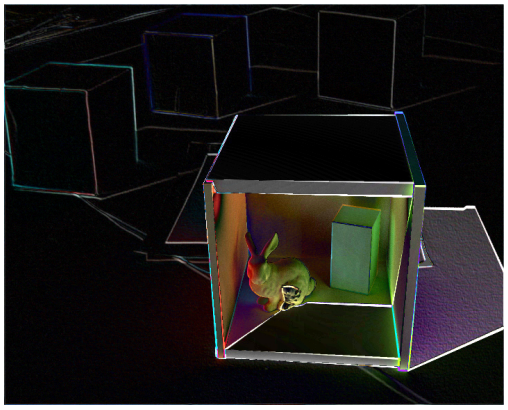
(b)



(c)

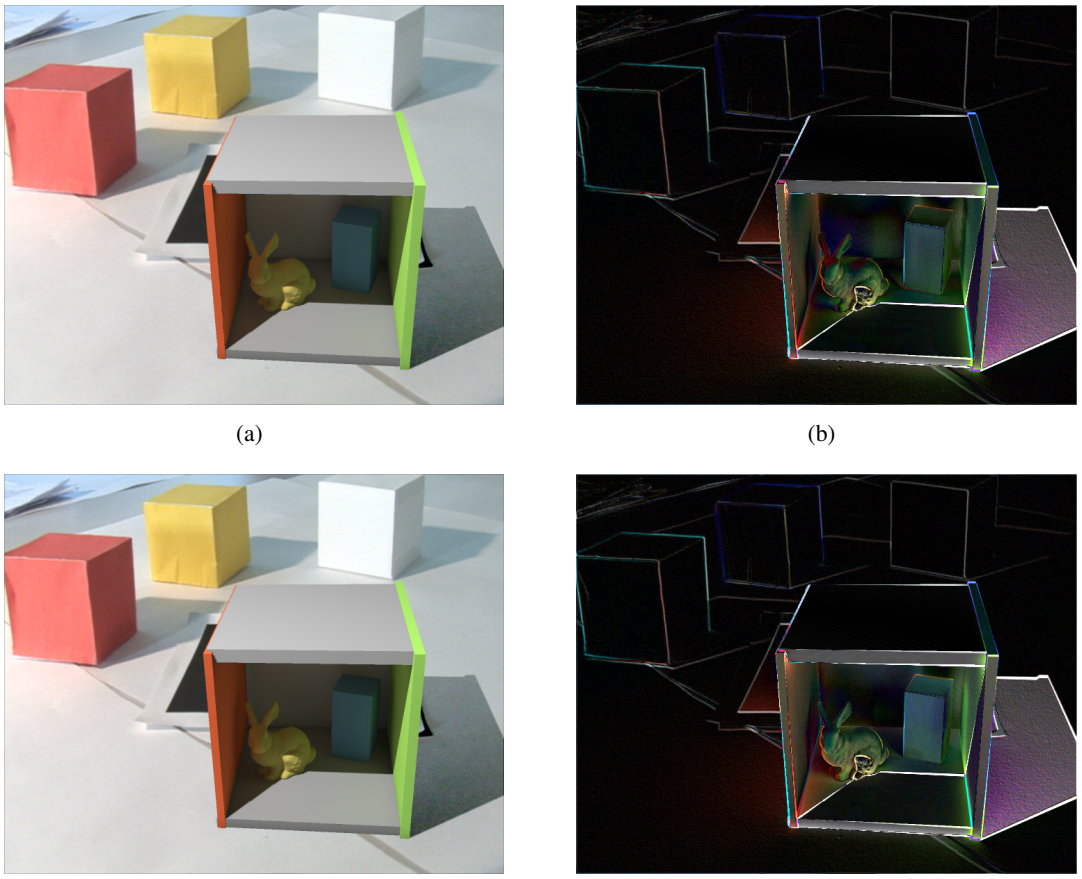


(d)



(e)

**Figure 5.18:** Image (a) shows a ground-truth image of a Cornell box (Image courtesy of Kan and Kaufmann [54]). Image (b) is rendered at 9.8fps using a similar approach as proposed by Ritschel et al. [117]. It is a little darker as only one additional light bounce is calculated. Image (c) shows the difference between image (b) and the ground truth (a) multiplied by a factor of five. Image (d) shows the biased multiple bounce calculation method of our approach [67], rendered at 28fps. The corresponding difference image is shown in (e).



(a) (b)

(c) (d)

**Figure 5.19:** Image (a) is rendered at 3.6fps using a similar method to Ritschel et al. [117], but this time three light bounces are calculated. Image (b) shows the difference between the groundtrouth 5.18(a) and (a) multiplied by a factor of five. In comparison, our new biased multiple bounce method shown in image (c) is rendered at 28fps and the corresponding difference image is shown in (d).

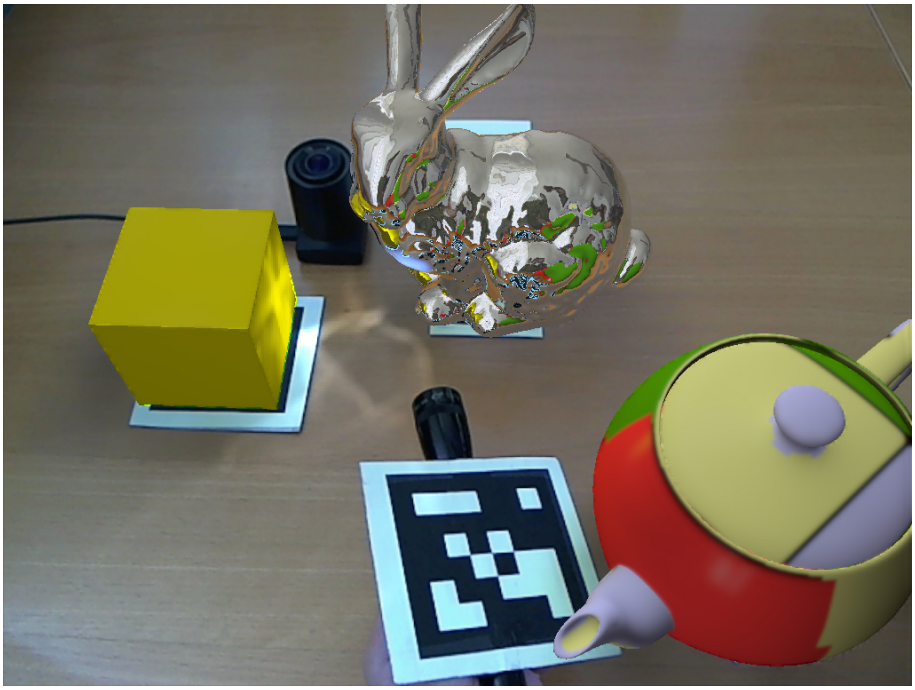


**Figure 5.20:** This image shows the Stanford bunny with reflective material illuminated by a virtual spotlight, causing reflections and caustics onto the real table and the virtual yellow cube (rendered at 19 fps).

## 5.3 Reflective and Refractive Objects for Mixed Reality

In this section, we present a rendering method that integrates reflective or refractive objects into our differential instant radiosity (DIR) framework usable for mixed-reality (MR) applications. Such objects are very special from the light interaction point of view as they reflect and refract incident rays. Therefore, they may cause high-frequency lighting effects known as caustics. Using instant radiosity (IR) methods to approximate these high-frequency lighting effects would require a large amount of virtual point lights (VPLs) and is therefore not desirable due to real-time constraints. Instead, our approach combines differential instant radiosity with three other, more specialized methods. One method handles more accurate reflections compared to simple cube maps by using impostors. Another method is able to calculate two refractions in real time, and the third method uses small quads to create caustic effects. Our proposed method replaces parts in light paths that belong to reflective or refractive objects using these three methods and thus tightly integrates into DIR. In contrast to previous methods that introduce reflective or refractive objects into MR scenarios, our method produces caustics that also emit additional indirect light. The method runs at real-time frame rates, and the results show that reflective and refractive objects with caustics improve the overall impression for MR scenarios.

Our main contributions are:

- Differential rendering effects are also applied on reflective and refractive objects. Note that the original differential rendering method [22] supported reflective and refractive objects, but the reflected or refracted objects where virtual representations of the real objects, and thus no differential rendering effects where applied.

- Caustics from spot lights emit indirect light.

The main advantage of our method is its high performance and its tight integration with DIR so that this global illumination solution for MR is extended with light coming from reflections and refractions.

### 5.3.1 Background

Our proposed method for reflective and refractive objects, including caustics, is a combination of four different methods, which we shortly describe in this section.

#### Differential Instant Radiosity

Differential instant radiosity was introduced in the previous section. Please note, that the method presented in this section is based on our first approach [67] and thus inherits limitations like double shadowing and inconsistent indirect illumination, which were already resolved in the previous section. Furthermore, for the proposed method we use a slightly altered differential rendering (DR) equation, as proposed by Debevec [22], to improve visual quality:

$$\Delta L = L_{rv} - L_r \tag{5.16}$$

$$L_{final} = L'_{cam} + \alpha \Delta L \tag{5.17}$$

where $L'_{cam}$ is the see-through camera image ($L_{cam}$) masked where virtual objects are placed and $\alpha = L'_{cam}/L_r$ is a scaling factor to compensate for the relative error between the camera image $L_{cam}$ and the $L_r$ buffer. In this way, the absolute values in the differential buffer $\Delta L$ are relative to $L'_{cam}$ instead to $L_r$, resulting in more visually pleasing images (see Debevec [22]). Note that Equations 5.16 and 5.17 only apply to diffuse objects, reflective and refractive objects are described in Section 5.3.2.

In DIR [67], the buffers $L_{rv}$ and $L_r$ are calculated using VPLs, and decision flags are used to determine whether the light contribution by a VPL should be added to $L_{rv}$ and $L_r$ or only to $L_{rv}$. Although instant radiosity causes indirect lighting, effects like caustics are difficult to simulate since the technique is mainly applicable for low-frequency lighting effects. That said, Dachsbacher and Stamminger [19] showed that simulating caustics is possible with instant radiosity. However, a lot of VPLs are needed just for a single caustic, which results in high computation costs. Therefore, in our method we avoid rendering caustics using virtual point lights and instead use more efficient methods.

#### Reflections, Refractions, and Caustics

Since specular objects should have high quality reflections, the second method used in this work was introduced by Popescu et al. [107]. They introduced a convenient way to create realistically looking reflections. A simple approach is to just render a cube map from the point of view of a reflective or refractive object similar to Pirk [106]. In this way, artifacts appear if a reflected object is close to the reflective or refractive one. The method from Popescu et al. [107] avoids these artifacts by using impostors. For each object that is reflective or refractive, a list of billboards is created, where each billboard belongs to a surrounding object that should appear in the reflection/refraction. When the reflecting or refracting object is rendered, the refracted and reflected rays are intersected with each billboard. Since the billboards do not just store the color but also the normal and the depth at each pixel, a ray hit position can be refined iteratively, leading to higher quality reflections.

The third method, which introduces realistic looking refractions, was proposed by Wyman [148]. It is able to compute two refractions for an incident ray using a two-pass approach. In the first pass, the back-facing triangles of the refractive object are rendered into a depth and normal buffer. In the second pass, the front faces are rendered. At each pixel, Snell's law is used to calculate the refracted ray direction. With the stored depth from the previous pass, a new position at which the refracted ray approximately exits the object can be calculated. Using this point and the associated normal, a new refracted outgoing ray is computed.

The fourth method was proposed by Wyman and Davis [149] and is used to compute caustics in real-time. For that, a scene gets rendered from the light source position. Besides the standard shadow map, three additional buffers are generated. One contains the hit positions from the refracted rays. The second buffer stores the photon intensity, and the third one stores the incident photon direction. In the second pass, the photons are rendered from the point of view of the camera or the spot light source using quads, where each quad approximates one photon. These quads are then blended into a caustic buffer. Using this buffer, realistically looking caustics can be rendered.

### 5.3.2 Extending Differential Instant Radiosity

The original DIR method only supported diffuse to medium glossy light bounces. In terms of Heckbert's [44] classification of light paths, and considering only one indirect light bounce, this means that the system was limited to *LDE* and *LDDE* paths, where *L* is the light source, *D* is a diffuse reflection at an object in the scene, and E is the eye. However, reflective and refractive objects, which generate caustics, were not supported. In the proposed extension to DIR, these reflections (*S*) and (double) refractions (*SS*) are added. Furthermore, for any primary spotlight source, the reflective or refractive objects produce caustics that emit indirect light.

First, we will introduce reflective and refractive objects to support *LDSE* and *LDSSE* light paths in DIR. Then we describe how caustic effects (*LSDE* and *LSSDE* light paths) are added, including those with an additional indirect light bounce (*LSDDE* and *LSSDDE*), using VPLs.

### Reflective and Refractive Objects in DIR

Imagine a virtual reflecting and refracting sphere that is surrounded by several real objects as illustrated in Figure 5.21. At the sphere's surface point $p$, the incoming reflected and refracted light from points $p_{refl}$ and $p_{refr}$ is composed according to Fresnel's law. In the original DIR approach, reflections and refractions at such a point are always rendered objects because the sphere is set to be virtual. In other words, only $L_{rv}$ stores the rendered representations of the real objects (points $p_{refl}$ and $p_{refr}$) and $L_r$ is set to zero. However, there are no differential rendering effects applied to real reflected or refracted objects, which is not desirable, leading to less accurate and therefore, less plausible appearance of real reflected and refracted objects.

Grosch [34] described the idea that points $p_{refl}$ and $p_{refr}$ could be back-projected into image space to lookup the color in the see-through video frame if they belonged to real objects. In this way, the reflected and refracted colors are taken from the see-through video frame itself instead of the rendered representations, leading to a higher degree of plausibility. Point $p_{refl}$ (ray two) lies inside the camera frustum (dashed yellow lines) and can be used for back-projection if the occluding green box is a virtual object, otherwise not. On the other hand, $p_{refr}$ (ray three) lies outside of the camera frustum and therefore, the rendered representation must be used. Please note that the back-projection is only valid if the real surfaces are assumed to be diffuse.

The previous example described only one out of several cases that need to be handled. Each of the light interacting elements (light source, reflected object, refracted object, reflective or refractive object) could be real or virtual (see Figure 5.21). In order to handle all possible cases, two steps must be altered in the DIR pipeline.

First, we will describe how the light accumulation step must be adapted for reflective or refractive objects. Then we describe how the final image gets composed when reflective or refractive objects are involved. Please note that the GI solution buffers are computed in screen space and on a per-pixel basis. Each pixel belongs either to a diffuse object or a reflective or refractive object. This means that Equations 5.16 and 5.17 are only applied if a diffuse object is visible at a pixel. On the other hand, equations marked with primes are only applied if a reflective or refractive object is visible at a pixel. For readability we therefore, introduce two additional GI solution buffers called $L'_r$ and $L'_{rv}$, although in practice $L_r$ and $L'_r$ are one and the same buffer and so are the buffers $L_{rv}$ and $L'_{rv}$.

**GI Solution Buffer Computation** As in the previous example, let us assume that $p_{refl}$ and $p_{refr}$ are points on real diffuse objects and that the reflective and refractive object is a virtual one. Then, in contrast to the original DIR approach, the reflected and refracted real objects have to be written into the $L'_r$ buffer even though point $p$ belongs to a virtual object (see Equation 5.19).

The outgoing radiance at point $p$ towards the view direction (ray one) is a composition, according to Fresnel's law, of the incident radiances from the reflective and refractive ray directions illustrated by the orange arrows in Figure 5.21. The functions $L_{refl}(ls)$ and $L_{refr}(ls)$ calculate the incident radiance from the reflected and refracted ray directions at point $p$ for a given light source $ls$ with the methods described in Section 5.3.1. Depending on Equation 5.20 the contribution of $p_{refl}$ or $p_{refr}$ due to light source $ls$ (all primary light sources and all VPLs) is added to the $L'_r$ buffer or not. Let $r(x)$ be a function that returns *true* if element $x$ is associated

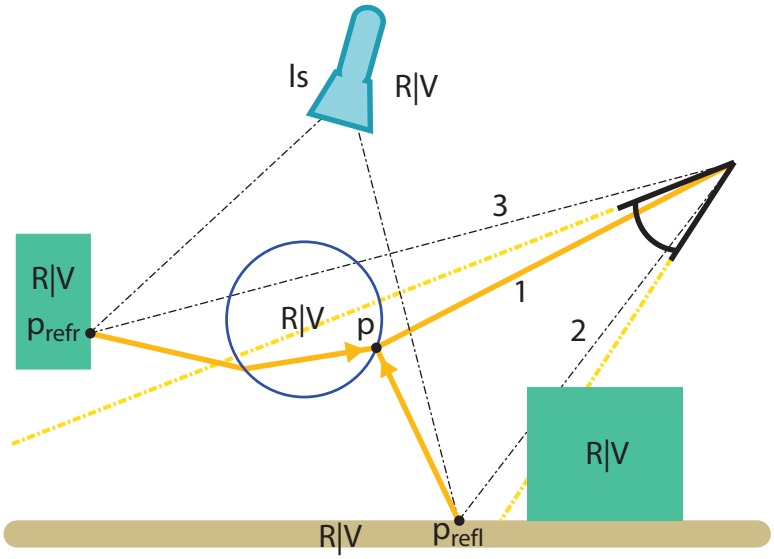



**Figure 5.21:** This figure illustrates all possible cases. Each light interacting object (light source, reflected object, refracted object, reflective or refractive object (blue sphere)) can be a real or a virtual object. The dashed yellow lines illustrate the view frustum of the video see-through camera. Rays one, two, and three show possible back-projection cases where rays one and two are inside and ray three is outside of the view frustum (see Section 5.3.2).

with a real object and *false* if not. $x$ can be a light source ($ls$), a light path ($\bar{x}$) or a surface point ($p$). Then the two GI solutions for reflective or refractive objects are computed as follows:

$$L'_{rv} = \sum_{ls} \left( L_{refl}(ls) + L_{refr}(ls) \right) \tag{5.18}$$

$$L'_{r} = \sum_{ls} \left( L_{refl}(ls) \cdot real(ls, p_{refl}) + L_{refr}(ls) \cdot real(ls, p_{refr}) \right) \tag{5.19}$$

$$real(ls, p) = \begin{cases} 1 & \text{if } r(ls) \wedge r(p) \\ 0 & \text{otherwise} \end{cases} \tag{5.20}$$

**Compositing Final Image** Since the reflected and refracted light contributions of real objects are also written into the $L'_r$ buffer, the contribution of real objects gets canceled out when the difference buffer $\Delta L'$ is calculated (see Equation 5.21). What remains in $\Delta L'$ are only the differential effects, caused by the virtual objects. The next step is to add information available in the video see-through image (see Equation 5.22). However, in order to get the correct color values from the video see-through image, back-projection as proposed by Grosch [34] must be applied.

$$\Delta L' = L'_{rv} - L'_r \tag{5.21}$$
$$L'_{final} = CI_{std}(p, p_{refl}, p_{refr}) + \alpha' \Delta L' \tag{5.22}$$

Equation 5.23 first checks whether all points are on real surfaces. If so, we can take the color at the pixel of point $p$ straight away, since it already shows the composed real reflections and refractions of $p_{refl}$ and $p_{refr}$. This is done by function $CI(x)$. It simply takes a 3D point $x$, projects it into screen space and returns the color value from the camera image ($L_{cam}$).

$L_{env}$ is a screen space buffer that contains data from the environment map (fish-eye lens) that has been reflected or refracted through the object, but only for those cases where no ray hit points for $p_{refl}$ or $p_{refr}$ can be found (e.g., when the ray leaves the modeled scene without intersection, see Figure 5.22). Equation 5.24 does the compositing and back-projection of the points $p_{refl}$ and $p_{refr}$. Therefore, in cases where not all points are on real surfaces, the returned color value of Equation 5.23 is the sum of the $L_{env}$ buffer and the $CI_{bp}$ function.

$$CI_{std}(p, p_{refl}, p_{refr}) = \begin{cases} CI(p) & \text{if } r(p) \wedge \\ & \quad r(p_{refl}) \wedge \\ & \quad r(p_{refr}) \\ L_{env} & \\ +CI_{bp}(p, p_{refl}, p_{refr}) & \text{otherwise} \end{cases} \tag{5.23}$$

In Equation 5.24, $\eta_1$ and $\eta_2$ are the Fresnel factors for the reflected and refracted radiances and $T_c$ is the transmittance color for refractive objects, which gets attenuated according to the distance the light travelled through the object. These translucent effects were implemented as described by Akenine-Möller et al. [2]. $CI_r(x)$ is a helper function that checks whether point $x$ is real or virtual and thus returns $CI(x)$ or zero. The data from the back-projection, however, is only used if the validation function $v(p, p_{refl}, p_{refr})$ returns *true* (see next Section) – otherwise the $L'_r$ buffer is used. Returning the $L'_r$ buffer cancels out the subtracted $L'_r$ part from Equation 5.21 and thus, only $L'_{rv}$ contributes to the final image.

$$CI_{bp}(p, p_{refl}, p_{refr}) = \begin{cases} \eta_1 CI_r(p_{refl}) & \text{if } v(p, p_{refl}, p_{refr}) \\ +\eta_2 T_c CI_r(p_{refr}) & \\ L'_r & \text{otherwise} \end{cases} \tag{5.24}$$

$$CI_r(p) = \begin{cases} CI(p) & \text{if } r(p) \\ 0 & \text{otherwise} \end{cases} \tag{5.25}$$

Similar to Equation 5.17, an $\alpha$ factor is computed to compensate for the relative error between the camera image $L_{cam}$ and the $L'_r$ buffers. However, for pixels belonging to reflective or refractive objects, the computation of $\alpha'$ must be altered as follows:

$$\alpha'(p, p_{refl}, p_{refr}) = \begin{cases} \frac{CI(p)}{L'_r} & \text{if } r(p) \wedge \\ & \quad r(p_{refl}) \wedge \\ & \quad r(p_{refr}) \\ \frac{CI_{bp}(p, p_{refl}, p_{refr})}{L'_r} & \text{otherwise} \end{cases} \tag{5.26}$$

Note that due to Equation 5.24, $\alpha'$ results in one if the points $p$, $p_{refl}$, and $p_{refr}$ are not valid.

**Dealing with Missing Information** So far we have assumed that there is always enough data for the reflected and refracted rays available, meaning that the validation function $v(p, p_{refl}, p_{refr})$ always returns *true*. However, in common scenarios this is usually not the case, as shown in Figure 5.22. There are two stages where data might not be valid.

The first one is at the reflection and refraction stage. For each reflective or refractive object, there are two so-called geometry buffers (G-Buffers) that store all information necessary to shade a point. These G-buffers contain the surface information of the hit points of reflected or refracted rays. However, it might happen that a reflected or refracted ray does not hit any of the pre-modeled real or virtual geometry as illustrated by the leftward going ray in Figure 5.23. In such cases, no illumination computation is possible since the corresponding G-buffers cannot be filled. Our strategy here is to use the information available in the environment map from the fish-eye lens camera and write the color information into a separate buffer $L_{env}$ at the corresponding screen space position that only contains color information from the environment map. The buffers $L'_{rv}$ and $L'_r$ are left untouched.

The second problem may arise at the compositing stage, in Equation 5.24, where the ray hit positions of reflected or refracted rays are back-projected to look up the video see-through image $L_{cam}$. First, if the lookup position is inside the video image (as illustrated by rays number one and two in Figure 5.21), we have to check if the color information is actually what we are looking for. It is possible that another real object occludes the hit point in the video feed, as happens with ray two in Figure 5.21. Therefore, the depths of the back-projected point and the depth stored in the primary G-Buffer are compared, and if they are within an epsilon range, the color information from the video image is assumed to be valid. However, if the looked up color information is not valid due to the depth test, the color information from the $L'_{rv}$ buffer at the original (not back-projected) point $p$ is used. Please note that the not back-projected pixel of the $L'_{rv}$ buffer contains the reflected or refracted objects visible on the surface of the reflective or refractive object.

In case the back-projected look up is outside of the video image, the data from the $L'_{rv}$ buffer is used. This case is illustrated by ray three in Figure 5.21. Furthermore, note that artifacts may appear at the border line between the back-projected camera image data and the $L'_{rv}$ buffer data if the colors differ too much (see zoom-in in Figure 5.22).

An exception is made for virtual refractive objects. In most cases the refractive component is far more important than the reflected one ($\eta_1 \ll \eta_2$) and only at the borders of an object, where the surface is seen from a very flat angle, the reflection component gets more important. However, according to the introduced equations the $L'_{rv}$ buffer would be used if the reflected ray does not hit any object ($v(p, p_{refl}, p_{refr}) = false$). Therefore, the validation function returns *true*

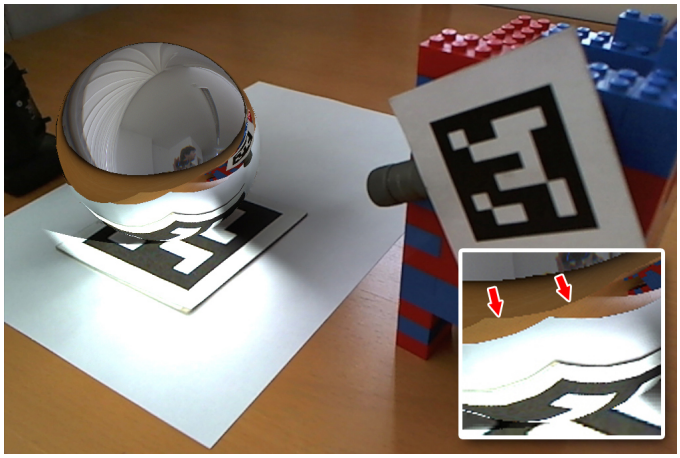


**Figure 5.22:** The image shows a virtual reflective sphere illuminated by a virtual spot light. The see-through video image is also visible in the lower part of the sphere. At pixels where no information is available from the video image, the information from the $L'_{rv}$ buffer is used. If there is also no information available in the $L'_{rv}$ buffer, then the image from the $L_{env}$ buffer, generated from the fish-eye lens camera, is used. The bottom right cut out shows a zoom-in on the sphere. Note the color change due to differences in the back-projected camera image and the $L'_{rv}$ buffer.

if point $p$ is virtual and $p_{refr}$ is valid. In this way, we can nicely refract the see-through video even though the reflected component is not valid. This leads to visually more pleasing results, however, note that this may introduce wrong coloring information – a limitation of our approach.

### Caustics in DIR

Reflective and refractive objects cause high-frequency caustic effects that cannot be approximated with instant radiosity techniques within a reasonable amount of computation time. Therefore, we integrated a faster method from Wyman and Davis [149] to compute the caustic illumination effects. In the next paragraph, we describe how caustics can be added to the GI solution buffers $L_{rv}$ and $L_r$ and then introduce reflected and refracted VPLs to simulate light paths with additional diffuse bounces, like *LSDDE* or *LSSDDE*.

**GI Solution Buffer Computation** Caustic effects are integrated for real or virtual spotlight sources and, similar to reflective or refractive objects, different cases have to be handled to decide whether a caustic quad should be added to the $L_r$ buffer or not:

$$L_{rv} = \sum_{ls} C_{quad}(ls) \tag{5.27}$$

$$L_r = \sum_{ls} C_{quad}(ls) \cdot real(ls, p_{hit}, p) \tag{5.28}$$

$$real(ls, p_{hit}, p) = \begin{cases} 1 & \text{if } r(ls) \wedge r(p_{hit}) \wedge r(p) \\ 0 & \text{otherwise} \end{cases} \tag{5.29}$$

where $C_{quad}$ is the caustic quad (see Section 5.3.3 for more details) on its hit position $p_{hit}$, emitted from surface $p$, and $ls$ are all spotlight sources. The $L_{rv}$ buffer is straightforward and does not need any special conditions. However, a caustic quad is only written into the $L_r$ buffer if all light interacting elements belong to real objects. Note that these quads are additively added to the $L_{rv}$ and $L_r$ buffers, which results in caustics, introducing *LSDE* and *LSSDE* light paths. Visible caustics in reflective or refractive objects are computed in a similar way, except that their contribution is added to the $L'_{rv}$ and $L'_r$ buffers and that the Fresnel factors and the transmittance color for refracted caustics is multiplied to them. To simulate *LSDDE* and *LSSDDE* paths, caustics need to emit additional light. At that point, VPLs can be used again.

**Creating Reflected and Refracted VPLs** In differential instant radiosity, so-called primary light sources are used to create VPLs. One of them is the spotlight source, which behaves like standard spotlights, except that it creates so-called reflective shadow maps (RSM) [18]. In previous approaches, these RSMs are used to create virtual point lights at surface points illuminated by the spot light. However, since reflective or refractive objects cause high-frequency illumination effects, placing VPLs directly on their surface is not a good idea.

Instead, we reflect or refract the position of a VPL that should originally have been placed on a reflective or refractive object as shown in Figure 5.23. In this way, the VPLs are placed where caustics appear, and additional indirect light gets emitted, simulating reflective *LSDDE* and refractive *LSSDDE* light paths.

Although the final color of a reflective and refractive object is composed of the incident refracted and reflected radiance, only one light path can be used to create the additional VPL. This is a limitation due to the way VPLs are created on the GPU, since all VPLs are generated in parallel. Therefore, a decision has to be made whether to follow the reflected or the refracted ray. Furthermore, the availability of hit information must be taken into account. Although the real scene geometry needs to be pre-modeled for DR, not every reflected or refracted ray will hit this geometry, since usually only nearby objects are pre-modeled. Therefore, if the reflected or refracted ray does not hit any surface, no valid data is available. Our method takes these cases into account by first checking if both rays have valid hit data available. If only one of them has valid data, then this data is used to create a VPL on the surface that the ray hits. In case of two valid hit points for the reflected and refracted ray, a quasi-random Halton value is used to choose the final VPL position. Both ray hits have the same probability, and no weighting is applied. Since each hit position is chosen every second frame on average, the VPL's intensity is doubled.

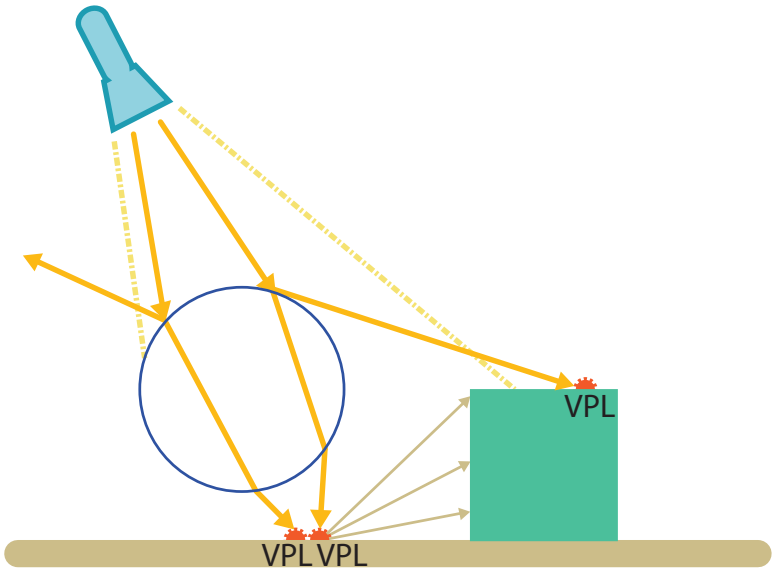



**Figure 5.23:** This figure illustrates the placement of the reflected and refracted VPLs. Depending on whether ray hits can be found, either the reflected or refracted ray hit is used to place a VPL. This VPL then approximates indirect illumination due to caustics.

### 5.3.3 Implementation

#### Rendering Reflective and Refractive Objects

The differential instant radiosity method is based on a deferred rendering system. This means that a G-buffer is created from the point of view of the camera. The G-buffer provides all information necessary for later illumination computation. Using a G-buffer has the advantage that the illumination is only calculated for pixels that are visible to the camera. Since multiple G-buffers are introduced in this paper, we will call this G-buffer the primary G-buffer $GBuf_P$.

For reflective and refractive objects, this G-buffer cannot be used straight away, as the surface information on these objects does not provide the information necessary to calculate its final color. The final color consists of the incident radiance from the reflected and the refracted ray directions. Therefore, we create two additional G-buffers ($GBuf_{refl}$ and $GBuf_{refr}$). Compared to the primary G-Buffer ($GBuf_P$), these G-Buffers contain the geometry information from the reflective and refractive ray hits of a reflective or refractive object. Note that they are also rendered from the point of view of the camera. They are created using the methods from Popescu et al. [107] and Wyman [148], with the difference that they store the geometric information from the ray hits.

In the next step, the additional G-Buffers are used to calculate the illumination from all primary light sources and all VPLs. The result is added to the $L'_{rv}$ and $L'_r$ buffers according to the equations mentioned in Section 5.3.2. Note that the resulting incident radiance is pre-multiplied according to Fresnel's law.

**Rendering Caustics**

To render caustics, two so-called photon G-Buffers ($PGBuf_{refl}$ and $PGBuf_{refr}$) are created for each spotlight. In contrast to the previous G-Buffers, these G-Buffers are rendered from the point of view of the spotlight. Similar to Shah et al. [130], we count the number of pixels to estimate the photon intensities. Note that the final photon intensity furthermore, depends on the spotlight characteristics, the distance to the light source itself, the attenuation and the transmittance color for refracted photons. Once $PGBuf_{refl}$ and $PGBuf_{refr}$ are created, they can be utilized to splat photon quads in the screen space into a so-called caustic buffer $CBuf$. The photon's quad size is calculated similar to Wyman [147]. The caustic buffer stores the photon intensity, number of photon hits, the depth, and average incident direction of the photons. With the caustic buffer $CBuf$ and the primary G-Buffer $GBuf_P$, the contribution to the $L_{rv}$ and $L_r$ buffers is computed. Since caustics should also be visible in reflections and refractions ($L'_{rv}$ and $L'_r$ buffers), the caustic buffer $CBuf$ and the G-buffers $GBuf_{refl}$ and $GBuf_{refr}$ are used to compute the reflected and refracted caustics.

### 5.3.4 Limitations

The proposed method inherits some limitations from the original differential instant radiosity method [67]. These are double shadowing artifacts, inconsistent color bleeding, and the need to pre-model the real environment. Although these limitations are addressed in Section 5.2, which is based on the solution presented by Knecht et al. [68], these improvements are not yet considered for reflective and refractive objects in DIR. While double shadowing artifacts should be straightforward to implement, inconsistent color bleeding introduces challenges for VPL placement in case of reflection or refraction. Currently, real caustics which are visible in the see-through video image can only be canceled out if there is no occluding virtual object between the caustic-emitting object and the surface of the caustic. In Figure 5.26(a) at the table in front of the yellow cube, the real caustic is still visible. Furthermore, the illumination computation time on the G-Buffers for reflections and refractions could be improved by applying the method proposed by Nichols and Wyman [98].

The system does not support multiple reflections and refractions between several reflective or refractive objects. This could be done with the method proposed by Umenhoffer et al. [140], where the complete scene is rendered into layered distance cube maps and ray marching as well as a secant search is then used to find appropriate hit points. By storing also the normals, multiple reflections and refractions can be simulated. However, note that multiple reflections and refractions add complexity to the computation equations due to multiple ray splits on reflective and refractive objects.

Another limitation is that VPLs do not cause any caustics. For now, only spot lights are able to generate them. In the future, the rendering quality of the caustics could be further improved by a method proposed by Wyman and Nichols [150]. They do not create caustic maps using a rasterization pass, which results in over and undersampling due to the high-frequency nature of caustics. Instead, photon samples are only created when further photons are needed in order to satisfy the quality metric. In this way, they can create higher quality caustic effects with the same amount of photons.

Furthermore, multiple bounces, as described in Section 5.2.6, are not supported because VPLs are not able to create *LSDE* and *LSSDE* light paths, which is necessary for additional light bounces that include reflective or refractive objects.

As the method is based on our first DIR solution [67], the reconstructed geometry from the Microsoft Kinect sensor cannot be used in conjunction with reflective or refractive objects. However, this is more related to the question on how the various G-Buffers could be created from the position and normal maps provided by the real-time reconstruction stage. Note that even though this would be solved, the Kinect sensor is not able to reconstruct the geometry of real reflective or refractive objects.

### 5.3.5 Results

For the results shown in this section, we used a PC with an Intel Core2 Quad CPU at 2.8Ghz, 8GB of memory, and an NVIDIA Geforce GTX 580 with 1.5GB of video memory. All results were rendered at a resolution of 1024x768 pixels. A uEye camera from IDS with a fish-eye lens was used to acquire the environment map, and a Logitech HD Pro 910 webcam was used for the video see-through input.

For all test scenarios, we used a total of 256 VPLs with an imperfect shadow map size of 128x128 pixels, using a total of 1024 points per VPL to represent the scene. Both the caustic maps for the pocket light source and the impostor texture maps had a resolution of 512x512.

In Figures 5.24 and 5.25, a spot light illuminates a virtual Stanford bunny that has green-colored glass. The light travels through the Stanford bunny and introduces green caustics on the box behind it. Figure 5.24 shows caustics which do not introduce additional indirect illumination. In contrast, Figure 5.25 shows one added indirect light bounce by placing VPLs on surface points showing caustics. These green caustics then introduce indirect color bleeding on the white paper and on the desk. This image was rendered at 13 frames per second (polygon count: 69680).

Figures 5.26(a) – (d) show a comparison between real and virtual caustic illumination effects. A small pocket light illuminates a metallic pocket bottle from the right side, causing a caustic on the real table and on the virtual yellow box. In the top row 5.26(a) and 5.26(b) the pocket bottle is a real object, whereas in the bottom row 5.26(c) and 5.26(d) a virtual pocket bottle is placed into the scene. In the left column 5.26(a) and 5.26(c) a real pocket lamp is used, and in the right column 5.26(b) and 5.26(d) a virtual one (real pocket lamp is switched off). Please note how the yellow box is reflected in all images on the pocket bottle. All images were rendered at 20 frames per second (polygon count: 3028).

### 5.3.6 Conclusion

In this section, we presented an extension to the differential instant radiosity method that tightly integrates reflective and refractive objects. The original differential rendering method was not able to apply the differential effects to reflected or refracted objects. The proposed method adds differential effects to those objects and back-projects color information from the see-through video image if possible. In this way, we are able to simulate reflective and refractive objects in mixed reality scenarios.

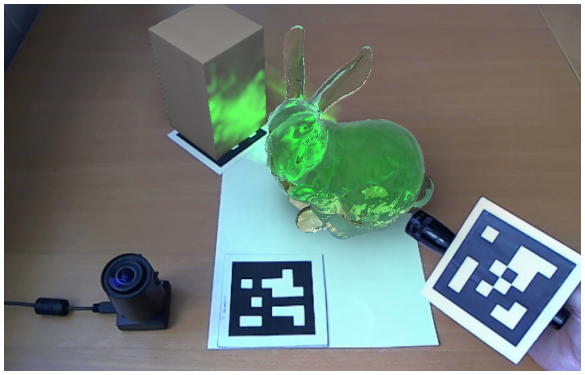


**Figure 5.24:** In this figure, a spotlight creates caustics on a box without any indirect illumination effects.

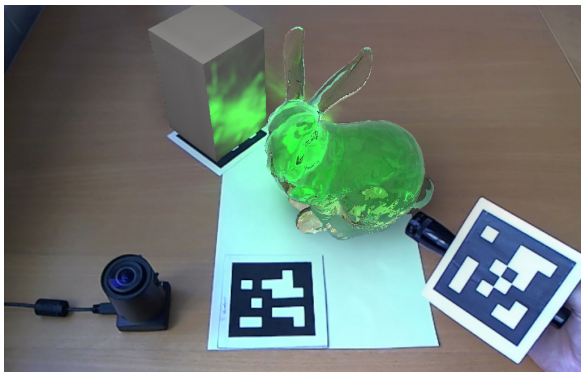
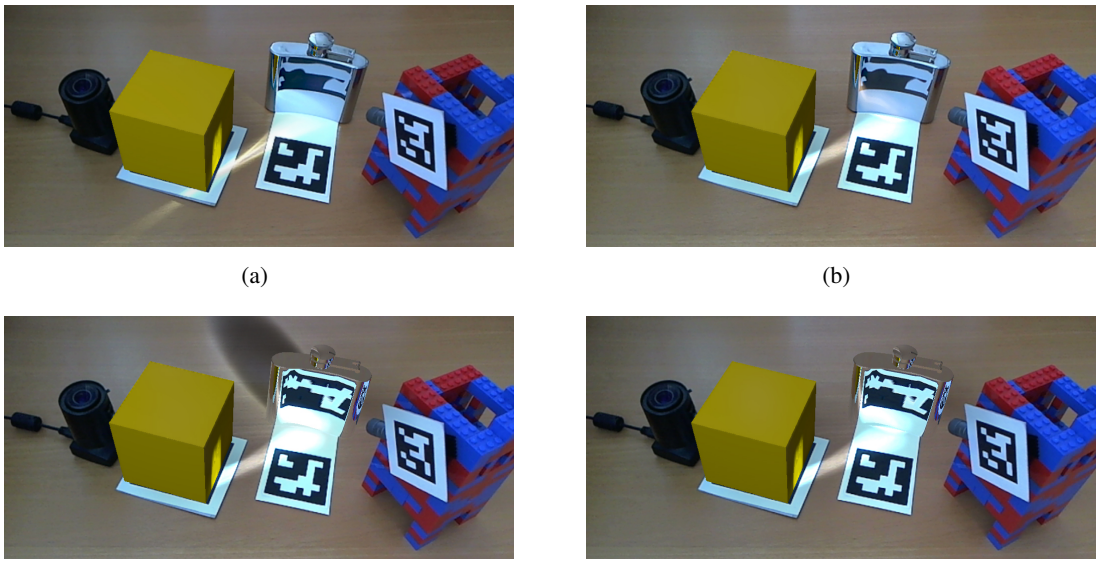


**Figure 5.25:** In this figure, a spotlight creates caustics on the box. VPLs placed on the caustics cause additional indirect illumination on the white paper.

Furthermore, spotlights are able to generate caustics from those reflective or refractive objects. By using a fast splatting method, the resulting caustics have a higher quality than caustics produced with virtual point lights. In addition, these caustics emit light at their hit points, causing one additional indirect light bounce. With the proposed method, we are able to achieve 15 to 25 frames per second.



(a) (b)

(c) (d)

**Figure 5.26:** (a) Real spotlight illuminates real pocket bottle. (b) Virtual spotlight illuminates real pocket bottle. (c) Real spotlight illuminates virtual pocket bottle. (d) Virtual spot light illuminates virtual pocket bottle.

CHAPTER 6

# Evaluation

## 6.1 Introduction

The previous chapters presented methods to improve the appearance of virtual objects in real environments and in the ideal case make them indistinguishable from real objects. In this chapter, we sketch how an ideal research framework for photorealistic mixed reality should look like. We then describe the current framework that was developed with the goals of the ideal one in mind. Based on this framework, a preliminary user study was performed to test whether task performance is influenced by rendering quality. A second user study will be presented in Section 6.4. We developed a web survey in a side-by-side comparison style, where participants had to judge whether the virtual objects fit better into the real scene compared to the other image or not. The results show that the methods described in Chapters 4 and 5 do have a significant influence on the perceived quality of the virtual objects.

## 6.2 A Research Framework for Visually Plausible AR

### 6.2.1 The ideal framework

In Chapter 3, we described the three varieties of realism and proposed to divide image features in mixed reality into two categories called *visual cues* and *augmentation style*. To investigate the influence of the visual cues and the augmentation style, an ideal research framework for photorealistic mixed reality should produce photorealistic results including camera artifacts, so that virtual objects are indistinguishable from real objects. Furthermore, since it should be used for experiments, it should be very flexible to configure.

The framework should allow easily plugging different modules into the rendering pipeline and it should be efficient to set up experiments. The API should be designed in a way that new hardware devices can be easily incorporated into the existing framework. Furthermore, utility functions for data logging, tracking and calibration should be provided.

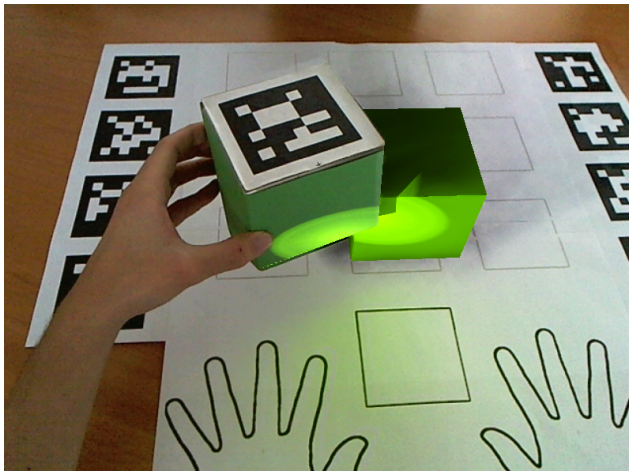


**Figure 6.1:** This figure shows the augmented scene of our experiment including shadows and color bleeding.

Such a framework could be used to study how the human visual system (HVS) processes images and how different visual cues alter perception. Especially in medical AR training simulators, it is important that the spatial perception correlates with the real world. Otherwise the simulation would diverge too much from real world situations to be a suitable training method.

### 6.2.2 The current framework

Guided by the goals of the ideal framework, we propose a software research framework offering new possibilities to investigate perceptual issues. With the proposed framework, we are able to study perceptual issues with shadows, dynamic environmental illumination and indirect illumination, as shown in Figure 6.1 – all at real-time frame rates. Kruijff et al. [72] introduced a taxonomy of the main perceptual issues in augmented reality. They classified these based on the so-called perceptual pipeline, which consists of five stages: *Environment, Capturing, Augmentation, Display Device*, and, finally, the *User*. The presented framework fits into the *capturing* and *augmentation* stages of the perceptual pipeline.

In order to achieve interactive to real-time frame rates, we have to relax the term photorealistic MR. Instead we will use the term *visually plausible* because the rendered images should look visually plausible to the human visual system while they will not be absolutely photorealistic. For example, we will use approximative visibility functions instead of an absolutely correct one to gain rendering performance.

The framework is based on the DIR method presented by Knecht et al. [67] (see Chapter 5). It is developed in C# and runs on Windows 7 64 Bit. The graphical output is done via the SlimDX and DirectX 10 APIs. It is therefore very easy and fast to develop new experiments, as C# offers many tools and functions.

To allow for a very flexible framework, the rendering pipeline can be defined in a XML configuration file that can be loaded via the GUI. In this way, it is easily possible to exchange a

tracking system or change a camera without the need to alter the whole experiment.

As a lot of studies are about rendering visual features, shader development should be very efficient. In our framework they can be manipulated in an external editor at runtime. As soon as the shader is saved, it will be reloaded automatically. In this way, instant visual feedback is provided.

The current renderer supports two types of shadows. For spotlight sources we use standard shadow mapping, and for indirect illumination we use by default imperfect shadow maps (ISM) [117] for every virtual point light as described in Chapter 5. However, standard shadow mapping can also be used for the virtual point lights. Furthermore, shadowing and indirect illumination can be switched on and off separately at runtime. In this way, the influence of local illumination versus global illumination in an MR setup can be investigated in interactive experiments.

The fish-eye camera currently in use is only able to capture low dynamic range images. However, the rendering framework uses the method from Landis [74] to extrapolate a high dynamic range image from it. This is a very rough approximation and the best solution would be to have a HDR camera.

Dynamic spotlights are also supported. They can either be real pocket lamps that are tracked or virtual. They illuminate the real and virtual objects accordingly. The framework can handle multiple camera streams on the fly and the captured frames are available as textures in the video memory or directly in main memory. In this way, they can easily be changed if necessary in a post-capture step.

The tracking interface currently supports three different types of trackers. The first one is the Studierstube Tracking framework [127]. The second one is based on the PTAM tracking method from Klein and Murray [64] and the third one supports the VRPN protocol [137].

### 6.2.3 Technical Issues

As this evaluation framework is work in progress, there are still several limitations and technical issues that are unsolved. One of the main issues for further perceptual studies is that the framework in the current stage does not support stereo rendering.

Furthermore, calibration is crucial when it comes to accurate rendering. As Kruijff [72] mentions, there are several points in the perceptual pipeline where errors decrease the quality of the final results, and this is also true for this framework. If the tracking is not accurate, wrong edges are far more obvious due to artificial indirect illumination overlays. Klein and Drummond [62] already proposed a method where rendered edges are accurately snapped to edges in the video stream and thus reduce visible tracking errors.

The fish-eye lens camera does not deliver any distance information of the environment. So it is not possible to take near light sources accurately into account except when they are tracked.

The method used to compose the final images limits the framework to video see-through head mounted displays (HMD). Furthermore, the real-time global illumination computation needs a powerful graphics card and thus mobile augmented reality is not supported yet.

**Figure 6.2:** Participant performing the experiment.

## 6.3 Preliminary User Study

To test our framework, we have conducted a preliminary user study on the influence of shadows and indirect illumination for five different tasks.

### 6.3.1 Experiment Setup

The experiment was conducted at the HIT Lab NZ. The study setup as shown in Figure 6.2 consisted of a table plate with several fiducial markers, two standard USB webcams, a HMD, and two targets (small green cubes with fiducial markers). To track hand movement we attached three different optical markers on the participant's hand: One at the index finger, one at the thumb, and one at the wrist (see Figure 6.3).

One webcam was attached to the HMD to capture the participant's view. The other one was placed above the table. Using this setup, we could achieve correct tracking even in situations when the cube marker was not visible to the head mounted camera.

### 6.3.2 Task description

The first task showed a virtual cube at a random position, while the real cube was fixed in the middle of the table. The participants had to estimate the distance between the real and the virtual cube in centimeters (see Figure 6.4).

In the second task, the virtual cube was randomly placed in front of the participants. They had to grab the real cube, located on a fixed starting point, and move it to the virtual cube's position (see Figure 6.5).

The third task was similar to task two but this time, the virtual and the real cube were swapped. The real cube was placed at random positions on the table by the experimenter and the virtual cube had to be moved to the same position using the cursor keys on a computer keyboard (see Figure 6.6).

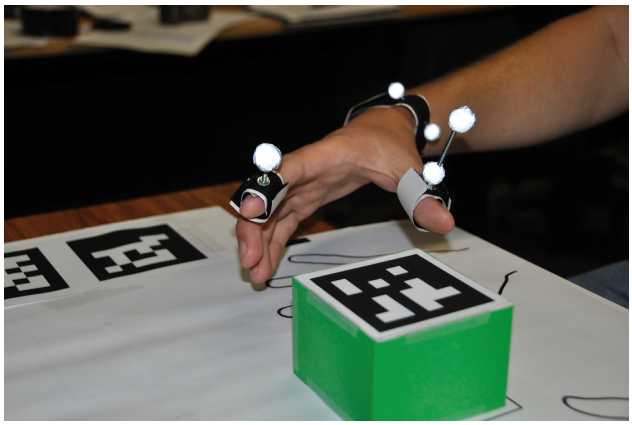


**Figure 6.3:** The green box and the markers for tracking the hand movement.

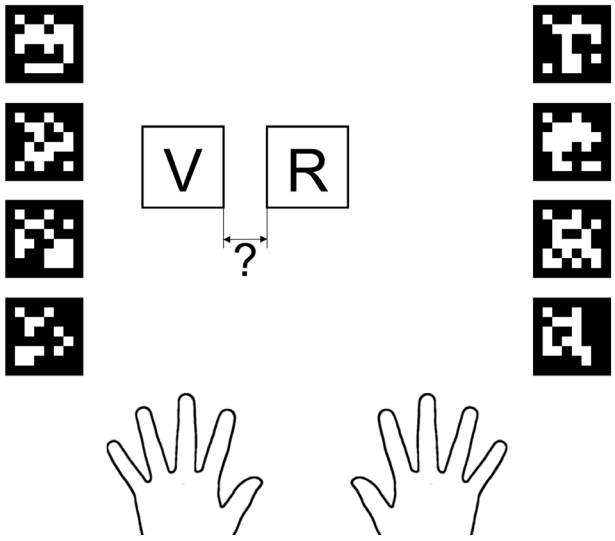



**Figure 6.4:** In task one, the distance between a virtual and a real cube should be estimated in centimeters.

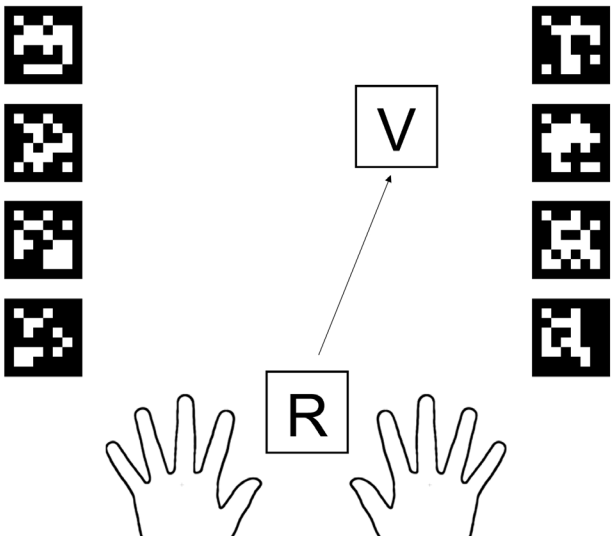



**Figure 6.5:** In task two the real cube must be placed at the position of the virtual cube.

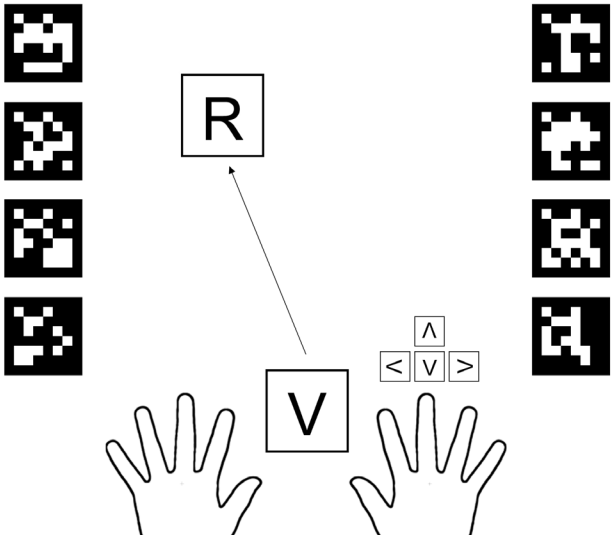



**Figure 6.6:** In task three the virtual cube must be placed at the position of the real cube using the keyboard.

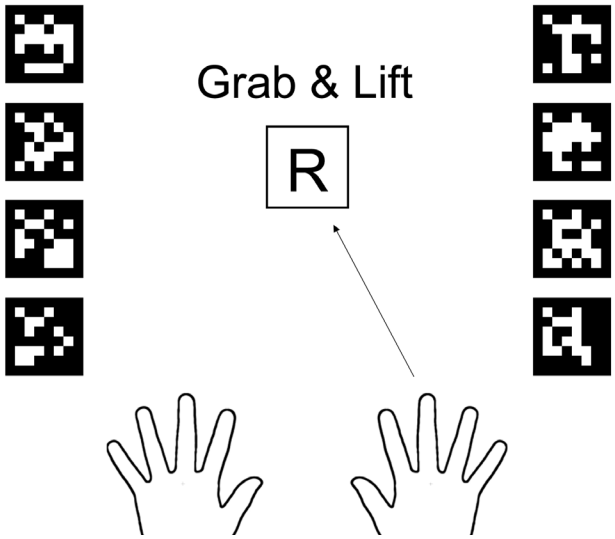



**Figure 6.7:** In task four the participant has to grab and lift the real cube as fast as possible.

In task four, the real cube (without any virtual augmentation) was placed at a random position on the table and the participant had to grab and lift it up as fast as possible. Before the task started and the scene was seen through the HMD, the participants were asked to place their hands at a fixed starting position (see Figure 6.7).

Task five was similar to task four except that the cube was overlaid with a virtual cube. In this way, the visual input was virtual, but the tactile input when grabbing and lifting was real (see Figure 6.8). The participants were instructed to perform tasks two to five as fast and as accurate as possible.

**Rendering modes:** For all tasks, we had three rendering modes (see Figure 6.9). The first rendered the scene without any cast shadows or indirect illumination. The second included shadowing between real and virtual objects but no indirect illumination. The third rendering mode included inter-object shadowing and indirect illumination, causing color bleeding. The study followed a within subject design and the conditions were administered according to a latin square to minimize the risk of carry-over effects. After the participants had finished all five tasks, they were interviewed.

### 6.3.3 Results and Discussion

Twenty-one people participated in the study, fifteen male and six female participants between the age of 19 to 59. All participants but one, who had to be excluded because of color blindness, had normal or corrected to normal eyesight.

It took between 30 to 60 minutes for each participant to finish all five tasks and the interview. Because not all data met the requirements for a repeated measures ANOVA (normality, sphericity), we analyzed the data using non-parametric Friedman tests.

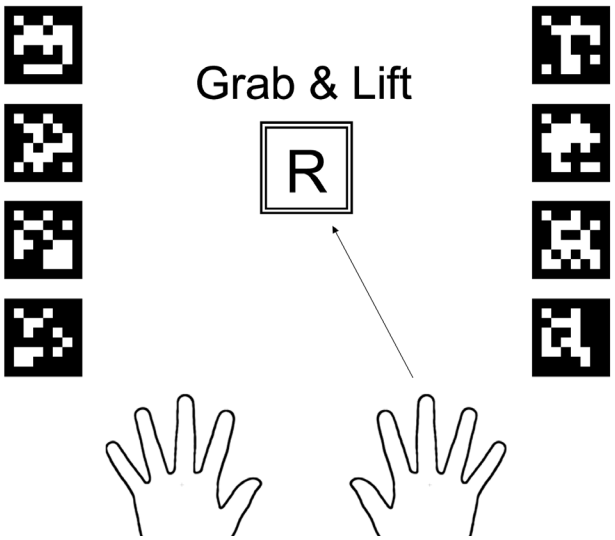



**Figure 6.8:** In task five the participant sees the virtual cube overlaid onto the real cube. Again the participant has to grab and lift the virtual cube as fast as possible.

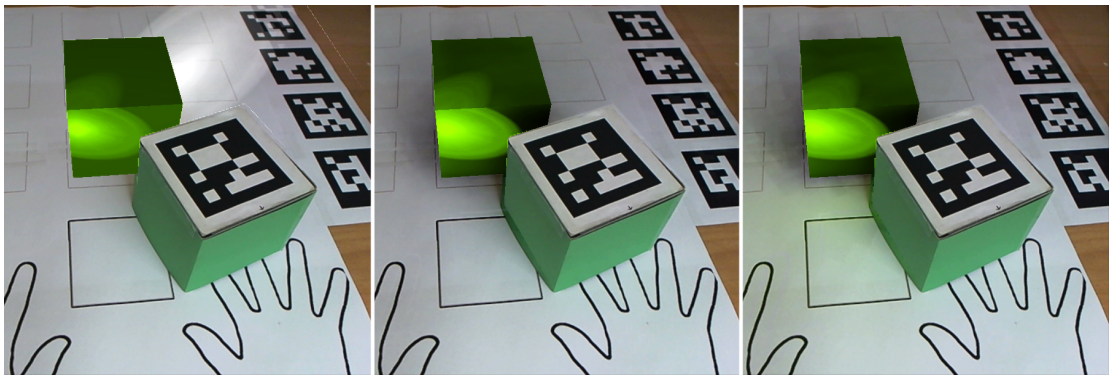


**Figure 6.9:** The three different rendering modes (left to right): no shadows/no indirect illumination, shadows/no indirect illumination, and shadows/indirect illumination.

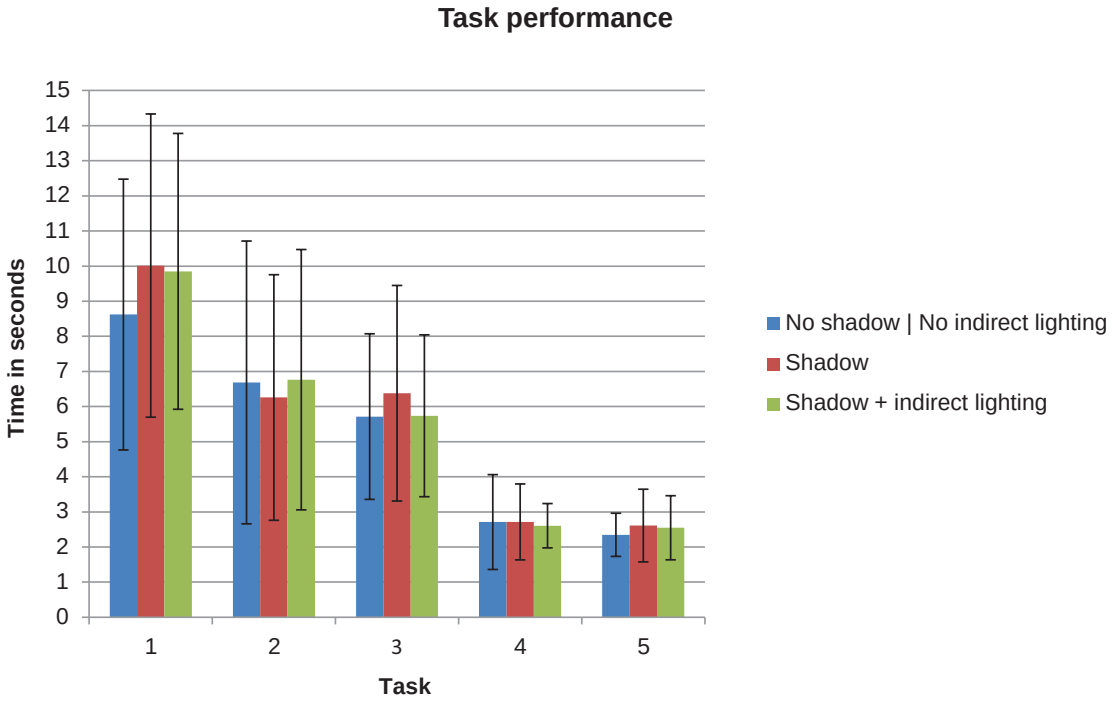



**Figure 6.10:** This figure shows the task performance in seconds for each task and rendering modes.

Figure 6.10 shows the task performance in seconds for each task, and Figure 6.11 shows the errors for the first three tasks where distances had to be estimated. Our analysis did not show any evidence that the different rendering modes had an effect on task performance. This goes in line with the experiments performed by Thompson et al. [139]. However, we have to be cautious in comparing these two experiments because in our user study, the participants had to judge distances less than one meter, whereas Thompson's experiment was based on locomotion and the distances ranged from 5 to 15 meters. Furthermore, they used an immersive virtual-reality system whereas we used an MR environment. Another study by Lee et al. [75] also investigated the influence on visual appearance and search task performance in an outdoor augmented-reality environment as well as three virtual-reality environments with different levels of realism. Out of 16 questions, only 4 showed a significant effect for different levels of realism. However, the authors also mentioned that this could be due to the difficult lighting conditions or the differences in the real environment to the modeled virtual environment.

When we designed the tasks, we first planned to disable occlusion, so that it could not be used as a depth cue. With no occlusion, the virtual cube would always be rendered on top of any real-world object, even in situations in which it should be occluded by a real cube. However, for a more realistic study setup, we decided to allow occlusion. As expected, our study shows that most of the participants used the occlusion cue to place the cubes at the right spot, regardless whether the virtual or the real cubes where manipulated (task 2 & 3). Seven participants

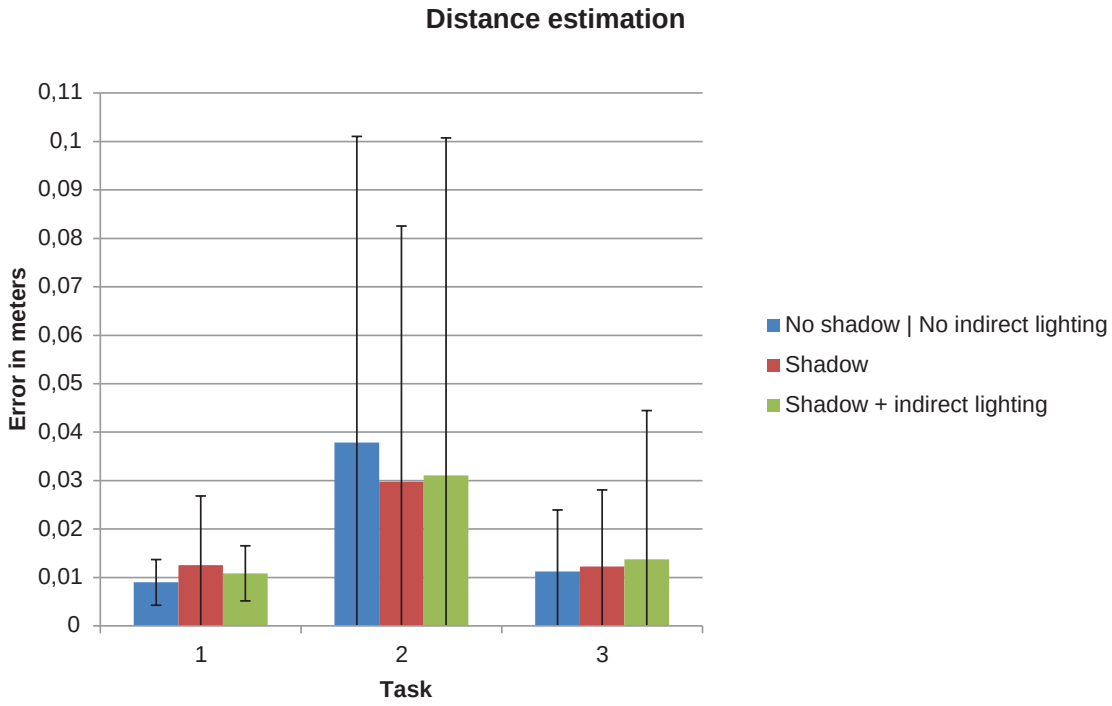



**Figure 6.11:** This figure shows the error of the distance estimations for tasks one to three.

recognized the shadows, but only one recognized indirect illumination.

In task one, the virtual cube was randomly positioned along the main axes, and six participants mentioned that it was much easier to estimate the distance on the x and y axis rather than in depth direction. Although we could not find a significant effect to corroborate this, the distance estimation error was slightly less for the x and y axis. Furthermore, the time used for distance estimation is slightly smaller when no shadows and no indirect illumination are shown. This could indicate that the cognitive load is larger with shadows and indirect illumination due to more visual cues. However, both effects are not significant and rather small.

In task two, the real cube was moved to match the position of the virtual cube. Interestingly, seven participants found task three, manipulating the virtual cube to match the real cube using a computer keyboard, more intuitive and easier. Moreover, Figure 6.11 shows that the average error is larger than in task 3. The difference between the two tasks was that the target cube position in task 2 varied along three axes (x, y and z) whereas in task 3 it varied only in two axes (x and z) but not in height (y axis). This could explain the larger error, since no reference plane, like the table, was given in task 2. Furthermore, in task 3 the participants did not have to change the cube's orientation since it was already aligned correctly.

In tasks 4 and 5, some participants complained that the cube was too large to grab and that the marker for hand tracking disturbed the grabbing process.

We observed that the participants completed the tasks in very different ways. Some of the participants focused on speed, others more on accuracy. Some participants excessively moved

their head to get different viewing angles, while others nearly did not move at all. These different strategies probably influenced the final results and therefore, should be controlled in future experiments.

The take-home message of this study is that we could not measure any significant effects on task performance for different rendering modes. Participants made heavy use of the occlusion cue, and it would be interesting how the results change if occlusion gets disabled. Furthermore, it is important to mention that the results only apply for this specific setup, and it does not imply that other user studies get the same results for these rendering modes but with different scenario setups.

## 6.4 Web Survey

In this section, we present a web survey that was conducted to find out the relative quality of the developed methods from Chapters 4 and 5. In contrast to the previous study, we use a web survey presenting images in a side-by-side comparison instead of a complete mixed-reality setup. This has the advantage that tracking errors cannot impair judgment, and that more people can participate in the study under varying conditions (monitor settings and surrounding illumination).

### 6.4.1 Experiment Setup

The web survey is set up as a forced-choice experiment between two images per trial, showing mixed-reality scenes, positioned side-by-side. At first, an entry screen is shown, where the participants enter their gender and age. All the 65 following screens/trials are side-by-side comparisons presented to the participants, as shown in Figure 6.12. For each trial, we stated the same request to the participants: „Please click on the image where you think that the virtual objects fit better into the real world.“. Note that the participants do not know which objects are real or virtual.

The mixed-reality scenario is placed on a wooden table with a white wall in the background. In order to avoid direct comparison between the side-by-side images, different scene configurations are used. While the objects stay the same (boxes, a banana, a red or blue table lamp, a book and a Cornell box), they are partly rearranged for different scene configurations. Furthermore, the camera position is also varied for different scene configurations. The maximum angle from the scene center towards the different camera poses is approximately 30 degrees. The average distance of the camera towards the scene center is approximately 51 centimeters. In total, six different scene configurations were created.

In this study setup, three different rendering modes could be switched on or off to generate different qualities of rendered images. The first mode, called *global illumination* (GI), activates or deactivates shadows and indirect illumination. In contrast to the previous user study, we do not separate the shadow and indirect illumination modes, in order to reduce the trial count. The second mode, called *color mapping* (CM), enables or disables the adaptive statistics-based color mapping method, as described in Chapter 4. If this mode is deactivated, we use a simplified version of the tone-mapping operator from Reinhard et al. [113]. The third mode, called *cam-*

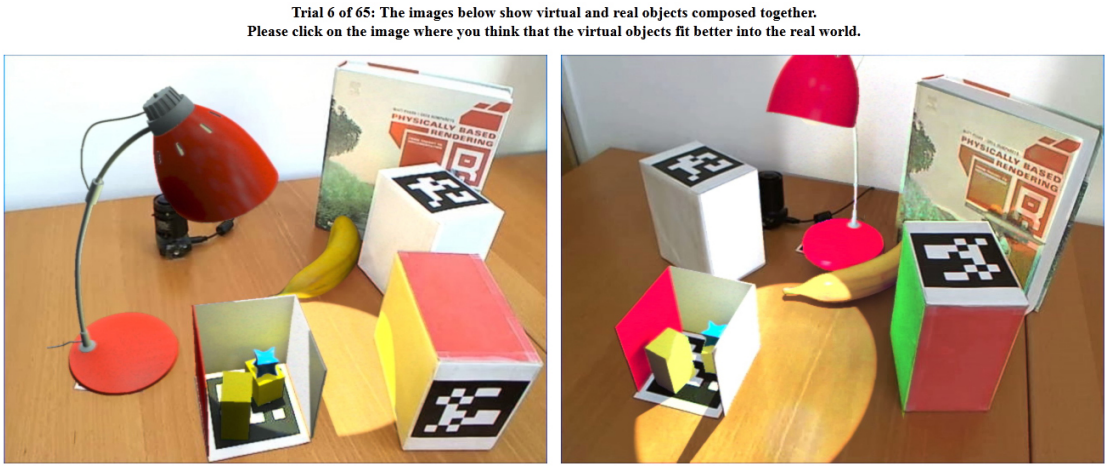



**Figure 6.12:** This image shows a trial of the web survey.

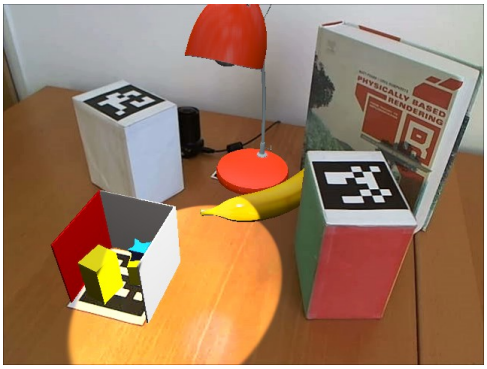
*era artifacts* (CA), adds camera artifacts, as proposed by Klein and Murray [65], to the virtual objects.

These rendering modes can be combined, to so called *rendering mode compositions* (RMCs), in an arbitrary way. We therefore end up having eight rendering mode compositions:

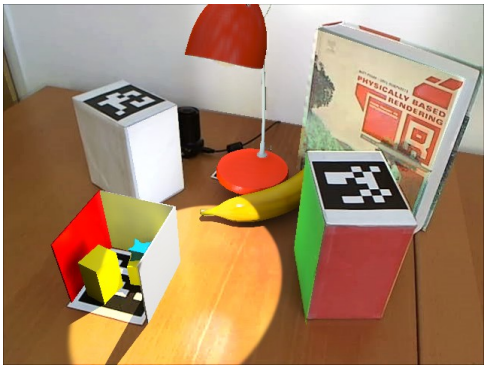
- Low-quality rendering $LQ = \neg GI \wedge \neg CM \wedge \neg CA$: All rendering modes are deactivated.
- Modes *GI*/*CM*/*CA*: These modes simply have one of the rendering modes (*GI*, *CM* or *CA*) activated.
- Mode $(GI \wedge CM)$: The rendered images contain shadows and indirect illumination. Moreover, adaptive color mapping is activated.
- Mode $(GI \wedge CA)$: Shadows and indirect illumination are activated including additional camera artifacts.
- Mode $(CM \wedge CA)$: Rendering without shadows and indirect illumination but with adaptive color mapping and camera artifacts.
- High-quality rendering $HQ = GI \wedge CM \wedge CA$: In this composition all rendering modes GI, CM and CA are activated.

To reduce the amount of trials, we did not compare each RMC against each other. Instead, we wanted to compare the RMCs to the low-quality and high-quality rendering mode compositions, as these were assumed to be the lower- and upper bound of the perceived quality. Therefore, the presented rendering mode composition pairs were $\{LQ, GI\}$, $\{LQ, CM\}$, $\{LQ, CA\}$, $\{LQ, GI \wedge CM\}$, $\{LQ, GI \wedge CA\}$, $\{LQ, CM \wedge CA\}$, $\{HQ, GI\}$, $\{HQ, CM\}$, $\{HQ, CA\}$, $\{HQ, GI \wedge CM\}$, $\{HQ, GI \wedge CA\}$, $\{HQ, CM \wedge CA\}$ and $\{LQ, HQ\}$.

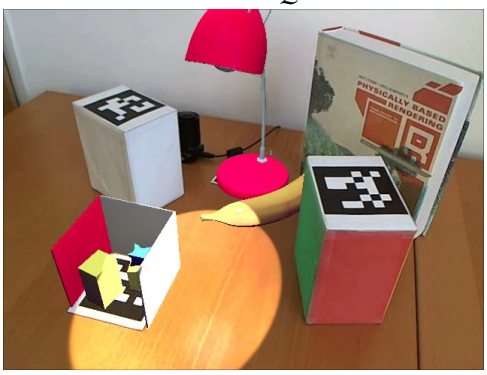
Each of these 13 RMC pairs was presented five times to a participant, leading to a total of 65 trials per participant. To avoid any patterns and learning effects between the trials, the selection



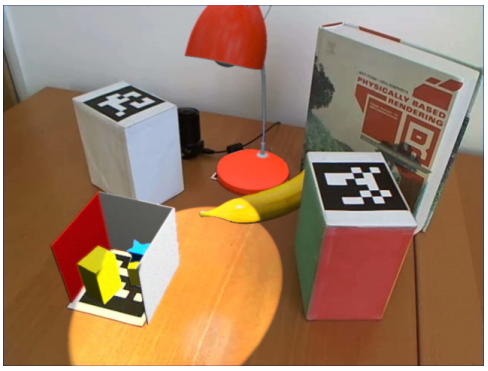
Mode *LQ*



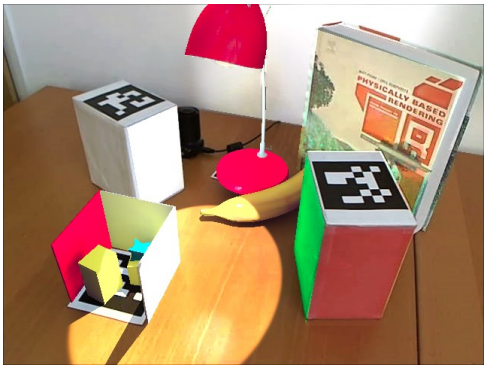
Mode *GI*



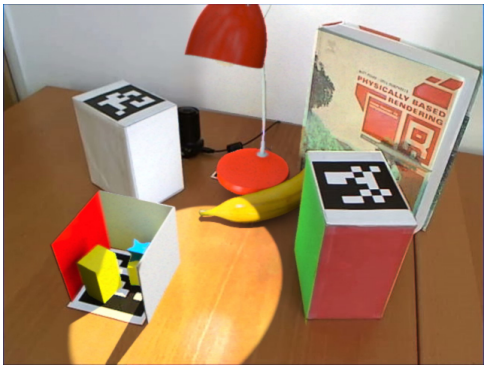
Mode *CM*



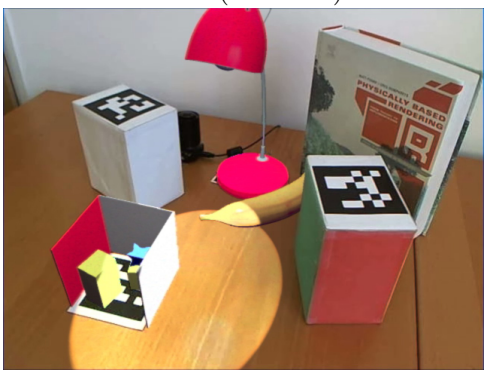
Mode *CA*



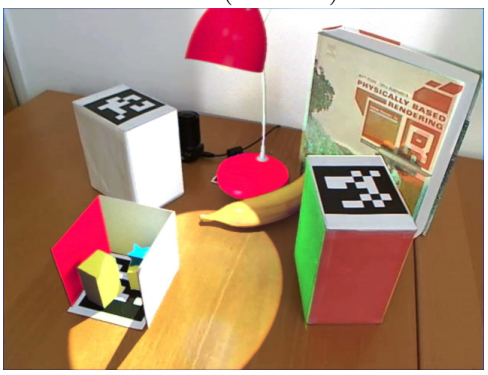
Mode $(GI \wedge CM)$



Mode $(GI \wedge CA)$



Mode $(CM \wedge CA)$



Mode *HQ*



**Figure 6.13:** In this figure, the different rendering mode compositions are shown.

of the scenes for each RMC and the order of the trial pairs itself were randomized. Figure 6.13 shows all RMCs with the same scene configuration.

### 6.4.2 Statistical Analysis

For the statistical analysis, we used the software *R* [109] in combination with the *prefmod* package [42]. This package was developed to evaluate data of paired comparison studies, where participants select preferences over two objects/items, in our case with images of different RMCs. The package is based on, and extends, the Bradley-Terry model [8]. This model can then be fitted to our data using a Generalized Linear Model (GLM). The resulting estimates of this model fit illustrate the influence of each rendering mode on the perceived quality. *prefmod* can be used to calculate so-called *worth* values, denoted as $\pi$, which give a ranking of each object/item compared to the other ones. More precisely, one can calculate the probability that item $i$ is preferred over item $j$ as follows:

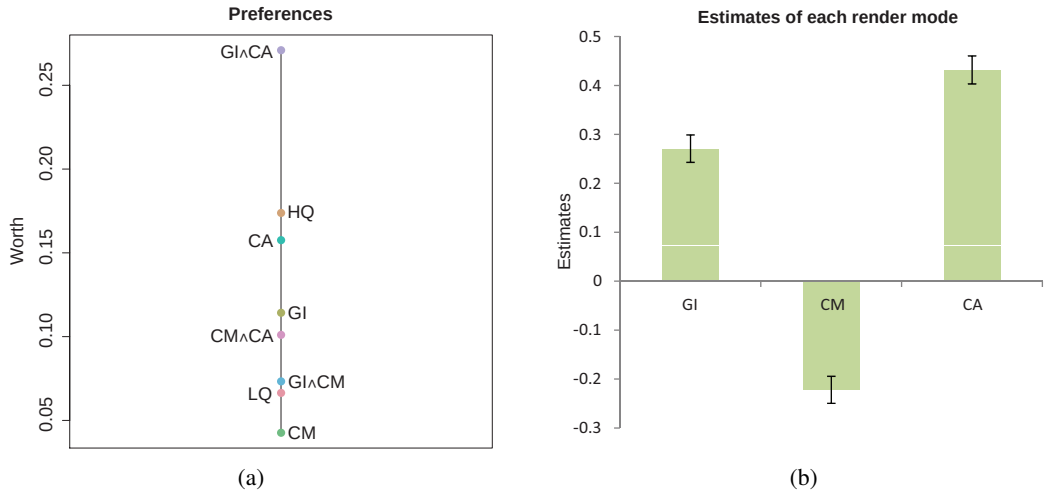
$$p(i \prec j|\pi_i, \pi_j) = \frac{\pi_i}{\pi_i + \pi_j} \tag{6.1}$$

To build these rankings it is not necessary to compare every object against every other object, which fits to our user-study design. In addition to the rankings, we also calculate the *coefficient of agreement* [61], denoted as $u$, which is an indicator how diverse the participants make their judgments. If all participants give the same answer to all paired comparisons, $u$ evaluates to one. However, if $u$ is smaller than zero, this means that there is practically no agreement among the participants.

### 6.4.3 Results and Discussion

The call for participation for the user study was distributed via email and Facebook. The participants were not aware of the different modes that were tested. From the 56 participants, 21 were female and 35 male, with an average age of 30.7 years. The collected data also showed that 103 people started the user study, but only 54% finished it. This shows that for such an experiment, the number of trials should be as low as possible. An alternative would be to pay for a service like Amazon's mechanical turk and gather information of a larger group, with a higher international disparity, or perform the study in a fixed laboratory setting.

First, we analyzed the ranking of each RMC. Since the RMCs are composed of the three rendering modes *GI*, *CM*, and *CA*, each rendering mode was added in the evaluation as object covariates. The resulting worth plot is shown in Figure 6.14(a). It clearly shows that our assumption that the low-quality RMC is the least ranked and that the high-quality RMC is the one with highest rank was not correct. In this worth plot, the adaptive color-mapping rendering feature *CM* has the lowest preference, and the highest preference over all participants has RMC ($GI \wedge CA$). Moreover, the coefficient of agreement evaluates to 0.12. Figure 6.14(b) shows the estimates of the fitted GLM for each rendering mode, and Table 6.1 shows that each rendering mode has a significant influence. Additionally, it shows that the rendering mode *CM* has a negative impact on the perceived quality.



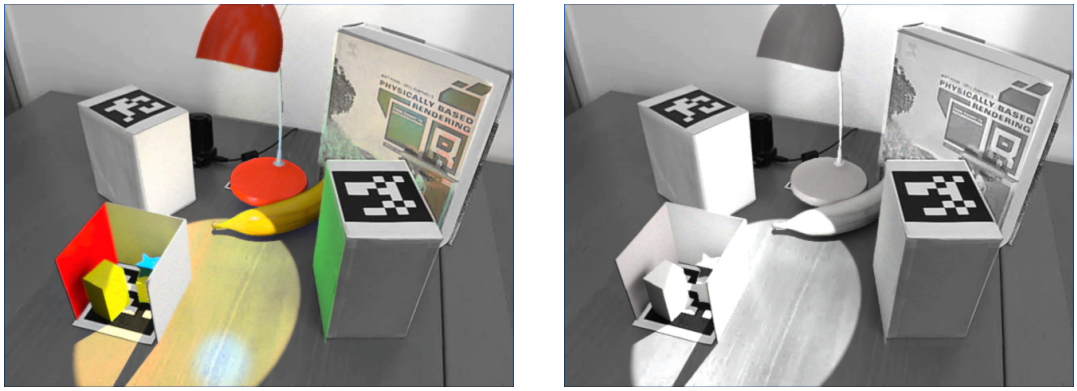

(a) (b)

**Figure 6.14:** Figure (a) shows the preferences for each rendering mode composition, based on the combination of the different rendering modes *GI*, *CM* and *CA*. Figure (b) shows the estimates of each rendering mode. Note that rendering mode *CM* has a negative impact on the perceived quality.

| Rendering Mode | Estimate | Std. Error | $Pr(>\|z\|)$ |
|---|---|---|---|
| *GI* | 0.27093 | 0.02801 | <2e-16 |
| *CM* | -0.22193 | 0.02762 | <2e-16 |
| *CA* | 0.43168 | 0.02863 | <2e-16 |

**Table 6.1:** This table shows the estimates for each rendering mode and the standard error. While rendering mode *GI* and *CA* have a positive impact, the influence of *CM* is negative. All three rendering modes have a significant effect on the perceived quality.

An explanation for the poor performance of the statistics-based color mapping method could be that the color mapping methods produce slightly desaturated or oversaturated images with standard camera settings. By study design, we decided to only use example images with standard camera settings because we wanted to avoid the situation that the camera image itself has too much influence on how the participants judge RMCs. Therefore, the study emphasizes the weak spot of our color-mapping method, since its strengths lie in cases where the camera has distorted color settings like with difficult lighting situations or changed parameter settings. For example, one could say that the virtual objects fit better into image 6.15(b) than image 6.15(a). However, in this case both camera images are desaturated. If this is not the case, the camera image itself could have too much influence on the decision, a circumstance we wanted to avoid. For a better understanding on the perceived quality of the statistics-based color mapping method, a separate study should be performed. However, for this study the results show that rendering mode CM has a negative impact and thus needs further improvements when the standard camera settings are used.

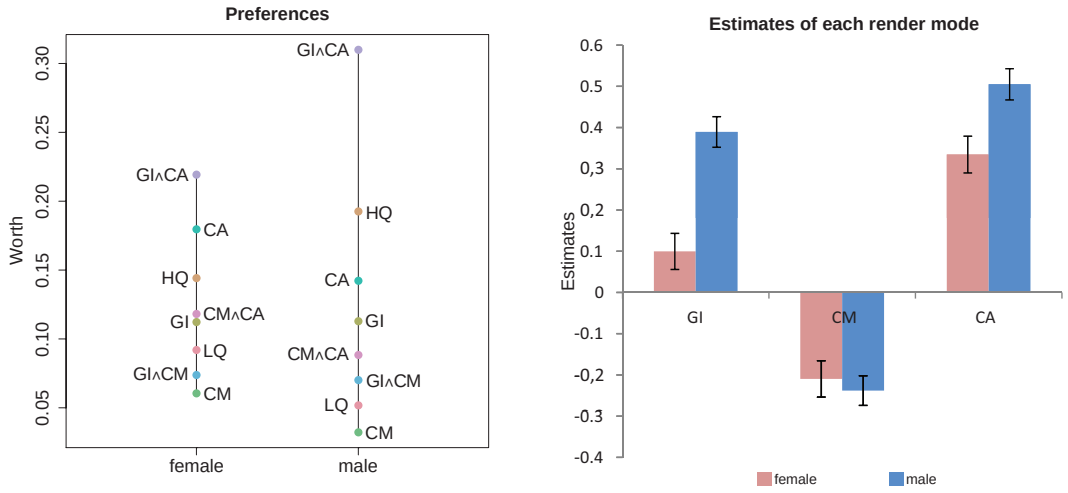



(a) (b)

**Figure 6.15:** In these images the camera image was desaturated. Image (a) shows rendering mode composition $(GI \wedge CA)$, and image (b) shows the high-quality rendering mode composition $HQ$. In such a setup one could perceive the high-quality rendering mode composition as having a better quality. However, for our user study, we only used images with the standard camera settings like in Figure 6.13.

Beside the poor performance of the color-mapping method, there are clear preferences for the rendering modes *GI* and *CA* compared to the low-quality rendering mode *LQ*. Moreover, the rendering mode composition $(GI \wedge CA)$ has the highest preference compared to all other rendering mode compositions and thus offers the highest perceived visual quality.

We also investigated whether the sex of the participants had any influence on the preferred rendering modes. For this, we created two separate data sets, one for all female and one for all male participants. Figure 6.16(a) shows the preferences according to the female and male participants. One can see that rankings of the female participants are closer together, meaning that the preferences are not as pronounced as they are for male participants. This is also reflected in the coefficients of agreement. For female participants the coefficient of agreement is only 0.06, while for male participants, it is three times higher, at $u = 0.18$. Tables 6.2 and 6.3 give the exact numbers of the estimates, errors and p-values for female and male participants. They also show that the significance of render mode *GI* is lower for female participants. However, this analysis and the results have to be treated very cautiously because we did not record whether the participants had a background in computer graphics or not and this could be an alternative explanation.




(a) (b)

**Figure 6.16:** Figure (a) shows the preferences for each rendering mode composition with a separate data analysis according to the sex of the participants. The ranking of the RMCs is evaluated based on the combination of the different rendering modes *GI*, *CM* and *CA*. The estimates for the rendering modes are shown in Figure (b).

| Render Mode | Estimate | Std. Error | $Pr(>\|z\|)$ |
|---|---|---|---|
| *GI* | 0,09972 | 0,04385 | 0.023 |
| *CM* | -0,20956 | 0,04395 | 1.86e-06 |
| *CA* | 0,33465 | 0,04465 | 6.65e-14 |

**Table 6.2:** This table shows the estimates for each render mode for female participants. Note that the impact of render mode *GI* is less pronounced compared to the impact for male participants as shown in Table 6.3.

| Render Mode | Estimate | Std. Error | $Pr(>\|z\|)$ |
|---|---|---|---|
| *GI* | 0.38945 | 0.03706 | < 2e-16 |
| *CM* | -0.23795 | 0.03588 | 3.31e-11 |
| *CA* | 0.50502 | 0.03786 | < 2e-16 |

**Table 6.3:** Table shows the estimates for each rendering mode for the male participants.

The take-home message of this user study is that the color-mapping method has a negative impact on the perceived visual quality and therefore needs to be improved. Furthermore, both rendering modes *GI* and *CA* have a positive impact on the perceived visual quality and if they are combined $(GI \wedge CA)$, the preference for this rendering mode composition even increases. The participants' gender might have an effect on the preference scores. However, to be certain about these results, more information about the participants should be collected in follow-up studies, to exclude other side effects.

## 6.5 Conclusion

In this chapter, we proposed a new research framework to perform perceptual experiments. To our knowledge, this is one of the first research frameworks that can take real-time global illumination and dynamic surrounding illumination effects into account. To test the research framework, a pilot user study was performed to investigate the influence of different rendering modes on user performance in five different tasks. The results indicated that there were no significant effects of these rendering modes on task performance.

In a follow-up user study, we implemented a web survey with side-by-side trials where participants had to decide on which image the virtual objects fit better into the real scene. Three different rendering modes could be switched on and off. In combination, these rendering modes summed up to a total of eight different rendering mode compositions. The results show that all rendering modes have a significant impact. However, the adaptive statistic-based color-mapping method *CM* has a negative impact and thus needs further improvement before it can be used in MR applications. The other two rendering modes, shadows and indirect illumination *GI*, as well as camera artifacts *CA* had a positive impact on the perceived quality of the images, and therefore should be used in a combined manner $(GI \wedge CA)$, to produce visually plausible images for mixed-reality scenarios.

The two user studies have very different results regarding the question whether the different rendering modes do have an impact or not. However, since performing a specific task and deciding which virtual objects fit better into a real scene are two totally different actions, they cannot be compared directly. So in conclusion, we can say that it depends on the application scenario whether the rendering modes have an impact or not.

CHAPTER 7

# Conclusion and Outlook

In this thesis, we presented a set of methods that try to improve the visual quality for mixed-reality applications. The contributions in this thesis can be split into three main topics, which were described in the *Reconstruction*, *Reciprocal Shading for Mixed Reality*, and the *Evaluation* chapters.

## 7.1 Conclusions

Chapter 4 described methods that focus on the reconstruction of the real environment. We presented two methods using the Microsoft Kinect sensor to reconstruct the real geometry. Another method in the reconstruction chapter focused on the estimation of the material characteristics at interactive frame rates, and finally, we presented two methods that adaptively try to find a color-mapping function so that the virtual objects look as if they were seen through the observing camera.

When dealing with the reconstruction of the real environment, one key observation is that it is highly dynamic. It constantly changes, be it due to the relocation of objects or due to changes of the incident illumination due to moving clouds. Therefore, it is important that the reconstruction should happen instantly during run time, in mixed-reality scenarios. Moreover, the reconstruction methods should react to changes in the real environment immediately. However, since there is no sensor device which can capture all available data of the real environment, the reconstruction methods itself, but also algorithms which use the reconstructed data as input, need to work reliably even though partial data about the real environment is missing. Only in case that the reconstruction methods are able to adapt to the dynamic behavior of the real environment, they will be sufficient for interactive mixed-reality applications. For example, in our geometry reconstruction methods we exploited temporal coherence to gather more information of the real environment than would be available from a single depth image of the Kinect sensor. However, by also adaptively reacting to changes in the environment, the methods do support dynamic scenes to a certain degree.

Chapter 5 described two methods for rendering reciprocal shading effects between real and virtual objects. The first method is called differential instant radiosity. It combines differential rendering and instant radiosity in such a way that it can be used for mixed-reality applications. The second method integrates reflective or refractive objects into the differential instant radiosity method.

Real-time global illumination algorithms simplify the rendering equation to gain rendering performance. For example, we used imperfect shadow maps for shadow calculation of indirect illumination and also kept the amount of virtual point lights as low as possible. However, these simplifications cause artifacts which are not desired. In differential rendering, two global illumination solutions need to be computed. To reduce the impact of such artifacts it is therefore necessary that both solutions are based on the same light paths. Moreover, if the computation of these two global illumination solutions happens in parallel, a lot of shading calculations need to be done only once. However, rendering methods for visually plausible mixed-reality applications are still computationally demanding. It is therefore advisable that temporal coherence between adjacent frames is exploited whenever possible. Throughout this thesis, we reused information from the previous frame at various stages in the pipeline. It is also important to note that reciprocal shading effects are not the only significant factor for visually plausible mixed reality. Real objects are seen through the observing camera, and the camera characteristics therefore plays an important role. Colorful virtual objects which are perfectly rendered might look disturbing when the real objects have slightly desaturated colors and suffer from noise. Therefore, it is important to photometrically register the virtual objects in the real environment, but then also apply the camera characteristics on them.

Chapter 6 presented two user studies that were performed to evaluate two of the described algorithms. The first user study revealed no significant effect of different rendering modes on task performance. In contrast, we could find significant effects on the perceived visual quality for different rendering effects in our second user study.

The results of these user studies cannot be really compared to each other, since the participants had to do different things in the experiments. However, they show that the influence of our improved rendering methods depend on the applications for which they are used. The second user study also revealed the importance of adding camera artifacts in a postprocessing step to the virtual objects to resemble the camera characteristics. In conclusion, to get high-quality visually plausible mixed-reality applications, methods that photometrically register virtual objects in the real environment should be combined with methods that apply the camera characteristics.

## 7.2 Outlook

The presented work in the field of visually plausible mixed reality has obviously lots of room for further research. Virtual objects are still recognized as being virtual due to several reasons. One reason is the registration of the virtual objects in the real world. In this thesis, we did not focus on this part but regardless of the rendering quality, if the virtual objects jitter due to tracking errors or float around with respect to the real world, they will be recognized immediately as artificial.

Another problem is that the rendering quality highly depends on the information that can be reconstructed from the real environment. Therefore, beside the reconstruction of the real objects, the incident illumination is captured using a camera with a fish-eye lens attached. Throughout this thesis, we used this camera in the real scene to capture the surrounding illumination. However, this is a very intrusive way for mixed-reality scenarios. In the future, this should not be necessary anymore and in fact, there is already work from Gruber et al. [37] that allows reconstructing the surrounding illumination without any intrusive camera or chrome sphere.

We proposed a method to reconstruct the geometry of real dynamic objects at high frame rates with the price that only those surface parts are reconstructed which are visible to the camera. Back-facing parts of objects are not apparent to the rendering system. KinectFusion [96] is capable of storing hidden surfaces that were seen previously. However, this method does not support dynamic scenes very well. For higher quality renderings, it would be good to have information about invisible geometry while still supporting dynamic scenes.

Our BRDF estimation method was a first attempt to perform material characterization at interactive frame rates. However, due to the simplifications, the quality of the results is limited and more research should be done to improve it. Maybe there is also a way to exploit temporal coherence in combination with KinectFusion to improve the quality of BRDF estimations similar to the work of Ritschel and Grosch [116].

The user study revealed that our camera-based color mapping approach does not improve the visual coherence of virtual objects in mixed-reality scenarios. Although we believe that this kind of methods will be important in future video see-through MR applications, the results clearly showed that the color-mapping quality needs to be improved. A big issue here is that this algorithm performs very well in cases where the camera colors are distorted or suddenly change. For example, if the camera changes its aperture or does some other automatic adjustments, our proposed methods can react to this. In such situations our method can render the virtual objects in a more plausible way than using a standard method. However, in the common indoor daylight illumination situation, our method unfortunately does not provide as good results as the tone-mapping operator from Reinhard et al. [113].

The differential instant radiosity method may suffer from visual artifacts since temporal coherence is exploited. Furthermore, only the Phong reflectance model or completely specular surfaces are supported. In the future, it would be nice to have different, for example physically plausible, reflectance models to support more types of surfaces. Furthermore, area light sources other than the environment light source are not supported yet. There are also methods that could improve the rendering performance significantly such as proposed by Lensing and Broll [76]. Additionally, specular objects do not work together with the proposed real-time reconstruction methods and thus, if in use, they demand manual pre-modeling of the scene. An alternative to rasterization-based rendering systems, like presented in this thesis, was proposed by Kan and Kaufmann [55, 54]. They use pure GPU ray-tracing techniques to photorealistically add virtual objects into real scenes. They are able to produce high-quality renderings but have lower performance compared to our methods. Finally, hybrid rendering systems, which use rasterizing and ray-tracing methods in combination, could improve the visual quality. Such a system was already developed under our supervision by Adam Celarek in his Bachelor thesis [10].

For further user studies, our evaluation framework should support stereo rendering. Since

the rendering method already pushes the limits of the graphics hardware, rendering a complete second frame is not possible yet while maintaining useable frame rates. However, many parts in the image pairs are the same and maybe a more sophisticated method can keep the additional rendering overhead quite small. Furthermore, the various topics of visually plausible mixed reality need further experiments. For BRDF estimation, it would be interesting to know how accurate they actually have to be. For example, a user study could measure thresholds at which point the material characteristics are recognized as being wrong in a similar way to the study of Lopez-Moreno et al. [84]. Such thresholds could then be used to decide whether an interactive BRDF estimation method has a sufficient estimation quality or not. Similar thresholds could be measured for the color-mapping approaches to know at which point the colors of virtual and real objects differ too much. These are just two possible further user studies related to our work. Of course there are plenty of other studies necessary to understand how human perception works and which elements can be exploited for visually plausible mixed-reality applications.

In summary, we have demonstrated that the differential instant radiosity method significantly improves the visual plausibility of virtual objects while still maintaining high frame rates. Therefore, it provides a much better illusion than conventional mixed-reality applications.

# Bibliography


[1] Kusuma Agusanto, Li Li, Zhu Chuangui, and Ng Wan Sing. Photorealistic rendering for augmented reality using environment illumination. In *Proceedings of the 2nd IEEE/ACM International Symposium on Mixed and Augmented Reality*. In ISMAR '03. IEEE Computer Society, Washington, DC, USA, 2003, pages 208–216.

[2] Tomas Akenine-Möller, Eric Haines, and Natty Hoffman. *Real- Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008, pages 392–395.

[3] Nicolas Alt, Patrick Rives, and Eckehard Steinbach. Reconstruction of transparent objects in unstructured scenes with a depth camera. In *Proceedings of the 20th IEEE International Conference on Image Processing*. In ICIP '13. Melbourne, Australia, September 2013.

[4] Ronald T. Azuma. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385, August 1997.

[5] Zhen Bai and Alan F. Blackwell. Analytic review of usability evaluation in ISMAR. *Interacting with Computers*, 24(6):450–460, November 2012.

[6] Simone Bianco, Raimondo Schettini, and Leonardo Vanneschi. Empirical modeling for colorimetric characterization of digital cameras. In *Proceedings of the 16th IEEE International Conference on Image Processing*. In ICIP'09. IEEE Press, Cairo, Egypt, 2009, pages 3469–3472.

[7] Samuel Boivin and Andre Gagalowicz. Image-based rendering of diffuse, specular and glossy surfaces from a single image. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. In SIGGRAPH '01. ACM, New York, NY, USA, 2001, pages 107–116.

[8] Ralph A. Bradley and Milton E. Terry. Rank Analysis of Incomplete Block Designs: I. The Method of Paired Comparisons. *Biometrika*, 39(3/4), 1952.

[9] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[10] Adam Celarek. Merging ray tracing and rasterization in mixed reality. Bachelor's thesis. Favoritenstrasse 9-11/186, A-1040 Vienna, Austria: Institute of Computer Graphics and Algorithms, Vienna University of Technology, November 2012.

[11] Vien Cheung, Stephen Westland, David Connah, and Caterina Ripamonti. A comparative study of the characterisation of colour cameras by means of neural networks and polynomial transforms. *Coloration Technology*, 120(1):19–25, 2004.

[12] Petrik Clarberg, Wojciech Jarosz, Tomas Akenine-Möller, and Henrik Wann Jensen. Wavelet importance sampling: efficiently evaluating products of complex functions. In *ACM SIGGRAPH 2005 Papers*. In SIGGRAPH '05. ACM, Los Angeles, California, 2005, pages 1166–1175.

[13] Daniel Cohen-Or, Olga Sorkine, Ran Gal, Tommer Leyvand, and Ying-Qing Xu. Color harmonization. In *ACM SIGGRAPH 2006 Papers*. In SIGGRAPH '06. ACM, Boston, Massachusetts, 2006, pages 624–630.

[14] Robert L. Cook and Kenneth E. Torrance. A reflectance model for computer graphics. In *SIGGRAPH '81: Proceedings of the 8th annual conference on Computer graphics and interactive techniques*. ACM, Dallas, Texas, United States, 1981, pages 307–316.

[15] Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, and Elmar Eisemann. Interactive indirect illumination using voxel-based cone tracing: an insight. In *ACM SIGGRAPH 2011 talks*. In SIGGRAPH '11. ACM, Vancouver, British Columbia, Canada, 2011, 20:1–20:1.

[16] James E. Cutting. How the eye measures reality and virtual reality. *Behavior Research Methods, Instruments, & Computers*, 29(1):27–36, 1997.

[17] Carsten Dachsbacher, Jaroslav Krivánek, Milos Hasan, Adam Arbree, Bruce Walter, and Jan Novák. Scalable realistic rendering with many-light methods. In *EG 2013 - STARs*. M. Sbert and L. Szirmay-Kalos, editors. Eurographics Association, Girona, Spain, 2012, pages 23–38.

[18] Carsten Dachsbacher and Marc Stamminger. Reflective shadow maps. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*. In I3D '05. ACM, Washington, District of Columbia, 2005, pages 203–231.

[19] Carsten Dachsbacher and Marc Stamminger. Splatting indirect illumination. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*. In I3D '06. ACM, Redwood City, California, 2006, pages 93–100.

[20] Naveen Dachuri, Seung Man Kim, and Kwan H. Lee. Estimation of few light sources from environment maps for fast realistic rendering. In *Proceedings of the 2005 international conference on Augmented tele-existence*. In ICAT '05. Christchurch, New Zealand, 2005, pages 265–266.

[21] Paul Debevec. A median cut algorithm for light probe sampling. In *ACM SIGGRAPH 2005 Posters*. In SIGGRAPH '05. ACM, Los Angeles, California, 2005.

[22] Paul Debevec. Rendering synthetic objects into real scenes: bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, 1998, pages 189–198.

[23] Balaji Dhanasekaran and Norman Rubin. A new method for GPU based irregular reductions and its application to k-means clustering. In *Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units*. In GPGPU-4. ACM, Newport Beach, California, 2011, 2:1–2:8.

[24] David Drascic and Paul Milgram. Perceptual issues in augmented reality. In *SPIE Volume 2653: Stereoscopic Displays and Virtual Reality Systems III*, 1996, pages 123–134.

[25] George Drettakis, Luc Robert, and Sylvain Bougnoux. Interactive common illumination for computer augmented reality. In *Proceedings of the Eurographics Workshop on Rendering Techniques '97*. Springer-Verlag, London, UK, UK, June 1997, pages 45–56.

[26] Andreas Dünser and Mark Billinghurst. Evaluating augmented reality systems. In Borko Furht, editor, *In Handbook of Augmented Reality, Borko Furht*, part Part 1, pages 289–307. Springer, 2011.

[27] Mohamed Elhelw, Marios Nicolaou, Adrian Chung, Guang-Zhong Yang, and M. Stella Atkins. A gaze-based study for investigating the perception of visual realism in simulated scenes. *ACM Transactions on Applied Perception*, 5(1):3:1–3:20, January 2008.

[28] Mark D. Fairchild. A color scientist looks at video. *3rd International Workshop on Video Processing and Quality Metrics (VPQM)*, 2007.

[29] Wenbin Fang, Ka Keung Lau, Mian Lu, Xiangye Xiao, Chi Kit Lam, Philip Yang Yang, Bingsheng He, Qiong Luo, Pedro V. Sander, and Ke Yang. Parallel data mining on graphics processors. Technical report. Department of Computer Science, Engineering, Hong Kong University of Science, and Technology, October 2008.

[30] Reza Farivar, Daniel Rebolledo, Ellick Chan, and Roy Campbell. A parallel implementation of k-means clustering on GPUs. In *WorldComp 2008*. Las Vegas, Nevada, July 2008.

[31] James A. Ferwerda. Three varieties of realism in computer graphics. In *Society of Photo-Optical Instrumentation Engineers*. Volume 5007. In SPIE '03 Conference Series, June 2003, pages 290–297.

[32] Jan Fischer, Dirk Bartz, and Wolfgang Strasser. Enhanced visual realism by incorporating camera image effects. In *Proceedings of the 5th IEEE and ACM International Symposium on Mixed and Augmented Reality*. In ISMAR '06. IEEE Computer Society, Washington, DC, USA, 2006, pages 205–208.

[33] Alain Fournier, Atjeng S. Gunawan, and Chris Romanzin. Common illumination between real and computer generated scenes. In *Proceedings of Graphics Interface '93*, May 1993, pages 254–262.

[34] Thorsten Grosch. Differential photon mapping: consistent augmentation of photographs with correction of all light paths. In *Eurographics 2005 Short Papers, Trinity College*, 2005.

[35] Thorsten Grosch, Tobias Eble, and Stefan Mueller. Consistent interactive augmentation of live camera images with correct near-field illumination. In *VRST '07: Proceedings of the 2007 ACM symposium on Virtual Reality Software and Technology*. Newport Beach, California, 2007, pages 125–132.

[36] Lukas Gruber, Denis Kalkofen, and Dieter Schmalstieg. Color harmonization for augmented reality. In *Proceedings of the 9th IEEE International Symposium on Mixed and Augmented Reality*. In ISMAR '10. IEEE Computer Society, Seoul, South Korea, 2010, pages 227–228.

[37] Lukas Gruber, Thomas Richter-Trummer, and Dieter Schmalstieg. Real-time photometric registration from arbitrary geometry. In *Proceedings of the 2012 IEEE International Symposium on Mixed and Augmented Reality*. In ISMAR '12. IEEE Computer Society, Washington, DC, USA, 2012, pages 119–128.

[38] Mark Grundland and Neil A. Dodgson. Color histogram specification by histogram warping. In *Proceedings of society of Photo-Optical Instrumentation Engineers*. Volume 5667. In SPIE '05 Conference Series, 2005, pages 610–621.

[39] Paul Guerrero. Approximative real-time soft shadows and diffuse reflections in dynamic scenes. Master's thesis. Favoritenstrasse 9-11/186, A-1040 Vienna, Austria: Institute of Computer Graphics and Algorithms, Vienna University of Technology, October 2007.

[40] Jungong Han, Ling Shao, Dong Xu, and Jamie Shotton. Enhanced computer vision with Microsoft Kinect sensor: a review. *IEEE Transactions on Cybernetics*, 2013.

[41] Timothy J. Hattenberger, Mark D. Fairchild, Garrett M. Johnson, and Carl Salvaggio. A psychophysical investigation of global illumination algorithms used in augmented reality. *ACM Transactions on Applied Perception*, 6(1):2:1–2:22, February 2009.

[42] Reinhold Hatzinger and Regina Dittrich. prefmod: an R package for modeling preferences based on paired comparisons, rankings, or ratings. *Journal of Statistical Software*, 48(10):1–31, 2012.

[43] Vlastimil Havran, Miloslaw Smyk, Grzegorz Krawczyk, Karol Myszkowski, and Hans-Peter Seidel. Importance sampling for video environment maps. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches*. Los Angeles, California, 2005, page 109.

[44] Paul S. Heckbert. Simulating global illumination using adaptive meshing. PhD thesis. EECS Department, University of California, June 1991.

[45] Sebastian Heymann, Aljoscha Smolic, Karsten Müller, and Bernd Froehlich. Illumination reconstruction from real-time video for interactive augmented reality. In *International Workshop on Image Analysis for Multimedia Interactive Services*. In WIAMIS '05. Montreux, 2005, pages 1–4.

[46] Guowei Hong, M. Ronnier Luo, and Peter A. Rhodes. A study of digital camera colorimetric characterisation based on polynomial modelling. *Color Research and Application*, 26(1):76–84, 2001.

[47] Bai Hong-tao, He Li-li, Ouyang Dan-tong, Li Zhan-shan, and Li He. K-Means on Commodity GPUs with CUDA. In *Proceedings of the 2009 WRI World Congress on Computer Science and Information Engineering*. Volume 3. In CSIE '09. IEEE Computer Society, Washington, DC, USA, 2009, pages 651–655.

[48] Helen H. Hu, Amy A. Gooch, Sarah H. Creem-Regehr, and William B. Thompson. Visual cues for perceiving distances from objects to surfaces. *Presence: Teleoperators and Virtual Environments*, 11(6):652–664, December 2002.

[49] Geoffrey S. Hubona, Philip N. Wheeler, Gregory W. Shirah, and Matthew Brandt. The relative contributions of stereo, lighting, and background scenes in promoting 3D depth visualization. *ACM Transactions on Computer-Human Interaction*, 6(3):214–242, September 1999.

[50] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. KinectFusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*. In UIST '11. ACM, Santa Barbara, California, USA, 2011, pages 559–568.

[51] Katrien Jacobs and Céline Loscos. Classification of illumination methods for mixed-reality. *Computer Graphics Forum*, 25:29–51, 1, March 2006.

[52] Bernhard Kainz, Stefan Hauswiesner, Gerhard Reitmayr, Markus Steinberger, Raphael Grasset, Lukas Gruber, Eduardo Veas, Denis Kalkofen, Hartmut Seichter, and Dieter Schmalstieg. OmniKinect: real-time dense volumetric data acquisition and applications. In *Proceedings of the 18th ACM symposium on Virtual Reality Software and Technology*. In VRST '12. ACM, Toronto, Ontario, Canada, 2012, pages 25–32.

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

[54] Peter Kán and Hannes Kaufmann. Differential irradiance caching for fast high-quality light transport between virtual and real worlds. In *Proceedings of International Symposium on Mixed and Augmented Reality*. In ISMAR '13'. IEEE Computer Society, 2013, pages 133–141.

[55] Peter Kán and Hannes Kaufmann. High-quality reflections, refractions, and caustics in Augmented Reality and their contribution to visual coherence. In *Proceedings of the 2012 IEEE International Symposium on Mixed and Augmented Reality*. In ISMAR '12. IEEE Computer Society, Washington, DC, USA, 2012, pages 99–108.

[56] Henry R. Kang. *Computational Color Technology (SPIE Press Monograph Vol. PM159)*. SPIE- International Society for Optical Engineering, 2006.

[57] Anton Kaplanyan. Light Propagation Volumes in CryEngine 3. In *SIGGRAPH '09 course*. New Orleans, Louisiana, 2009.

[58] Anton Kaplanyan and Carsten Dachsbacher. Cascaded light propagation volumes for real-time indirect illumination. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D graphics and games*. In I3D '10. ACM, Washington, D.C., 2010, pages 99–107.

[59] Kevin Karsch, Varsha Hedau, David Forsyth, and Derek Hoiem. Rendering synthetic objects into legacy photographs. In *Proceedings of the 2011 SIGGRAPH Asia Conference*. In SA '11. ACM, Hong Kong, China, 2011, 157:1–157:12.

[60] Alexander Keller. Instant radiosity. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. In SIGGRAPH '97, 1997, pages 49–56.

[61] Maurice Kendall and Jean D. Gibbons. *Rank Correlation Methods*. A Charles Griffin Title, 5th edition, September 13, 1990.

[62] Georg Klein and Tom Drummond. Sensor fusion and occlusion refinement for tablet-based AR. In *Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality*. In ISMAR '04. IEEE Computer Society, Washington, DC, USA, 2004, pages 38–47.

[63] Georg Klein and David Murray. Compositing for small cameras. In *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*. In ISMAR '08. IEEE Computer Society, Washington, DC, USA, 2008, pages 57–60.

[64] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proceedings of the 2007 6th IEEE/ACM International Symposium on Mixed and Augmented Reality*. In ISMAR '07. IEEE Computer Society, Washington, DC, USA, 2007, pages 1–10.

[65] Georg Klein and David W. Murray. Simulating low-cost cameras for augmented reality compositing. *IEEE Transactions on Visualization and Computer Graphics*, 16:369–380, 2010.

[66] Martin Knecht, Georg Tanzmeister, Christoph Traxler, and Michael Wimmer. Interactive BRDF estimation for mixed-reality applications. *Journal of WSCG*, 20(1):47–56, June 2012.

[67] Martin Knecht, Christoph Traxler, Oliver Mattausch, Werner Purgathofer, and Michael Wimmer. Differential instant radiosity for mixed reality. In *Proceedings of the 9th IEEE International Symposium on Mixed and Augmented Reality*. In ISMAR '10. IEEE Computer Society, Seoul, South Korea, 2010, pages 99–108.

[68] Martin Knecht, Christoph Traxler, Oliver Mattausch, and Michael Wimmer. Reciprocal shading for mixed reality. *Computers & Graphics*, 36(7):846–856, November 2012.

[69] Martin Knecht, Christoph Traxler, Werner Purgathofer, and Michael Wimmer. Adaptive camera-based color mapping for mixed-reality applications. In *Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality*. In ISMAR '11. IEEE Computer Society, Washington, DC, USA, 2011, pages 165–168.

[70] Martin Knecht, Christoph Traxler, Christoph Winklhofer, and Michael Wimmer. Reflective and refractive objects for mixed reality. *IEEE Transactions on Visualization and Computer Graphics*, 19(4):576–582, April 2013.

[71] Matthias Korn, Maik Stange, Andreas von Arb, Lisa Blum, Michael Kreil, Kathrin-Jennifer Kunze, Jens Anhenn, Timo Wallrath, and Thorsten Grosch. Interactive augmentation of live images using a HDR stereo camera. *Journal of virtual reality and broadcasting*, 4(12):107–118, January 2007. Jens Herder, editor.

[72] E. Kruijff, J.E. Swan, and S. Feiner. Perceptual issues in augmented reality revisited. In *Proceedings of the 9th IEEE International Symposium on Mixed and Augmented Reality*. In ISMAR '10. IEEE Computer Society, Seoul, South Korea, 2010, pages 3–12.

[73] Samuli Laine, Hannu Saransaari, Janne Kontkanen, Jaakko Lehtinen, and Timo Aila. Incremental instant radiosity for real-time indirect illumination. In *Proceedings of Eurographics Symposium on Rendering 2007*, 2007, pages 277–286.

[74] Hayden Landis. Production-ready global illumination. *SIGGRAPH Course Notes*, 16, July 2002.

[75] Cha Lee, Gustavo A. Rincon, Greg Meyer, Tobias Hollerer, and Doug A. Bowman. The effects of visual realism on search tasks in mixed reality simulation. *IEEE Transactions on Visualization and Computer Graphics*, 19(4):547–556, April 2013.

[76] Philipp Lensing and Wolfgang Broll. Efficient shading of indirect illumination applying reflective shadow maps. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D graphics and games*. In I3D '13. ACM, Orlando, Florida, 2013, pages 95–102.

[77] Philipp Lensing and Wolfgang Broll. Fusing the real and the virtual: a depth-camera based approach to mixed reality. In *Proceedings of the 10th IEEE International Symposium on Mixed and Augmented Reality*. In ISMAR '11. IEEE Computer Society, Washington, DC, USA, 2011, pages 261–262.

[78] Philipp Lensing and Wolfgang Broll. Instant indirect illumination for dynamic mixed reality scenes. In *Proceedings of the 2012 IEEE International Symposium on Mixed and Augmented Reality*. In ISMAR '12. IEEE Computer Society, Washington, DC, USA, 2012, pages 109–118.

[79] Robert R. Lewis. Making shaders more physically plausible. Technical report. Vancouver, BC, Canada, Canada, 1993.

[80] Chang Jiang Li, Zhong Zhang, T. Imamura, and T. Miyaki. An efficient BRDF acquisition for glossy surface. In *Proceedings of the 2010 3rd International Conference on Advanced Computer Theory and Engineering*. Volume 2. In ICACTE '10, 2010, pages 141–145.

[81] You Li, Kaiyong Zhao, Xiaowen Chu, and Jiming Liu. Speeding up k-means algorithm by GPUs. In *Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology*. In CIT '10. IEEE Computer Society, Washington, DC, USA, 2010, pages 115–122.

[82] Sebastian Lieberknecht, Andrea Huber, Slobodan Ilic, and Selim Benhimane. RGB-D camera-based parallel tracking and meshing. In *Proceedings of the 10th IEEE International Symposium on Mixed and Augmented Reality*. In ISMAR '11. IEEE Computer Society, Washington, DC, USA, 2011, pages 147–155.

[83] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, September 2006.

[84] Jorge Lopez-Moreno, Veronica Sundstedt, Francisco Sangorrin, and Diego Gutierrez. Measuring the perception of light inconsistencies. In *Proceedings of the 7th Symposium on Applied Perception in Graphics and Visualization*. In APGV '10. ACM, Los Angeles, California, 2010, pages 25–32.

[85] Cindee Madison, Daniel J Kersten, William B Thompson, Peter Shirley, and Brian Smits. The use of subtle illumination cues for human judgement of spatial layout. Technical Report (UUCS-99-001). Computer Science Deptartment, University of Utah, 1999.

[86] Claus B. Madsen and Michael Nielsen. Towards probe-less augmented reality - a position paper. In *Third International Conference on Computer Graphics Theory and Applications*. In GRAPP '08. INSTICC - Institute for Systems, Technologies of Information, Control, and Communication, Funchal, Madeira, Portugal, April 7, 2008, pages 255–261.

[87] Morgan McGuire and David Luebke. Hardware-accelerated global illumination by image space photon mapping. In *HPG '09: Proceedings of the Conference on High Performance Graphics 2009*. New Orleans, Louisiana, 2009, pages 77–89.

[88] Stephan Meister, Shahram Izadi, Pushmeet Kohli, Martin Hämmerle, Carsten Rother, and Daniel Kondermann. When can we use KinectFusion for ground truth acquisition? In *Workshop on Color-Depth Camera Fusion in Robotics, IEEE International Conference on Intelligent Robots and Systems*. 1, 2012.

[89] Bruno Mercier, Daniel Meneveaux, and Alain Fournier. A framework for automatically recovering object shape, reflectance and light sources from calibrated images. *International Journal of Computer Vision*, 73:77–93, 1, June 2007.

[90] Microsoft. Kinect SDK: http://research.microsoft.com/en-us/um/redmond/projects/kinectsdk/. [Online; accessed 30-September-2013].

[91] Eihachiro Nakamae, Koichi Harada, Takao Ishizaki, and Tomoyuki Nishita. A montage method: the overlaying of the computer generated images onto a background photograph. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, 1986, pages 207–214.

[92] Gaku Nakano, Itaru Kitahara, and Yuichi Ohta. Generating perceptually-correct shadows for mixed reality. In *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*. In ISMAR '08. IEEE Computer Society, Washington, DC, USA, 2008, pages 173–174.

[93] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[94] Laszlo Neumann and Attila Neumann. Color style transfer techniques using hue, lightness and saturation histogram matching. In *Computational Aesthetics in Graphics, Visualization and Imaging 2005*. Girona, Spain, May 2005, pages 111–122.

[95] Richard A. Newcombe and Andrew J. Davison. Live dense reconstruction with a single moving camera. In *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition*. In CVPR '10. IEEE, San Francisco, CA, USA, June 2010, pages 1498–1505.

[96] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. KinectFusion: real-time dense surface mapping and tracking. In *Proceedings of the 10th IEEE International Symposium on Mixed and Augmented Reality*. In ISMAR '11. IEEE Computer Society, Washington, DC, USA, 2011, pages 127–136.

[97] Greg Nichols, Jeremy Shopf, and Chris Wyman. Hierarchical image-space radiosity for interactive global illumination. *Computer Graphics Forum*, 28:1141–1149, June 2009.

[98] Greg Nichols and Chris Wyman. Interactive indirect illumination using adaptive multiresolution splatting. *IEEE Transactions on Visualization and Computer Graphics*, 16(5):729–741, September 2010.

[99] Greg Nichols and Chris Wyman. Multiresolution splatting for indirect illumination. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*. In I3D '09. ACM, Boston, Massachusetts, 2009, pages 83–90.

[100] Fred E. Nicodemus. Reflectance nomenclature and directional reflectance and emissivity. *Applied Optics*, 9(6):1474–1475, 1970.

[101] Jan Novák, Thomas Engelhardt, and Carsten Dachsbacher. Screen-space bias compensation for interactive high-quality global illumination with virtual point lights. In *Symposium on Interactive 3D graphics and games*. In I3D '11. ACM, San Francisco, California, 2011, pages 119–124.

[102] Michael Oren and Shree K. Nayar. Generalization of Lambert's reflectance model. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. ACM, New York, NY, USA, 1994, pages 239–246.

[103] Francisco Ortiz and Fernando Torres. Automatic detection and elimination of specular reflectance in color images by means of MS diagram and vector connected filters. *IEEE Transactions on Systems Man and Cybernetics Part C Applications and Reviews*, 36(5):681–687, September 2006.

[104] Saulo Pessoa, Guilherme Moura, Joao Lima, Veronica Teichrieb, and Judith Kelner. Photorealistic rendering for augmented reality: a global illumination and BRDF solution. In *2010 IEEE Virtual Reality Conference (VR)*. IEEE, Boston, MA, USA, March 2010, pages 3–10.

[105] Bui Tuong Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18:311–317, 6, June 1975.

[106] Sören Pirk. GPU-based rendering of reflective and refractive objects in augmented reality environments. Master's thesis. University of Applied Sciences, Oldenburg, Germany, 2007.

[107] Voicu Popescu, Chunhui Mei, Jordan Dauble, and Elisha Sacks. Reflected-scene impostors for realistic reflections at interactive rates. *Computer Graphics Forum*, 25(3):313–322, 2006.

[108] Tania Pouli and Erik Reinhard. Progressive color transfer for images of arbitrary dynamic range. *Computers & Graphics*, 35(1):67–80, 2011.

[109] R Core Team. *R: a language and environment for statistical computing*. R Foundation for Statistical Computing. Vienna, Austria, 2013.

[110] Paul Rademacher, Jed Lengyel, Edward Cutrell, and Turner Whitted. Measuring the perception of visual realism in images. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*. Springer-Verlag, London, UK, UK, 2001, pages 235–248.

[111] Erik Reinhard, Michael Ashikhmin, Bruce Gooch, and Peter Shirley. Color transfer between images. *IEEE Computer Graphics and Applications*, 21(5):34–41, September 2001.

[112] Erik Reinhard and Tania Pouli. Colour spaces for colour transfer. In *Proceedings of the Third international conference on Computational color imaging*. In CCIW'11. Springer-Verlag, Milan, Italy, 2011, pages 1–15.

[113] Erik Reinhard, Michael Stark, Peter Shirley, and James Ferwerda. Photographic tone reproduction for digital images. *ACM Transactions on Graphics*, 21(3):267–276, 2002.

[114] Irene Reisner-Kollmann. Multi-view 3d reconstruction based on human-like perception. PhD thesis. Favoritenstrasse 9-11/186, A-1040 Vienna, Austria: Institute of Computer Graphics and Algorithms, Vienna University of Technology, February 2013.

[115] T. Ritschel, T. Engelhardt, T. Grosch, H.-P. Seidel, J. Kautz, and C. Dachsbacher. Micro-rendering for scalable, parallel final gathering. In *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers*. Yokohama, Japan, 2009, pages 1–8.

[116] T. Ritschel and T. Grosch. On-line estimation of diffuse materials. In *Dritter Workshop Virtuelle und Erweiterte Realitaet der GI-Fachgruppe VR/AR*. Volume 3, 2006, pages 95–106.

[117] T. Ritschel, T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz. Imperfect shadow maps for efficient computation of indirect illumination. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*. Singapore, 2008, pages 1–8.

[118] Tobias Ritschel, Carsten Dachsbacher, Thorsten Grosch, and Jan Kautz. The state of the art in interactive global illumination. *Computer Graphics Forum*, 31(1):160–188, February 2012.

[119] Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. Approximating dynamic global illumination in image space. In *Proceedings of the 2009 symposium on Interactive 3d graphics and games*. In I3D '09. Boston, Massachusetts, 2009, pages 75–82.

[120] Carlos Ureña Rosana Montes. An overview of BRDF models. Technical report. Deptartment Lenguajes y Sistemas Informáticos University of Granada, Granada, Spain, March 2012.

[121] Daniel L. Ruderman, Thomas W. Cronin, and Chuan-Chin Chiao. Statistics of cone responses to natural images: implications for visual coding. *Journal of the Optical Society of America A*, 15:2036–2045, 1998.

[122] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation*. In ICRA '11. Shanghai, China, May 2011.

[123] Imari Sato, Yoichi Sato, and Katsushi Ikeuchi. Acquiring a radiance distribution to superimpose virtual objects onto a real scene. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):1–12, 1999.

[124] Yoichi Sato, Mark D. Wheeler, and Katsushi Ikeuchi. Object shape and reflectance modeling from observation. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. In SIGGRAPH '97. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1997, pages 379–387.

[125] Nikhil Sawant and Niloy J. Mitra. Color harmonization for videos. In *Indian Conference on Computer Vision, Graphics and Image Processing*. In ICCVGIP '08, 2008, pages 576–582.

[126] Daniel Scherzer, Stefan Jeschke, and Michael Wimmer. Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In *Rendering Techniques 2007 (Proceedings Eurographics Symposium on Rendering)*. Eurographics Association, Grenoble, France, 2007, pages 45–50.

[127] Dieter Schmalstieg. Studierstube Tracker. `http://handheldar.icg.tugraz.at/stbtracker.php`. [Online; accessed 30-September-2013].

[128] B. Segovia, J. C. Iehl, R. Mitanchey, and B. Péroche. Non-interleaved deferred shading of interleaved sample patterns. In *Proceedings of the 21st ACM SIGGRAPH / EUROGRAPHICS symposium on Graphics hardware*. In GH '06. Vienna, Austria, 2006, pages 53–60.

[129] Benjamin Segovia. Interactive light transport with virtual point lights. Thèse de Doctorat en Informatique. Université Lyon 1, October 2007.

[130] Musawir A. Shah, 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.

[131] Stefan Spelitz. Color distribution transfer for mixed-reality applications. Bachelor's thesis. Favoritenstrasse 9-11/186, A-1040 Vienna, Austria: Institute of Computer Graphics and Algorithms, Vienna University of Technology, October 2012.

[132] Andrei State, 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*. In SIGGRAPH '96. ACM, New York, NY, USA, 1996, pages 429–438.

[133] Natsuki Sugano, Hirokazu Kato, and Keihachiro Tachibana. The effects of shadow representation of virtual objects in augmented reality. In *Proceedings of the 2nd IEEE/ACM International Symposium on Mixed and Augmented Reality*. In ISMAR '03. IEEE Computer Society, Washington, DC, USA, 2003, pages 76–.

[134] Peter Supan, Ines Stuppacher, and Michael Haller. Image based shadowing in real-time augmented reality. *International Journal of Virtual Reality*, 5(3):1–7, 2006.

[135] Richard Szeliski. Rapid octree construction from image sequences. *CVGIP: Image Understanding*, 58:23–32, 1, July 1993.

[136] Takehiro Tawara and Kenji Ono. An application of photo realistic water surface interaction using mixed reality. In *Proceedings of the Sixth Workshop on Virtual Reality Interactions and Physical Simulations*. In VRIPHYS '09. Eurographics Association, Karlsruhe, Germany, 2009, pages 59–65.

[137] Russell M. Taylor II, Thomas C. Hudson, Adam Seeger, Hans Weber, Jeffrey Juliano, and Aron T. Helser. VRPN: a device-independent, network-transparent VR peripheral system. In *Proceedings of the ACM symposium on Virtual reality software and technology*. In VRST '01. ACM, Baniff, Alberta, Canada, 2001, pages 55–61.

[138] Sinje Thiedemann, Niklas Henrich, Thorsten Grosch, and Stefan Müller. Voxel-based global illumination. In *Symposium on Interactive 3D graphics and games*. In I3D '11. ACM, San Francisco, California, 2011, pages 103–110.

[139] William B. Thompson, Peter Willemsen, Amy A. Gooch, Sarah H. Creem-Regehr, Jack M. Loomis, and Andrew C. Beall. Does the quality of the computer graphics matter when judging distances in visually immersive environments. *Presence: Teleoperators and Virtual Environments*, 13(5):560–571, October 2004.

[140] Tamás Umenhoffer, Gustavo Patow, and László Szirmay-Kalos. Robust multiple specular reflections and refractions. In, *GPU Gems 3*, pages 387–407. Addison-Wesley, 2008.

[141] Yuki Uranishi, Akimichi Ihara, Hiroshi Sasaki, Yoshitsugu Manabe, and Kunihiro Chihara. Real-time representation of inter-reflection for cubic marker. In *Proceedings of the 2009 8th IEEE International Symposium on Mixed and Augmented Reality*. In ISMAR '09. IEEE Computer Society, Washington, DC, USA, 2009, pages 217–218.

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

[143] Rui Wang, Rui Wang, Kun Zhou, Minghao Pan, and Hujun Bao. An efficient GPU-based approach for interactive global illumination. In *ACM SIGGRAPH 2009 papers*. In SIGGRAPH '09. ACM, New Orleans, Louisiana, 2009, 91:1–91:8.

[144] Gregory J. Ward. Measuring and modeling anisotropic reflection. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*. In SIGGRAPH '92. ACM, New York, NY, USA, 1992, pages 265–272.

[145] T. Whelan, M. Kaess, M.F. Fallon, H. Johannsson, J.J. Leonard, and J.B. McDonald. Kintinuous: spatially extended KinectFusion. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*. Sydney, Australia, July 2012.

[146] Ren Wu, Bin Zhang, and Meichun Hsu. Clustering billions of data points using GPUs. In *Proceedings of the combined workshops on unconventional high performance computing workshop plus memory access workshop*. In UCHPC-MAW '09. ACM, Ischia, Italy, 2009, pages 1–6.

[147] Chris Wyman. Hierarchical caustic maps. In *Proceedings of the 2008 symposium on Interactive 3D graphics and games*. In I3D '08. ACM, Redwood City, California, 2008, pages 163–171.

[148] Chris Wyman. 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*. In GRAPHITE '05. ACM, Dunedin, New Zealand, 2005, pages 205–211.

[149] Chris Wyman and Scott Davis. Interactive image-space techniques for approximating caustics. In *Proceedings of the 2006 symposium on Interactive 3d graphics and games*. In I3D '06. ACM, Redwood City, California, 2006, pages 153–160.

[150] Chris Wyman and Greg Nichols. Adaptive caustic maps using deferred shading. *Computer Graphics Forum*, 28(2):309–318, May 7, 2009.

[151] Xuezhong Xiao and Lizhuang Ma. Color transfer in correlated color space. In *Proceedings of the 2006 ACM international conference on virtual reality continuum and its applications*. In VRCIA '06. ACM, Hong Kong, China, 2006, pages 305–309.

[152] S. Xu and A. M. Wallace. Recovering surface reflectance and multiple light locations and intensities from image data. *Pattern Recognition Letters*, 29:1639–1647, 11, August 2008.

[153] Yizhou Yu, Paul Debevec, Jitendra Malik, and Tim Hawkins. Inverse global illumination: recovering reflectance models of real scenes from photographs. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. In SIGGRAPH '99. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1999, pages 215–224.

[154] Mario Zechner and Michael Granitzer. Accelerating k-means on the graphics processor via cuda. In *Proceedings of the 2009 First International Conference on Intensive Applications and Services*. In INTENSIVE '09. IEEE Computer Society, Washington, DC, USA, 2009, pages 7–15.

[155] Ming Zeng, Fukai Zhao, Jiaxiang Zheng, and Xinguo Liu. A memory-efficient KinectFusion using octree. In *Proceedings of the First international conference on Computational Visual Media*. In CVM'12. Springer-Verlag, Beijing, China, 2012, pages 234–241.

[156] Zuoyong Zheng, Lizhuang Ma, Zhong Li, and Zhihua Chen. An extended photometric stereo algorithm for recovering specular object shape and its reflectance properties. *Computer Science and Information Systems*, 7(1):1–12, 2010.

[157] Zuoyong Zheng, Lizhuang Ma, Zhong Li, and Zhihua Chen. Reconstruction of shape and reflectance properties based on visual hull. In *Proceedings of the 2009 Computer Graphics International Conference*. In CGI '09. ACM, Victoria, British Columbia, Canada, 2009, pages 29–38.

# Curriculum Vitae

## Personal


| | |
|---|---|
| **Name Surname** | Martin Knecht |
| **Gender** | male |
| **Day of Birth** | September 15th, 1982 in Feldkirch, Austria |
| **Marital Status** | single |
| **Academic Degree** | Dipl.-Ing. Mag.rer.soc.oec. (equ. M.Sc., M.Sc.) |
| **Languages** | German, English |
| **Phone** | +43 650 9387471 |
| **Email** | martin.knecht@aon.at |


## Education

| | |
|---|---|
| **since 2009/08** | Ph.D. student in Computer Science,<br>Vienna University of Technology, Austria<br>Advisors: Associate Prof. Michael Wimmer and Dr. Christoph Traxler |
| **2007/10 - 2009/09** | Master study at the University of Vienna, Austria<br>Major: Computer Science Management<br>Thesis: Real-time Global Illumination Using Temporal Coherence<br>Date of graduation: 23.09.2009 |
| **2006/10 - 2009/07** | Master study at the Vienna University of Technology, Austria<br>Major: Computer graphics and digital image processing<br>Thesis: Real-time Global Illumination Using Temporal Coherence<br>Date of graduation: 28.07.2009<br>Graduation with distinction |
| **2003/10 - 2006/09** | Bachelor study at the Vienna University of Technology, Austria<br>Major: Computer science in media<br>Date of graduation: 27.09.2006 |
| **1997/09 - 2002/07** | Secondary school „Höhere Technische Lehranstalt Rankweil“, Austria<br>Graduation with distinction |

## Professional

| | |
|---|---|
| **2010/09 - 2010/11** | Internship at Human Interface Technology Laboratory, New Zealand |
| **2009/08 - 2013/01** | Project Assistant; Institute of Computer Graphics and Algorithms, Vienna University of Technology, Austria |
| **2008/10 - 2009/07** | Tutor at Vienna University of Technology, Austria |
| **2008/07 - 2008/09** | Internship at Tampere University of Technology, Finland |
| **2007/10 - 2008/07** | Tutor at Vienna University of Technology, Austria |
| **2007/07 - 2007/08** | Internship at witsch visuals |
| **2003/01 - 2007/06** | Several web development projects for xoo design |
| **2002/10 - 2003/09** | Community Service |
| **2000/07 - 2000/08** | Internship at Telekom Austria |
| **1999/07 - 1999/08** | Internship at Hirschmann Electronic |

## Reviewing

| | |
|---|---|
| **2013** | CESCG, ISMAR, Pacific Graphics |
| **2012** | ISMAR, Pacific Graphics |
| **2011** | CESCG, CGI, TVCG |

## Talks

| | |
|---|---|
| **2013/01** | Summary of the contributions from the RESHADE project<br>CG-Konversatorium, Vienna University of Technology, Austria |
| **2010/06** | Reciprocal Shading for Mixed Reality<br>Czech Technical University in Prague |
| **2010/03** | Current state of the RESHADE project<br>CG-Konversatorium, Vienna University of Technology, Austria |

## Supervised Students

| | |
|---|---|
| **2012/03 - 2013/09** | Martina Rasch, Bachelor Thesis<br>HDR Image Acquisition for Augmented Reality |
| **2012/03 - 2012/10** | Stefan Spelitz, Bachelor Thesis<br>Color Distribution Transfer For Mixed Reality Applications |
| **2012/03 - 2012/11** | Adam Celarek, Bachelor Thesis<br>Merging Ray Tracing and Rasterization in Mixed Reality |
| **2011/10 - 2013/02** | Klemens Jahrmann, Bachelor Thesis<br>3D Reconstruction with the Kinect-Camera |
| **2011/10 - 2013/02** | Christoph Winklhofer, Master Thesis<br>Reflections, Refractions and Caustics in a Mixed Reality Environment |
| **2011/01 - 2011/10** | Georg Tanzmeister, Master Thesis<br>Interactive 3D Reconstruction and BRDF Estimation for Mixed Reality Environments |

## Awards

| | |
|---|---|
| **Best Paper Award** | Differential Instant Radiosity for Mixed Reality<br>ISMAR Conference, 2010, Seoul |
| **Emerging Technology Student Award** | Differential Instant Radiosity for Mixed Reality and Adaptive Camera-Based Color Mapping For Mixed-Reality Applications - Demo, RTT Excite, 2012, Munich |

## Peer-Reviewed Publications


[1] Martin Knecht, Andreas Dünser, Christoph Traxler, Michael Wimmer, and Raphael Grasset. A framework for perceptual studies in photorealistic augmented reality. In *Proceedings of the 3rd IEEE VR 2011 Workshop on Perceptual Illusions in Virtual Environments*. Singapore, March 2011, pages 27–32.

[2] Martin Knecht, Georg Tanzmeister, Christoph Traxler, and Michael Wimmer. Interactive BRDF estimation for mixed-reality applications. *Journal of WSCG*, 20(1):47–56, June 2012.

[3] Martin Knecht, Christoph Traxler, Oliver Mattausch, Werner Purgathofer, and Michael Wimmer. Differential instant radiosity for mixed reality. In *Proceedings of the 9th IEEE International Symposium on Mixed and Augmented Reality*. In ISMAR '10. IEEE Computer Society, Seoul, South Korea, 2010, pages 99–108.

[4] Martin Knecht, Christoph Traxler, Oliver Mattausch, and Michael Wimmer. Reciprocal shading for mixed reality. *Computers & Graphics*, 36(7):846–856, November 2012.

[5] Martin Knecht, Christoph Traxler, Werner Purgathofer, and Michael Wimmer. Adaptive camera-based color mapping for mixed-reality applications. In *Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality*. In ISMAR '11. IEEE Computer Society, Washington, DC, USA, 2011, pages 165–168.

[6] Martin Knecht, Christoph Traxler, Christoph Winklhofer, and Michael Wimmer. Reflective and refractive objects for mixed reality. *IEEE Transactions on Visualization and Computer Graphics*, 19(4):576–582, April 2013.

[7] Patrick Kühtreiber, Martin Knecht, and Christoph Traxler. BRDF approximation and estimation for augmented reality. In *15th International Conference on System Theory, Control and Computing*. Gheorghe Asachi Technical University of Iasi, Faculty of Automatic Control and Computer Engineering. Sinaia, Romania, October 2011, pages 318–324.