FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Interactive 3D Reconstruction and BRDF Estimation for Mixed Reality Environments

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Computergraphik/Digitale Bildverarbeitung

eingereicht von

Georg Tanzmeister
Matrikelnummer 0625380

an der
Fakultät für Informatik der Technischen Universität Wien

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

Wien, 24.10.2011

(Unterschrift Verfasser) (Unterschrift Betreuung)

Erklärung zur Verfassung der Arbeit

Georg Tanzmeister
Roseggergasse 15/21, 1160 Wien

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 Verfasser)

Acknowledgements

This page is dedicated to all the people who helped and encouraged me during my work. First of all I want to thank the Vienna University of Technology for the generous financial support to buy large parts of the equipment that I needed for my research. Especially I want to thank the Institute of Computer Graphics for their helpfullness in both academic and organisational affairs. In particular I want to thank my supervisors Martin Knecht, Christoph Traxler and Michael Wimmer for their support.

I also want to thank my collegues and friends for interesting discussions and for helping me to hold my motivation throughout the whole work.

A very big thank you goes to my family. Without the massive support from my parents I would not have been able to do a work to this extent and in this amount of time. Even more, as with this work I finish my studies, I want to use this opportunity to thank them for the invaluable support in all the 5 years of my study without which I would not have been able to master it the way I did.

Thank you.

Abstract

In mixed reality environments virtual content typically looks very artificial. One reason for that is because there is no consistent shading between the virtual and the real objects. Two examples are shadows and indirect illumination between the artificial and the real scene elements, which require to have information about the real world's geometry and its materials respectively.

In mixed reality interaction with real objects is a key feature. Integrating consistent shading in such a system means that the system at all times needs an up-to-date model of the scene's geometry, its lighting and its material characteristics. The information is usually obtained as a manual pre-processing step, which is a tedious, time-consuming task and has to be re-done whenever a scene element that is not tracked changes. This poses strong limits to the wide-spread use of such a technique and one would like to have it done automatically.

However, the automatic estimation of material characteristics of real objects using color images has always been an offline task in the literature having processing times from around 30 minutes up to several hours. In this work an interactive BRDF estimation technique is proposed, which uses the parallel power of current GPUs speeding up the running time to under half a second. One reason for the speed-up was a novel GPU K-Means implementation using MIP maps to calculate the new cluster centers on the GPU, which is often done on the CPU. The 3D geometry is also reconstructed in our technique since it is needed for indirect illumination and occlusion. We use the Microsoft Kinect sensor to acquire both, the geometry and the color images and capture the lighting environment using a fish-eye lens camera. With the algorithm presented in this thesis we have shown that real-time results are possible opening up its use in mixed reality systems in order to improve the appearance of virtual content.

Kurzfassung

In vielen Mixed Reality Applikationen wirken die virtuellen Objekte als wären sie nicht in die Szene integriert und erscheinen daher sehr künstlich. Das liegt unter anderem daran, dass sie anders schattiert werden als die Realen und Phänomene wie indirekte Beleuchtung, Schatten und Verdeckungen nicht konsistent sind.

Um solche Effekte zu berücksichtigen benötigt man ein geometrisches Modell der realen Szene sowie Informationen über deren Materialeigenschaften und deren Beleuchtung. Da sich in Mixed Reality Anwendungen all diese Komponenten dynamisch verändern können - man denke zum Beispiel an das Verschieben eines realen Gegenstands - genügt es nicht, diese Daten als Vorverarbeitungsschritt zu modellieren.

Die automatische Schätzung von Materialeigenschaften von realen Objekten anhand von Farbbildern ist allerdings mit sehr viel Rechenaufwand verbunden und Laufzeiten von solchen Algorithmen bewegen sich zwischen 20 Minuten und mehreren Stunden. In dieser Diplomarbeit wird ein Algorithmus zur interaktiven 3D Rekonstruktion und BRDF Schätzung vorgestellt, dessen Laufzeit weniger als eine halbe Sekunde beträgt. Diese immense Leistungssteigerung wurde unter anderem durch eine neuartige GPU Implementierung des K-Means Algorithmus erreicht, bei der, neben den Distanzberechnungen, auch die Schwerpunktsberechnungen mit Hilfe von MIP Maps auf die GPU verlagert wurden. Wir verwenden die Microsoft Kinect um Farb- und Tiefeninformationen über die Szene zu bekommen sowie eine Fischaugekamera um die Beleuchtung einzufangen und zeigen, dass Schätzungen von Materialeigenschaften anhand von Farbbildern in Echtzeit machbar sind, wodurch die Verwendung in Mixed Reality Systemen, zur Verbesserung der visuellen Darstellung von virtuellen Objekten, ermöglicht wird.

Contents

1 Introduction **1**

1.1 Motivation 3

1.2 Contribution 3

1.3 Structure of the Work 4

2 Related Work **5**

2.1 Differential Instant Radiosity for Mixed Reality 5

2.2 Bidirectional Reflectance Distribution Function 6

2.2.1 The Phong and Blinn-Phong Reflectance Model 8

2.2.2 The Ward Reflectance Model 9

2.2.3 Comparison between Phong and the Isotropic Ward 9

2.3 Image-based BRDF Estimation 10

2.3.1 Overview of BRDF Estimation Techniques 11

2.3.2 BRDF Estimation by Zheng et al. 14

2.3.3 Runtime Comparison 16

2.4 Highlight Removal 17

2.4.1 Overview of Highlight Removal Techniques 17

2.4.2 Highlight Removal by Ortiz and Torres 19

2.5 Normal Estimation 24

2.6 K-Means 25

2.6.1 Further Considerations on Standard K-Means 25
2.6.2 GPU-based K-Means 26

3 Interactive BRDF Estimation **29**

3.1 Overview 29
3.2 Scope of Zheng et al. and our Contribution 31
3.3 Test Scenario 33
3.4 Color and Geometry Acquisition Using the Kinect Sensor 33
3.4.1 The Data 34
3.4.2 From Depth Values to Point Clouds 35
3.5 Normal Estimation 37
3.6 Light Estimation 39
3.7 Highlight Removal 40
3.7.1 Comparison to the Original Version of Ortiz and Torres 42
3.8 Inverse Diffuse Shading 44
3.9 Clustering 45
3.9.1 GPU-based K-Means 46
3.9.2 Visualization 49
3.9.3 Clustering Results 49
3.10 Specular Reflectance Estimation 50
3.11 Diffuse Reflectance Estimation 52
3.12 Limitations 54

4 Implementation **57**

4.1 Hardware Configuration 57
4.2 Software Configuration 58
4.2.1 How GPU Implementations were done 58
4.2.2 The Kinect Sensor in a 64-bit Windows Environment 58

4.2.3 Normal Estimation using PCL 59
4.2.4 Specularity Estimation using DotNumerics 60
4.3 K-Means 62
4.3.1 Results and Comparison with OpenCV 62
4.3.2 Structure of the Render Targets in the GPU Implementation 63
4.3.3 Final Thoughts 63

5 Results **65**
5.1 Error Measurements 65
5.2 Quality Comparison 66
5.3 Scenario 1 67
5.4 Scenario 2 67
5.5 Performance 68

6 Conclusion **73**
6.1 Summary and Comparison with Related Work 73
6.2 Future Work 74

List of Figures **77**

List of Tables **79**

Bibliography **81**

CHAPTER 1

Introduction

In mixed reality systems, the real world gets *mixed* with virtual computer generated content. Very simplistic examples of such systems can be found in the television news, where fonts are rendered besides the TV reporter to emphasize the news headlines, or in videos of soccer games, where additional information like the score or the time is added. While in these simple examples the *augmentations* can be done without any information about the real-world scene simply by putting the font over the 2D image, in more sophisticated systems the virtual content should be registered, i.e. there should be a 3D relationship between the virtual and the real content. This leads to the first problem in mixed reality systems: *Where do we put the virtual content such that it seems to be an integral part of the real world scene?*

Without any knowledge about the real scene it is not possible to register the virtual content. And even if we have geometry information it still might be hard to decide where the object should actually be positioned. The problem of the unknown geometry and the question regarding where to put the object is commonly dealt with by using markers and assuming a flat surrounding. The virtual object is usually simply drawn *over* the image on top of the marker. Naturally, mutual occlusion in systems which only rely on markers cannot be handled correctly.

In Figure 1.1a we can see a conceptual image of how a mixed reality application may look like if only geometry information is used to render the virtual content. It should be noted here that in this example, apart from the position where the virtual object ought to be placed, also geometry information about the scene would need to be known, since we can observe occlusion. However, this is actually rare in typical mixed reality systems and the image one would most likely get would be without incorporating occlusion.

It is immediately noticeable to an untrained observer that the teapot in Figure 1.1a is computer-

(a) Virtual teapot (b) Real teapot

Figure 1.1: Left: A virtual object, the teapot, rendered into a real scene. Right: The outcome we would expect in the real world.

generated, although it is geometrically integrated into the scene. What we actually would expect is an image that looks like the one in Figure 1.1b. The major difference between the two images is that the artificial content one time is and one time is not also photometrically integrated, meaning there is a common shading and a common illumination of the real and the virtual objects. This leads to the second question: *How do we shade the virtual content together with its real surroundings such that it looks like it were real and part of the real scene?*

To clarify that it does not suffice to only consider the shading of the virtual object and to neglect the parts of the real scene one can think about shadows, which may be cast from virtual to real content or vice-versa. Additionally mutual indirect illumination, which is often visible as color bleeding, must also be taken into account.

This simulation of the mutual influence between real and virtual objects in order to get perceptually plausible consistent shading in mixed reality systems is the aim of *Reciprocal Shading for Mixed Reality* (RESHADE) [7], a research project sponsored by the FFG. By definition of Azuma [2], one criteria of augmented reality and hence of mixed reality is that interaction in real time is possible. Therefore we cannot rely on any offline methods, but we need to achieve interactive frame rates.

In the course of the project a novel rendering method for mixed reality systems was developed to accomplish this challenging task. We will give further details about the renderer in Section 2. With this novel rendering method we presented an answer to the second question, the one of how the virtual content together with the surroundings ought to be shaded for a consistent look.

Before we go on to the motivation of this thesis, we also want to say a few words about the first question, regarding where to put the augmentations. As mentioned before, many different tracking technologies have been developed for this problem. Some of them are quite stable and widely used and others, especially marker-less technologies, are still subject of current research. This is however not the topic of this work, although it is clear that a virtual object would never look real if it were not registered in the scene.

1.1 Motivation

One great accomplishment of RESHADE was Differential Instant Radiosity from Knecht et al. [28]. Without going into too much detail, it is clear that in order to include effects due to the real scenery, we need to *have* information about the real scenery. So to illuminate the virtual objects similar to the way the real objects are illuminated, we need to capture the real lighting. And to incorporate mutual effects, like indirect illumination, shadows and occlusion we need to know the geometry and the material characteristics of the real environment.

In our rendering system illumination acquisition is done in real-time by using a fish-eye lens camera. Geometry and materials are however manually modeled as a pre-processing step. This is suboptimal for at least two reasons: Firstly, modeling is time-consuming and needs training and thus poses a limit for a wide-spread use of such a mixed reality system. And secondly, every time an element of the scene which is not tracked changes, the model needs to be manually updated.

Especially in mixed reality environments, where interaction with real world objects is what makes it so fascinating and what opens up new possibilities in the way how we interact with the system, the real world is likely to change very often. Hence, even an automatic offline system would not be entirely satisfying, because then again we need to divide the scene into *movable* and *unmovable* parts and track all the movable objects. Hence, what we really want and what is the topic of this thesis, is a fully automatic system that captures geometry and material characteristics of a whole scene and all that in real time.

1.2 Contribution

The contribution of this work is a theoretical as well as practical design of a system that meets the ambitious goal of simultaneously estimating the geometry and the reflectance properties of multiple objects in interactive frame-rates. Current image-based reflectance estimation techniques all work offline and have processing times from around 20 minutes up to several hours. In this work we present a system that does the estimation based on one color image and a depth

map, which are captured from two online streams. Since we are working on a per frame basis and achieve interactive frame-rates, changes of the real scene do not pose any problem and we do not need to track real world objects that ought to be moved for interaction.

To summarize: Our contribution consists of a technique for estimating the 3D geometry and the material characteristics of multiple objects, which is

- interactive,
- can handle dynamic scenes,
- uses relatively cheap hardware and
- does not need any pre-processing.

1.3 Structure of the Work

The rest of this thesis is structured as follows. In Chapter 2 we present work related to material estimation. We quickly review BRDFs and show the two most important illumination models for our work. Apart from an informative state-of-the-art review of current material estimation techniques, we also summarize our novel rendering method and review work related to highlight removal. In Chapter 3 we then present our contribution, which consist of several different individual parts. As each part was tested on its own, the individual results are also given. In Chapter 4 implementation related information is provided. Finally, in Chapter 5 we demonstrate our results and with Chapter 6 we conclude.

CHAPTER 2

Related Work

This chapter presents an overview of work related to BRDF estimation. Section 2.1 reviews a novel rendering method for mixed reality, which was developed as part of the RESHADE project mentioned earlier. Section 2.2 explains what a BRDF is and presents two commonly used illumination models. Additionally, the question of how the color of a pixel on the image plane is calculated and what part the BRDF plays in the rendering equation is also referred. Section 2.3 then gives a comprehensive overview of existing BRDF estimation approaches with a detailed focus on the technique which is most similar to ours. Then, Section 2.4 presents several highlight removal techniques. Highlight removal is important when trying to cluster similar materials together and the most applicable approach for our scenario is examined in more detail. Finally, Section 2.5 and 2.6 present information about normal estimation and K-Means clustering, which will be useful in the following chapters.

2.1 Differential Instant Radiosity for Mixed Reality

Knecht et al. [28] developed a novel rendering method for mixed reality systems with the aim of seamlessly blend virtual content into a real scene. In many mixed reality systems the virtual content looks very artificial, see for example Figure 1.1 from the previous chapter, because there is no common illumination between the real and the virtual objects and additionally indirect illumination is often completely ignored. Differential Instant Radiosity combines Instant Radiosity, a global illumination algorithm, and Differential Rendering in a way so that shading is performed only once and real-time renderings are possible.

Usually, when trying to render a mixed reality scene using global illumination one has to acquire the geometry and the materials of the real scene and render them together with the

virtual objects. Since these measurements possibly introduce errors, which in turn lead to errors in the image of the real objects, the idea of Differential Rendering [10, 13] is to calculate only the difference between the image of the real scene and the complete rendering using both virtual and real objects, and to add this difference to the captured camera image.

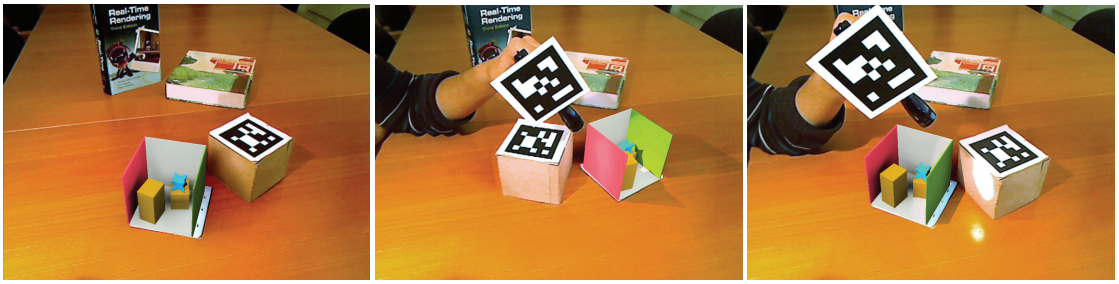
In Instant Radiosity by Keller [26] virtual point lights (VPLs) are placed in the scene to approximate a global illumination solution. Every time a ray cast from a light source into the scene intersects with an object, a VPL is created to store the amount of indirect illumination. After a sufficient amount of VPLs has been placed, the scene is rendered as if it consisted only of direct illumination provided by the VPLs. To take visibility between the objects and the light sources into account, shadow maps are used. Since calculating hundreds of shadow maps per frame is very expensive, Knecht et al. [28] use imperfect shadow maps [50], which are low-resolution shadow maps rendered by using a point representation of the scene. Imperfect shadow maps bring a significant speed-up in contrast to traditional shadow maps at the cost of accuracy in visibility.

In order to achieve a consistent shading between the virtual and the real, three things need to be known from the real world: The geometry, the material characteristics and the illumination. The first two are assumed to be known in the paper of Knecht et al. and it is the task of this thesis to design an automatic algorithm for the acquisition. The latter one, the real illumination, is captured by a fish eye lens camera to produce an up-to-date environment map. To combine environment maps with Instant Radiosity, one can think of a virtual primary light source which only sees this map [28]. VPLs are then distributed around the hemisphere according to the brightest spots in the environment map. The fish eye lens camera and the see-through camera are registered in a common coordinate system by using Studierstube Tracker [16].

Figure 2.1 demonstrates some of the results from Knecht et al. [28]. We can observe multiple mutual effects between virtual and real content rendered at real-time frame-rates. In Figure 2.1a we observe that a virtual object throws a shadow on a real one. In Figure 2.1b we can see indirect illumination from a virtual onto a real object and Figure 2.1c shows how indirect illumination is considered in the opposite direction - from a real onto a virtual object. A video of the results can be found at [8].

2.2 Bidirectional Reflectance Distribution Function

A *Bidirectional Reflectance Distribution Function* (BRDF) [39] is a function that describes reflectance characteristics of a material. In the real world when light hits an object, there are many different ways how the light interacts with this object depending on its material characteristics and its shape. Some objects may absorb part of the spectrum of the incoming light and scatter the



(a) (b) (c)

Figure 2.1: (a) Virtual object shadows a real one. (b) A pocket lamp points towards a virtual Cornell box causing red color bleeding through indirect illumination towards the desk and the cardboard box. (c) A pocket lamp illuminates the real cardboard box. Indirect illumination via VPLs causes the green wall of the Cornell box to appear brighter. Figures taken from [28]

other part uniformly in all directions, which yield diffuse colored objects, or some may reflect almost all the light in only one direction, which yield mirror-like objects. So to describe all the different opaque surfaces, the concept of BRDF's was introduced, which are multi-dimensional functions that define how the light is reflected. More specifically, a BRDF describes how much of the light coming from direction ω_i is reflected in direction ω_o, see Figure 2.2. It is at least a 4-dimensional function due to the two vectors ω_i and ω_o, which can be represented by 2 angles each over a hemisphere. The number of parameters can easily climb up to higher dimensions, since one might want to model non-uniform surfaces, where the outcome also depends on the surface point $\mathbf{p}$ (Spatially Varying BRDF) or even just color, where the amount of reflection is wavelength-dependent.

BRDF's are an essential part of the rendering equation and can either be stored in form of a table, as a result from a measurement or a simulation, or approximated by an analytic function. Measurements can be made with the use of a gonioreflectometer, which is a special device that usually creates a huge amount of data describing the scanned material. Analytical functions, on the other hand, are way more common in computer graphics, since apart from the need of special equipment, scanning materials is a time-consuming process.

Many different analytical models can be found in the literature ranging from very simple and hence fast empirical models, like the over 250 year-old Lambertian model, to complex and sometimes even physically-based ones like the Cook-Torrance [9], the He [17] and the Oren-Nayar [43] model. Below we will present two commonly used models in computer graphics and computer vision in more detail.

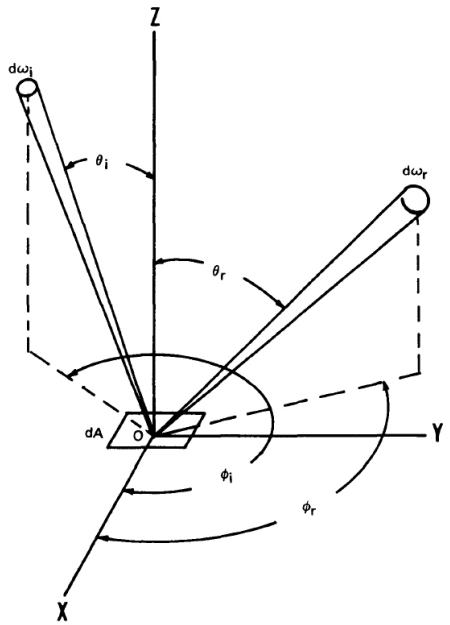


Figure 2.2: The geometry of the incident ray ω_i, defined by the angles (θ_i, ϕ_i), and the reflected ray ω_r,defined by the angles (θ_r, ϕ_r). The $\mathbf{Z}$-axis corresponds with the normal at the surface point. Figure courtesy of Nicodemus et al. [39].

2.2.1 The Phong and Blinn-Phong Reflectance Model

Probably the most famous illumination model, which is still used very often in computer graphics, is the Phong reflectance model [48]. It is a rather simple empirical model without any physical foundation. The BRDF, denoted by fr, is given by the following equation:

$$fr(\mathbf{L}, \mathbf{V}) = \mathbf{k_d} + \mathbf{k_s}\frac{(\mathbf{V} \cdot \mathbf{R})^{n_s}}{(\mathbf{N} \cdot \mathbf{L})}, \tag{2.1}$$

where $\mathbf{V}$ is the normalized vector from the surface point to the view point, $\mathbf{N}$ is the normal at that point, $\mathbf{L}$ is the normalized vector to the light source and $\mathbf{R}$ is the reflection vector based on the the normal and the light vector. The parameters $\mathbf{k_d}$ and $\mathbf{k_s}$ are the diffuse and the specular reflection coefficients, which are used to describe a certain material and the parameter n_s is used to control the size of the specular highlight.

We use the following simplified form of the rendering equation for local illumination where the intensity $\mathbf{I}$ of a pixel can be calculated as follows [18]:

$$\mathbf{I} = \mathbf{k_a}\mathbf{I_a} + \sum_{l=1}^{n} \mathbf{I}_l(\mathbf{N} \cdot \mathbf{L}_l) fr(\mathbf{L}_l, \mathbf{V}) \tag{2.2}$$

$$\mathbf{I} = \mathbf{k_a}\mathbf{I_a} + \sum_{l=1}^{n} \mathbf{I}_l[\mathbf{k_d}(\mathbf{N} \cdot \mathbf{L}_l) + \mathbf{k_s}(\mathbf{V} \cdot \mathbf{R}_l)^{n_s}], \tag{2.3}$$

where $\mathbf{I}_l$ is the intensity or the color of the l-th light source and $\mathbf{k_a}$ together with $\mathbf{I_a}$ describe the ambient illumination.

A very common variation to the Phong model was proposed by Blinn in 1977 [3] and is known under the name Blinn-Phong reflectance model. It is similar to the Phong model, except that instead of the reflection vector, the easier to compute halfway vector $\mathbf{H}$ is used to speed up the calculation while preserving the same characteristics:

$$\mathbf{I} = \mathbf{k_a}\mathbf{I_a} + \sum_{l=1}^{n} \mathbf{I}_l[\mathbf{k_d}(\mathbf{N} \cdot \mathbf{L}_l) + \mathbf{k_s}(\mathbf{N} \cdot \mathbf{H}_l)^{n_s}] \tag{2.4}$$

$$\mathbf{H} = \frac{\mathbf{L} + \mathbf{V}}{|\mathbf{L} + \mathbf{V}|} \tag{2.5}$$

2.2.2 The Ward Reflectance Model

Ward [64] presented a physically-plausible model, which is easy and fast to evaluate in comparison to other models like the He [17] model. With the Ward model it is possible to represent a wide range of surface materials, for example plastics and metals [64]. Ward presented two different variants: One for isotropic materials yielding a simpler formula and one for anisotropic materials. We only consider the isotropic model here as we want to keep things simple and fast. The amount of light that is reflected at a surface point is given by:

$$f_r = \frac{\mathbf{k_d}}{\pi} + \frac{\mathbf{k_s}}{\sqrt{\cos(\mathbf{N} \cdot \mathbf{L}) \cos(\mathbf{N} \cdot \mathbf{V})}} \frac{\exp[-(\tan^2(\mathbf{N} \cdot \mathbf{H}))\alpha^2]}{4\pi\alpha^2} \tag{2.6}$$

The parameters here are similar to the ones described in the Phong model and α is a control parameter that describes the surface roughness.

2.2.3 Comparison between Phong and the Isotropic Ward

One simplification we make in order to increase the performance is that we use the Phong model as opposed to the isotropic Ward model used in many other BRDF estimation algorithms. Hence, we also want to compare these two to see the benefits and the drawbacks.

The Ward model is physically plausible. It is well normalized, has meaningful parameters and is energy-conserving. On the other hand, the physical meaning and the correct range of the parameters of the Phong model are not clear, except from $\mathbf{k_d}$ which represents the diffuse

albedo. So it is hard to set them properly. Additionally it is not normalized. Westin et al. [66] compare 4 different BRDF models including the Phong and the Ward model. They test the models on different materials, like rough metal, rough plastic, white paper, smooth metal and metallic paint. They come to the conclusion that none of the 4 tested models performs well on every surface and that the Ward and the Phong model perform similar with slight changes depending on the material. However, the Ward model is more expensive to evaluate. So since the visual difference is marginal and we do not need physical plausible values as the parameters are only used to enhance the quality in mixed reality environments, the Phong model is an almost as good but cheaper alternative and suitable for our needs.

2.3 Image-based BRDF Estimation

Bidirectional reflectance distribution function estimation by using images almost always means calculating the surface reflection parameters for a certain reflection model, e.g. the Phong model or the Ward model. The aim is to fit parameters for an underlying often rather simplistic model, instead of trying to provide a complete BRDF in form of a table as a Gonioreflectometer is providing. Usually, the number of input images is anyway very low, so a complete BRDF would not be possible.

The process of BRDF estimation is sometimes synonymously called inverse rendering. Looking at Equation 2.2, which describes how the pixel color $\mathbf{I}$ can be calculated under local illumination, we can see that the parameters involved can be categorized into three classes, depending on whether they describe:

- the geometry of the object or scene, or
- the lighting environment, or
- the material characteristics of the object or scene.

The parameters belonging to the geometry are the surface normal $\mathbf{N}$, the viewing direction $\mathbf{V}$ and the surface point $\mathbf{X}$, which is implicitly used when calculating the light vector $\mathbf{L}$. The parameters determining the lighting environment are the light positions, which again are implicitly used to calculate the light vectors $\mathbf{L}$, and the light colors $\mathbf{I}_l$ and $\mathbf{I_a}$. The third class, the class of material parameters, cannot be generalized and depend on the BRDF model used. If we choose for example Phong, which yields Equation 2.3, these are $\mathbf{k_a}$, $\mathbf{k_d}$, $\mathbf{k_s}$ and n_s.

Strictly speaking, BRDF estimation only focuses on the estimation of the material characteristics and some of the BRDF estimation techniques we will present in the next section only focus

on this alone, assuming the geometry and lighting information to be known, e.g. by manually generating a model of a scene. However, it is usually not the case that one has one or several images of a real scene together with a representation of the geometry and the lighting environment. In mixed reality environments the scenes are typically not static since interaction is one of the key points by definition of Azuma [2]. Thus, modeling the scene as a pre-processing step is not desirable, since the model would soon be wrong or the interaction possibilities would need to be reduced dramatically.

Other material estimation techniques aim at estimating *all* the parameters including the geometry and the light environment and thus offer a complete framework. For our scenario we also need an approach of this sort, since we do not want any manual, time-consuming pre-processing steps.

In Section 2.3.1 we will give an overview of different BRDF estimation techniques in chronological order. Then, in Section 2.3.2 we will present the method closest to ours and in Section 2.3.3 we will show the running times of some of the algorithms discussed to summarize the state of the art.

2.3.1 Overview of BRDF Estimation Techniques

Image-based BRDF estimation techniques, as mentioned before, almost always try to fit parameters for a certain reflection model to produce a look that comes as close as possible to the real photograph. The specification of such reflection models was therefore of great significance and probably the starting point for the field of image-based BRDF estimation. Many techniques have then been developed to estimate the reflectance characteristics of an isolated object under specific illumination conditions [73], for example Sato et al. [52] and Lu and Little [33]. Those methods cannot be applied directly to whole scenes or even to non uniform objects, since then material characteristics are not constant anymore and their approaches will fail. Therefore, we will only concentrate on newer approaches. In Kühtreiber [29] you can find a survey of some of those newer BRDF estimation techniques that also take the spatially-varying nature of scenes into account.

Yu et al. [73] developed a technique to recover reflectance properties of real scenes from a sparse set of *High Dynamic Range* (HDR) photographs. They define the problem as inverse global illumination, where they estimate the parameters for Ward's reflectance model based on the known geometry, the known illumination and a couple of radiance maps. The authors also propose to partition the scene into groups of similar materials, where the specular characteristics stay constant but the diffuse texture can vary. This way they get more samples for the estimation of the specular component without restricting themselves to single diffuse-colored objects.

Those groups of surfaces are specified manually during the modeling stage.

Although the algorithm of Yu et al. is very interesting from a theoretical point of view and many newer techniques use some of their concepts, especially the idea of grouping together surfaces sharing the same specular reflectance, it is not applicable for our scenario since a lot of manual work has to be done, like creating a model of the geometry and the light source, specifying groups of surfaces and shooting pictures from many different positions. Additionally, their iterative inverse global illumination procedure involves a lot of processing and took six hours to complete on a 300MHz machine.

Then, Boivin and Gagalowicz [4] presented an iterative approach, which is also aimed at estimating the parameters for the Ward model. Similar to Yu et al. [73], they manually create a 3D model of the scene including the light source and group surfaces that share the same specular reflection characteristics together, but instead of using several HDR-images, they work on a single LDR-image. Their iterative procedure consists of seven stages called: *perfectly diffuse*, *perfectly specular*, *non-perfectly specular*, *diffuse and non-perfectly specular*, *isotropic (rough)*, *anisotropic (rough)* and *textured*. These stages, as the names suggest, correspond to certain assumptions and they are processed in the order given, from simple and fast to complex with a high processing time. They start with the assumption that a group is perfectly diffuse, estimate the diffuse reflection coefficient and render it with a global-illumination software. Then, the difference between the synthetic image and the real one is computed and the diffuse reflection coefficient is adjusted according to this difference. If after 4 iterations the difference still lies above a certain threshold, the algorithm moves on to the next step.

On the one hand their iterative procedure makes a good job in trying to minimize the complexity of the materials, but on the other hand in real scenes many surfaces exhibit texture characteristics, which requires their algorithm to compute a total of 4 *iterations* $* 7$ *stages* $= 28$ *global illumination renderings* plus the calculations to update the reflection parameters and the image comparisons. Hence, the algorithm will run slowly on almost all objects. Running times can be found in Section 2.3.3. Similar to the approach of Yu et al., a lot of manually pre-processing has to be done, like modeling the scene, the lights and grouping surfaces that exhibit similar specular characteristics together. Another observation we made is that in the last stage, where a surface is assumed to be textured, the specular component is assumed to be zero. This is surprising to us, since we expect many surfaces to exhibit varying diffuse parameters but a common similar specular parameter. A wooden table or a stone floor are just two examples of such surfaces.

Later, Wu et al. [69] demonstrated how to recover materials under complex illumination conditions from a single HDR image. Until then, almost all the techniques developed assume rather simple illumination conditions, i.e. point and area light sources. Since it is not possible to recover lighting, texture and reflectance properties simultaneously due to the lighting-texture

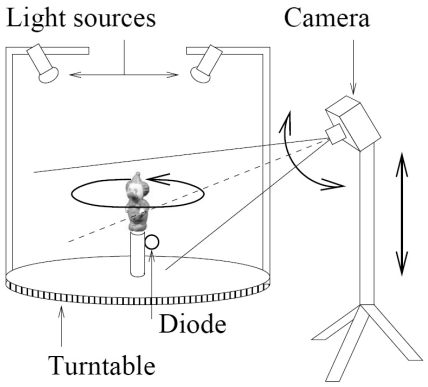


Figure 2.3: Acquisition system of Mercier et al. [34]. Figure also courtesy of Mercier et al. [34].

ambiguity [49], they first estimate the lighting using one or more high dynamic range environment maps, which describe the illumination of the object. Then, under the assumption of known lighting, they recover the reflection parameters of a slightly modified version of RADIANCE's model [65] using Simulated Annealing, where the objective function is the sum of squared distances of the photograph and the rendering. Although their approach is designed for a single homogeneous object, it can also be applied on transparent and translucent objects.

Wu et al. do not mention anything about the geometry of the models although it is clear that without any geometrical information, the lighting calculations do not make any sense and additionally they also only mention the material parameters in the results. The restriction to single uniform objects and the running time of 2-3 hours on a 667MHz CPU do not fit our needs.

Mercier et al. [34] proposed a complete framework for automatically recovering an object's shape, its reflectance and the light sources from calibrated images. To acquire those calibrated images, they place the object on a turntable so that they can shoot one image every 5 degrees. The light sources are also mounted on the turntable and so the illumination of the object does not change. The camera is located on a tripod and placed at 1.5 meters away. In Figure 2.3 you can see a sketch of the setup. After the image acquisition process, the authors start by calculating the geometry of the captured object using Szeliski's *Shape From Silhouette* (SFS) approach [59]. Then, they combine the Marching Cubes algorithm together with image pixels to extract the triangular mesh together with surface normals in order to define the object's shape. Based on the voxel structure they obtained from SFS, they cluster voxels having the same hue and the same normal together to make the optimization more robust. After the clustering, the lighting environment is estimated with no other additional information than the images that were initially captured. For each class of voxels Mercier et al. find one light source which can either

be a point light or a directional light source. Since the estimation of the light source differs if the surface is diffuse or glossy, the authors present a classifier to distinguish them and give the respective equations to calculate the light information for each of the two surface characteristics. Using a gradient descent algorithm they then optimize the parameters for the light source(s) and the BRDF.

Mercier et al.'s work is very interesting since it provides a full framework for the BRDF estimation process and does not require any additional pre-processing, like manually designing a model or creating groups of similar materials. For the use in mixed reality environment, it is however not well applicable since it requires a special setup and quite some time to take the 72-images necessary to acquire the whole shape and to do the estimation. The running time of their algorithm was between 30-50 minutes on an Intel Xeon 2.4GHz processor.

One year later, Xu and Wallace [71] presented a method to recover the reflectance of objects and the parameters of multiple lights. They acquire a 3D image by using a depth sensor and a stereo color image pair by using two calibrated CCD cameras. Xu and Wallace use the Blinn-Phong BRDF model for the material recovery. They derive equations to estimate the diffuse and the specular parameters as well as the locations and intensities of up to three distinct light sources.

Similar to other approaches mentioned earlier, the main problem of their approach when viewed with respects to our needs is the isolation of the object during the capturing process. The two intensity images must capture one single object with constant specular and varying diffuse characteristics, where all the pixels in the image that do not correspond to the object must be identifiable. Another problem arises due to the fact that the lighting environment is also calculated out of the intensity images. Flat surfaces introduce many errors because of the lack of data at most normal angles [71], which obviously would pose a huge problem in indoor mixed reality environments.

2.3.2 BRDF Estimation by Zheng et al.

The work closest to ours is the one by Zheng et al. [75]. It is similar to the work done by Mercier et al. [34] and also presents a complete framework for estimating the surface shape and its reflectance properties from multiple images. The whole image capturing process and the shape calculation is done similarly to Mercier, i.e. the authors also place the object to be inspected on a special turntable but with stationary light sources, shoot a number of images while rotating the object, use SFS to acquire the shape in a voxel structure and extract the triangular mesh of the surface by using marching cubes.

After the shape acquisition, Zheng et al. proceed differently. They do not try to estimate the

light environment from the images but rather assume that it is known by measurement in order to simplify the problem and to prevent errors in the light source estimation to be propagated to the material estimation. The measurement is performed by placing a shiny steel ball on the rotation platform and by shooting two images with different exposure times. One image is used to detect the silhouette of the ball in order to fit a circle and the other is used to extract the brightest highlight point, which is done manually.

So far, all the components of the BRDF except for the material characteristics are available. To represent the materials, the authors use the isotropic Ward model, as described in Section 2.2.2. Since many calibrated images are available, all the pixels in the images which represent a single surface point are stored as one vertex to produce more samples for the estimation process. Zheng et al. then cluster all the similar materials together with the aim to estimate common specular characteristics, i.e. a common $\mathbf{k_s}$ and α for every cluster, since a per-pixel approach would not provide enough data for the optimization, even though every vertex contains multiple samples. Unfortunately, specular highlights pose a significant problem to clustering algorithms, since they introduce a significant change in the intensity function. However, what is actually needed is a clustering that includes the specular highlights to the material on which they appear. Hence, the authors use Ortiz's algorithm [45] to remove all the specularities in the image. Highlight removal will be discussed in more detail in Section 2.4.

An additional problem to the clustering process is due to the lighting, since the color of a material as seen from a specific viewpoint also depends on the orientation with respect to the light source and thus on its normal. Hence, Zheng et al. propose to cluster directly on the diffuse parameter $\mathbf{k_d}$ of each vertex, which can be derived by throwing away the specular part in the Ward model and solving the local illumination equation formulated in Equation 2.2. In this case $\mathbf{I}$ is the pixel intensity in the captured image. The clustering is done by using K-Means. The authors further simplify the problem by limiting themselves to only *one* point light source. In the next chapter, when we present our algorithm, we show how we adapted the equations to multiple light sources.

After clustering, Zheng et al. estimate the specular parameters of each class by using the Levenberg-Marquardt non-linear optimization to solve the least squares problems. The objective functions is the following:

$$F_k = \sum_{i,j} [\mathbf{I}_{i,j} - \mathbf{I}_l(\mathbf{N} \cdot \mathbf{L}) fr(\mathbf{L}, \mathbf{V})]^2 \qquad \forall \text{ vertices } i \in \text{ class } k, \tag{2.7}$$

where $\mathbf{I}_l$ is the color of the one point light source.

After $\mathbf{k_s}$ and α have been successfully estimated for each class, the diffuse parameters are re-estimated since a uniform $\mathbf{k_d}$ per cluster is not desirable. Hence, a per-vertex estimation of

$\mathbf{k_d}$ is applied to extract the diffuse texture. For each vertex i.e. surface point Equation 2.2 is solved using least-squares over all the RGB-samples of this vertex.

The algorithm of Zheng et al. does present some aspects which are beneficial for our scenario. Firstly, it is relatively simple and consists of individual parts which can be processed on its own. This facilitates the simultaneous use of the CPU and the GPU for the calculations. Secondly, it treats the difficult problem of clustering similar materials in a thorough way, by first removing specular highlights and by clustering on the diffuse parameters only, filtering out color variations due to surface normals. And thirdly, the algorithm is robust, since after a non-linear least-squares estimation of the specular part, the diffuse parameters are re-estimated in order to compensate errors due to wrong clustering or wrong geometry calculation.

However, there are still many problems which occur, when thinking about an application in our setting. We cannot rely for example on a controlled scenario, i.e. we cannot make dozens of images of every point of view of a rotating object with a stationary camera, where everything not belonging to the object is black. Hence, we cannot use SFS and need to think about a different way to acquire the geometry. Besides having no controlled scenario, this capturing process would also take too long and we do not want to have any pre-processing, but we want the algorithm to work in real-time. For similar reasons, we do not want to use a metallic sphere to capture the illumination, since it involves user interaction and has to be re-done every time the illumination changes. Apart from that, runtime is also a significant problem of Zheng et al.'s method. The authors mention that already the optimization procedure takes between 22 and 36 minutes and is thus miles away from interactive frame-rates, which is another of our ambitious goals.

2.3.3 Runtime Comparison

The aim of this section is to present the running times of some of the algorithms that have been described above. Table 2.1 shows the name of the authors together with the publication year, whether or not it provides a complete estimation, the running time and the hardware configuration. Complete estimation in this context means an estimation of the geometry, the light environment and the material characteristics.

From Table 2.1 we observe that none of the above algorithms had close to real-time ambition and that the actual running times are order of magnitude away from anything even close to real time. Especially the latter statement demonstrates the high ambition of our goals. Between 30 frames per second and one frame per hour lies a whopping difference of 5 orders of magnitude, which are to overcome somehow to make an image-based BRDF estimation algorithm applicable to mixed reality environments.

Method	Compl. Est.	Running time	Hardware
Yu et al. '99	No	6 hours	300MHz CPU
Boivin and Gagalowicz '01	No	4 hours	300MHz CPU
Wu et al. '04	No	2-3 hours	667MHz CPU
Mercier et al. '07	Yes	30-50 min	Xeon 2.4GHz CPU
Xu and Wallace '08	No	n/a	n/a
Zheng et al. '09	Yes	> 22 min	Intel T2400 CPU

Table 2.1: Overview of the running times of some of the algorithms presented in Section 2.3.1.

2.4 Highlight Removal

Highlight removal is the process of removing specular highlights in images. For most inhomogeneous materials the dichromatic reflection model is used to describe the reflection characteristics [62]. The dichromatic reflection model states that the light reflected by an object is a linear combination of a diffuse and a specular part (refer to Section 2.2). With the diffuse part it is possible to infer the object's color, whereas the specular part mostly represents the illuminant's color characteristics [57]. Such specular highlights are very common in real life scenes, because pure diffuse materials are very rare, but they often induce problems in many vision tasks, e.g. segmentation, as they lead to strong changes in the image intensity function.

In Section 2.4.1 we give an overview of existing techniques and discuss their applicability for our setting. In Section 2.4.2 we then describe the most applicable method for us in more detail.

2.4.1 Overview of Highlight Removal Techniques

The methods for highlight removal can be categorized into two different classes: Approaches that use multiple images taken under different conditions, e.g. a different light source location or a different viewing position, and approaches that work on a single image [72]. For our scenario highlight removal is designed to be a small step before clustering, hence it should be fast and additionally it should not require any interaction. Therefore, approaches like Nayar et al. [37], which uses polarizing filters, or Lin and Shum [31], which uses different light source positions, cannot be used.

Tan and Ikeuchi [62] developed a technique to separate diffuse and specular reflection components. They introduce the concept of a *Specular-Free* (SF) image, the pseudo-diffuse component of the image, which shares the same diffuse geometrical profile as the true diffuse component but with different surface colors. With the use of the dichromatic reflection model they

(a) Original image (b) SF image (c) Diffuse component

Figure 2.4: Highlight removal by Tan and Ikeuchi [62]. Figures also courtesy of Tan and Ikeuchi [62].

derive an equation where the color of a pixel can be expressed as a function of the maximum diffuse chromaticity $\tilde{c}$, which is defined as:

$$\tilde{c(\mathbf{x})} = \frac{max(I_r(\mathbf{x}), I_g(\mathbf{x}), I_b(\mathbf{x}))}{I_r(\mathbf{x}) + I_g(\mathbf{x}) + I_b(\mathbf{x})} \tag{2.8}$$

To create the SF image, they simply set the maximum diffuse chromaticity for every pixel to a constant. Although the SF image alone can be used in vision systems if the actual surface color is irrelevant, it is not possible to discriminate surfaces with the same hue but a different saturation [72]. To get a better estimation of the diffuse component, they identify pure diffuse pixels by using intensity logarithmic differentiation on the original and the SF image and then iteratively shift the more diffuse pixels to the neighboring ones. In Figure 2.4 the resulting diffuse image together with the SF and the original image is shown.

Despite the promising looking results, one limitation of the approach is that all surfaces must be chromatic, i.e. $(R \neq G \neq B)$. Hence, this technique is not well suited for our scenario, which is set in indoor environments, as surface colors from black to gray to white are very likely to be present.

Yang et al. [72] proposed a method which is very similar to the one of Tan and Ikeuchi [62], except the way of how the maximum diffuse chromaticity is estimated. They use bilateral filtering to propagate the chromaticity values from the diffuse pixels to the specular ones. Although their approach is capable of processing images at video frame-rates, it also suffers from the same

problems as the approach of Yang et al. [72].

Shen and Cai [58] presented a *Modified Specular-Free* (MSF) image, which is formed by subtracting from each pixel its minimum value of the R, G and B channel, if the minimum value is above a specific threshold. Those pixels satisfying this threshold are then regarded as specular candidates and their diffuse component is estimated. This approach, though, does not take into account that the saturation of specular highlights is low if the image is normalized with respect to the light color and thus it may encounter problems with saturated pixels where $\min\{R, G, B\}$ is just above the threshold.

A research area that is somehow related to highlight removal is image inpainting or hole filling. Regions containing highlights are either automatically detected or marked by a user and are then considered as holes, which ought to be filled. Tan et al. [61] state that a simple inpainting of the specular areas leads to surfaces that look flat since the boundary pixel values simply get propagated to fill the regions. Hence, they introduced a technique which incorporates information gathered from the highlight into the inpainting process for better shading results. In their paper the authors rely on a manual marking of the highlight and they mention that their algorithm requires a significant amount of computation, which both make it unattractive for us.

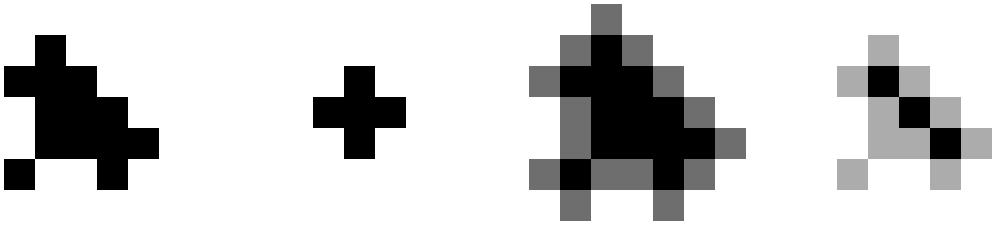
2.4.2 Highlight Removal by Ortiz and Torres

In this section we are going to present the approach most applicable to our scenario in more detail. Ortiz and Torres [45] developed a technique to remove specular highlights based on mathematical morphology. They identify specularities by transforming the color from RGB space to a color space similar to HSV and by looking for pixels with high brightness and low saturation values. Then, they use this specularity mask to inpaint the specular regions with the surrounding diffuse pixels by applying Morphological Opening by Reconstruction.

Before we look in more detail into their method, we quickly review the morphological operators used for a better understanding of how the algorithm works.

Morphological Dilation

„*Dilation* is an operation that 'grows' or 'thickens' objects in a binary image“ [15]. In a binary image there are only two classes of pixels, usually denoted as foreground and background. Given a two-dimensional structuring element S, e.g. a rectangle, we replace every foreground pixel $p = (x, y)$ of the binary image by the structuring element and thus all the pixels that are reached by S will also be defined as foreground. To be able to perform Dilation, we need to know the origin of S so that it can be placed on top of the current foreground pixel. In Figure 2.5c we can see the result of a Dilation of the image in 2.5a with the structuring element in 2.5b.



(a) Original image (b) Structuring element (c) Result after Dilation (d) Result after Erosion

Figure 2.5: Morphological Dilation and Erosion.

When dealing with gray-scale images the task becomes more difficult, since there does not exist a clear separation between foreground and background anymore. Which pixel should be replaced by the structuring element and what is its intensity? We cannot keep thinking of Dilation as an operation which is only performed at one class of pixels anymore, but we have to see it as a more general operation similar to convolution. The gray-scale extension to Dilation, denoted by $\oplus$, of the image I by the structuring element S is defined as follows [15]:

$$(I \oplus S)(x, y) = \max\{I(x - x', y - y') + S(x', y') \mid (x', y') \in D_S\}, \tag{2.9}$$

where D_S is the domain of S and $I(x, y) = -\infty, \forall (x, y) \notin D_I$.

If the structuring element S is flat, i.e. $S(x', y') = 0, \forall (x', y') \in D_S$, we can see that Equation 2.9 corresponds to a maximum filter, where each pixel p gets replaced by the maximum value of a certain region defined by S. Binary Dilation can also be seen as maximum filtering with a flat element and thus it can also be described by Equation 2.9.

Morphological Erosion

In contrast to Dilation, „*Erosion* 'shrinks' or 'thins' objects in a binary image“ [15]. Here we check for every pixel p of the foreground if the structuring element S fits completely into the object and if not we remove p. See Figure 2.5 for an illustration of an Erosion.

Erosion, denoted by $\ominus$, can also be extended to gray-scale images and can be described by the following formula [15]:

$$(I \ominus S)(x, y) = \min\{I(x - x', y - y') + S(x', y') \mid (x', y') \in D_S\} \tag{2.10}$$

Similar to Equation 2.9 I denotes the image, S the structuring element and D_S the domain of S. Here, $I(x, y) = +\infty, \forall (x, y) \notin D_I$ and the gray-scale Erosion can be seen as a minimum

filter for flat structuring elements.

Morphological Reconstruction

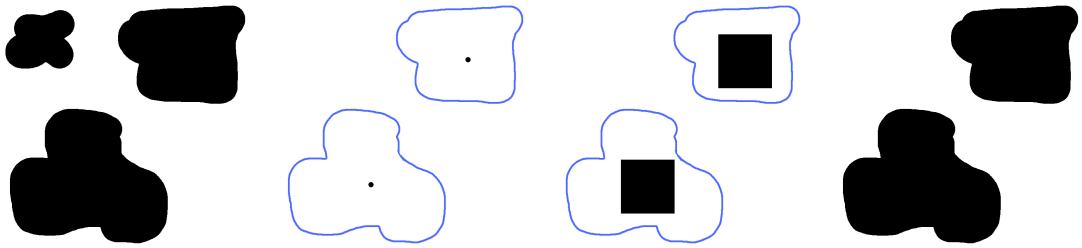
In Morphological Reconstruction there are two images and a structuring element involved. One image is the marker image, denoted by F, which is the starting point of the transformation and one is the mask image, denoted by G, which is used as constraint for the transformation [15]. The structuring element is only used to describe connectivity, e.g. the element in Figure 2.5b would define 4-connectivity. If F, G and S_r are given, Morphological Reconstruction proceeds as follows:

input : marker image F, mask image G and structuring element S_r
output: Reconstructed image H
Initialize H_1 to the marker image F
repeat
  $H_{k+1} = (H_k \oplus S_r) \cap G$;
until $H_{k+1} = H_k$;

To understand what is going on, let us consider an example. Looking at Figure 2.6a, we can see a binary image which shows a couple of objects in black. This will be the mask image G used as constraint for the Morphological Reconstruction. Let the image in Figure 2.6b be our marker image F, the initial image that will be set to H_1. We can see that it only has two small regions, the two small black dots, which are in the center of the two bigger objects of the mask G. The blue borders only serve as visualization tool and are not included in the actual marker image. Starting with the two black dots, F gets dilated by the structuring element under the restriction that the values cannot exceed the values of the mask. An intermediate result can be seen in Figure 2.6c. Note that in this example S represents 8-connectivity and not 4-connectivity as before. After several iterations there will not be any more changes and the resulting image is shown in 2.6d.

Extending the approach to gray-scale images is straight forward since we already defined how the gray-scale Dilation works. To understand the meaning of the set intersection in Algorithm 4, we have to visualize a gray-scale image as a 2-D heightfield. The intersection of two gray-scale images is then simply the intersection of the two 3-dimensional volumes formed by the heightmaps. Hence, one can picture the gray-scale Reconstruction as an operation, where the maximum values of the marker image are continuously propagated until they get bound by the mask.

What we have left out until now is how we actually create a useful marker image. One way



(a) (b) (c) (d)

Figure 2.6: Demonstration of Morphological Reconstruction. Original image (a), the mask G. Initial image (b), the marker image F. Note that the blue borders are added just for visualization purposes, the true marker image only consists of the two black dots! Intermediate result (c), after a couple of iterations. The blue borders are again added for visualization. The final reconstructed image (d).

to do it, is to erode the mask image by a certain structuring element. This operation is called Opening by Reconstruction, as it is similar to the standard Morphological Opening, where an image gets first eroded and then dilated. In Opening by Reconstruction, however, it gets first eroded and then reconstructed:

input : mask image G and structuring elements S_e and S_r for the Erosion and the Reconstruction

output: Reconstructed image H

1 $F = \ominus(G, S_e)$;

2 $H =$ reconstruct(F, G, S_r);

Ortiz and Torres' algorithm

Ortiz and Torres' algorithm can be divided into two parts: First, they identify specular highlights in an image and create a binary mask image where specular pixels are marked as 1s and second, they fill those pixels according to the surrounding diffuse ones using Morphological Reconstruction.

They find specularities by transforming the image into a color model called Serra's L1-Norme, a model similar to *Hue-Saturation-Value* (HSV). Hence, a particular color is also described by a hue, a saturation and a brightness or intensity value. To extract highlights, only the saturation s and the intensity m are of interest and they can be calculated from RGB-values as

follows [45]:

$$
\begin{aligned}
m &= \frac{1}{3}(r+g+b) \\
s &= \begin{cases} \frac{1}{2}(2r-g-b) = \frac{3}{2}(r-m) & \text{if } (b+r) \geq 2g \\ \frac{1}{2}(r+g-2b) = \frac{3}{2}(m-b) & \text{if } (b+r) < 2g \end{cases}
\end{aligned} \tag{2.11}
$$

The authors then detect highlighted pixels as those with a high value of m and a low value of s. In order to include the pixels which are right at the transition between specular and diffuse areas to the ones ought to be filled, they apply Dilation on the specularity mask yielding a new mask h. They do take into account that the contrast in different images can vary, which means that the thresholds would have to be adapted as well, so they suggest to apply a morphological contrast enhancement before the threshold operation. However, what they do not consider is that the light color can vary too. The color of highlights is strongly related to the color of the light and hence without any white-balancing, the saturation will not be low on highlights if the light color is saturated.

Next, they proceed with the inpainting of the masked pixels. Morphological operators need a complete lattice structure [55] [56], i.e. a partially ordered set where every subset has an infimum and a supremum. Color models like RGB and HSV do not represent such an ordering. Looking back at the definition of the Morphological Dilation or the Erosion, we can immediately see the problem when trying to extend the gray-scale morphology to color images. What is for example the minimum or the maximum of two RGB-values?

Hence, Ortiz and Torres propose to use lexicographical ordering to compare two values in Serra's L1-Norme. They first compare the intensity m, as this is the best way of preserving the contours [45], then the saturation s and finally the hue. Intuitively, first comparing the intensity and then the saturation does make sense, especially in the context of highlight removal, but comparing the hue directly still leaves questions unanswered. Hue is usually defined on a circle, going from red over to green over to blue. Why should green be *smaller* than blue? And how is the big gap between a purplish red and a yellowish red handled?

Using lexicographical ordering, the authors apply an Opening by Reconstruction filter as described above in Algorithm 2. To save computational time and to save the non-specular pixels from being altered, the Opening by Reconstruction only operates on the specular pixels, i.e. where $h(x, y) = 1$. The Erosion of the Opening by Reconstruction is done with a flat structuring element of size s, which has to be larger than the largest specular highlight. Due to the flat Erosion, which is equal to a minimum filter, every specular pixel gets replaced by the smallest pixel

in the neighborhood. Thus, specular highlights get removed by their surrounding diffuse pixels. Finally, the image gets reconstructed by the iterative Dilation procedure defined in Algorithm 4, where F is the eroded and G the original image.

Although Ortiz and Torres' algorithm still needs many calculations, especially due to the Opening by Reconstruction, its main idea that specular pixels get replaced by their diffuse neighbors is simple and let suggest that a fast implementation is possible. Hence, we modified and changed their algorithm in several ways to meet real-time constraints and implemented a GPU version of it, while preserving the main idea behind it. In the next chapter, when we present our BRDF estimation algorithm, we will also explain our highlight removal technique.

2.5 Normal Estimation

Fast calculation of normals in point clouds is still a hot research topic and we could have easily spent a whole thesis on that. In the context of BRDF estimation though, it is only a small step in the whole algorithm. Still, for the sake of completeness we present a few publications which we used during our research.

Klasing et al. [27] wrote a summary in which they compare different approaches for estimating normals in point clouds with respect to quality and speed. In their paper the authors come to the conclusion that a least-squares plane fitting approach using the *Principal Component Analysis* (PCA), like the technique presented by Hoppe et al. [20], is the universal method of choice when it comes to both, quality and speed. They present running times for a scenario with 14200 points, which unfortunately is far away from the number of points we need to process at a time. However, if we assume a linear growth with respect to the number of points, then their *PlanePCA* approach would need a couple of seconds on a 3.0 GHz processor if the input were a point cloud with about 300.000 points.

While the algorithms reviewed in Klasing et al.'s work [27] all run entirely on the CPU, there are also techniques which run on the GPU, e.g. Buchart et al. [6] or Jie et al. [24]. However, in Buchart et al. [6] the most time consuming step, the *K-Nearest Neighbor* (k-NN) search, is done on the CPU and Jie et al. [24], although it sounds very promising, does not give any running times of the normal estimation. Jie et al. only present the running times of their k-NN search and their surface smoothing algorithm, which uses the estimated normals. Still, the whole algorithm took less than a second for a model having 458074 points on a Tesla 1060(s) graphics card.

During our research on normal estimation we stepped over the *Point Cloud Library* (PCL) [47], a large scale, open project for point cloud processing [51]. Among other point cloud processing capabilities, PCL also supports normal estimation. Apart from a least-squares plane

fitting approach which uses exact covariance computation schemes, PCL also has a fast normal estimation feature using integral images that turned out to be particularly useful for our needs.

2.6 K-Means

K-Means is a clustering algorithm which separates an n-dimensional data set into k cluster. It was invented by Stuart P. Lloyd in 1957 and published in 1982 [32]. Therefore, it is also known as Lloyd's algorithm. The outline of the algorithm can be briefly described as follows:

1. Randomly choose k cluster centers μ_j that represented the k classes.
2. For every data point x, calculate its nearest cluster center using the Euclidean distance and assign the point to this class. Data point x_{ji} is the ith data point of class j.
3. For each class, calculate its new cluster center as the centroid of its data points, i.e.:

$$\mu_j = \frac{1}{n} \sum_i x_{ji} \tag{2.12}$$

4. Repeat step 2 and 3 until nothing changes.

K-Means is the most frequently used clustering algorithm [5]. The great advantages of K-Means are that it is simple and relatively fast compared to other clustering algorithms and thus is suitable for our approach.

2.6.1 Further Considerations on Standard K-Means

K-Means does not find the globally optimal solution though it is guaranteed to converge to some locally optimal one, which can however take many iteration [5]. Therefore the output depends on the initial cluster centers. Hence, the question of what initial cluster centers to choose comes up. Another upcoming question is how many clusters to pick, i.e. how the value of k should be chosen. Furthermore, what do we do if no data points get assigned to a cluster center, which can easily happen considering image data and choosing random initial colors.

As our goal is to achieve results in near real time using images as data input, we almost never will iterate until convergence, but we will stop after a certain amount of maximum iterations. Additionally to the fact that K-Means only converges locally, the convergence speed also depends on the initial cluster centers. A good way to conquer the problem of the right initial value is to run the algorithm several times with different initial start values taken randomly [5]. The results are then taken from the run which exhibits the least variance.

In a similar way it is possible to automatically choose the number of clusters k. One can run the algorithm several times, each time with a different k and make a trade off between the variance and the number of clusters. Note that the variance will always decrease with increasing number of clusters until eventually every data point is its own cluster center and the variance is zero [5].

The problem of an *empty* cluster center can be handled in two ways. It can either be ignored, so that the actual number of clusters is smaller than k, or we can assign it a new value. We prefer the latter approach, assigning a new random value each time we find a cluster center which has no data points.

2.6.2 GPU-based K-Means

Since we implemented a GPU-based K-Means algorithm we want to mention other GPU-based K-Means implementations as well. Both, Hong-tao et al. [19] and Li et al. [30] use CUDA for the task. Hong-tao et al. calculate the nearest cluster center for every data point on the GPU but use the CPU to re-order the data points and to calculate the number of entries per class so that the GPU can easily compute the new cluster centers. Li et al. also use a hybrid GPU/CPU approach to find the new cluster centers. Their algorithm however contains two different implementations based on the number of dimensions of the data points. If the dimension is low, they use GPU registers and if it is high, they observe a similarity between K-Means and matrix multiplication and use this observation for a speed-up. There exist many other papers that are very similar to the two aforementioned ones also using the same hybrid GPU/CPU approach, e.g. Wu et al. [70] and Zechner and Granitzer [74].

Dhanasekaran and Rubin [11] developed a new method to calculate the new cluster centers on the GPU without the need of copying lots of data between the CPU and the GPU, which is the case when using a hybrid GPU/CPU approach. They implemented their algorithm using OpenCL and in the best case they could achieve a speed-up of 3.5 times.

Dong et al. [12] also presented a GPU only K-Means implementation. They efficiently cluster *Virtual Point Lights* (VPL)s based on the idea of Scheuermann and Hensley [53]. They use 1-dimensional textures, where each texture consists of k texels with k being the number of clusters. There is for example a position texture which holds the position of the k new cluster centers and a count texture which holds the number of data points for every cluster. These textures are used as render targets. Clustering then works by rendering a point primitive for every data point i.e. every VPL, looking for the closest cluster center in the vertex shader and calculating the pixel position of the render target textures that corresponds to the closest cluster center. Additive blending is then used to accumulate the results.

In the next chapter we will present our BRDF estimation algorithm where one step consists of applying K-Means. Our implementation, which is also described in the next chapter, is quite different than the ones mentioned here since we achieve a GPU only implementation by using MIP maps.

CHAPTER 3

Interactive BRDF Estimation

We now want to move on to present our interactive bidirectional reflectance distribution function estimation algorithm. Before diving into a detailed description, an overview of the technique is given in Section 3.1. The basis of the method in this thesis lies in the work of Zheng et al. [75], which was presented in Section 2.3.2. The contribution of this thesis and the differences to Zheng et al.'s method are demonstrated in Section 3.2. Since the algorithm can be broken down nicely into several individual parts, they are separately described together with individual tests from Section 3.4 to Section 3.11 and the test scenario, which is used for the individual tests, is presented in Section 3.3. Finally, in Section 3.12 the limitations of our approach are given.

3.1 Overview

The aim of image-based BRDF estimation is to estimate the material parameters for a specific illumination model from image data. Since the material characteristics cannot be estimated alone without considering all the other parts in the rendering equation, like point locations, light sources, etc., those parameters have to be estimated as well. In Chapter 2 a more detailed overview of the problem of BRDF estimation was already given. The illumination model used in this thesis is the Phong model. We use the Phong model as opposed to Zheng et al. [75] because it achieves images of almost similar quality but it is cheaper to evaluate, see Section 2.2.3 for a comparison.

The basic idea of our algorithm is the following: First, acquire and calculate all the information needed by the rendering equation that is not related to the material characteristics, i.e. 3D point positions, normals and information about the lighting. Then, estimate the specular characteristics per object. The objects are identified by using a clustering algorithm, but since

clustering directly on the color image leads to bad results in the cluster image and therefore in the specularity estimation, the image is processed in two different stages before the actual clustering. After specularity estimation the final step is to calculate the diffuse parameters per pixel in order to handle textured surfaces.

In the following we give an outline of the algorithm together with a high-level overview of the individual steps:

1. **Data Acquisition**

   With the use of the Microsoft Kinect sensor a color and a depth image is acquired (Section 3.4.1). An environment map is obtained by the image stream of a fish-eye lens camera (Section 3.6).

2. **Data Extraction**

   The depth data is then used to calculate 3D point locations (Section 3.4.2) and point normals (Section 3.5). The environment map is used to estimate the lighting (Section 3.6).

3. **Highlight Removal**

   Highlights are removed to facilitate the clustering. Highlight pixels are detected as those having high brightness and low saturation values and a simple inpainting technique is used to get rid of them (Section 3.7).

4. **Inverse Diffuse Shading**

   The diffuse shading is removed from the color image to flatten the objects and to yield a more uniform color distribution (Section 3.8).

5. **Clustering**

   Similar colors are assumed to belong to similar materials and are therefore clustered together in order to estimate common specular reflection characteristics (Section 3.9).

6. **Specular estimation**

   The specular parameters are estimated using non linear least squares optimization per cluster (Section 3.10).

7. **Diffuse estimation**

   The diffuse parameters are estimated per pixel using the estimated specularity coefficients (Section 3.11).

In Figure 3.1 you can see an illustration of the outline of our algorithm. The individual steps of our technique are given as individual boxes and the resulting images of each step are also included. The individual steps and the resulting images will be explained in more detail in the following sections.

3.2 Scope of Zheng et al. and our Contribution

The biggest difference between the approach of Zheng et al. [75], which was reviewed in detail in Section 2.3.2, and the one presented in this thesis is the running time since interactive frame-rates must be achieved as required for mixed reality scenarios. In contrast to Zheng et al.'s method, the typical application setting of the algorithm in this work is not a controlled scenario using a special setup where the object to estimate is put on a tripod. Zheng et al. take several images while rotating the object around the tripod's axis and use Shape From Silhouette to acquire the geometry. Additionally, our method is not restricted to the estimation of material characteristics of only one object, which fits on this tripod, but whole scenes consisting of several objects can be processed. In order to fit our needs we therefore use a range scanner, which streams depth information in real-time. In particular we use the Microsoft Kinect sensor to acquire both, the geometry and the color images. Section 3.4 explains the device and how it is used for our technique in more detail.

Another major difference lies in the way how we estimate the lighting environment. While Zheng et al. use a metallic sphere to estimate the position of a single light source, we use the image stream of a fish-eye lens camera to estimate the positions of an arbitrary number of light sources. The image stream allows us to make the estimation dynamically. Hence, the system adapts automatically to changes in the lighting environment, whereas with the approach of Zheng et al. the estimation has to be re-done, requiring manual interaction, whenever the light changes.

Thirdly, since Zheng et al. only consider one light source, all their equations are only valid for a single light source scenario. Thus, we adapted the equations to fit multiple light sources. And finally, we do not fit the parameters for the Ward model but we use the Phong model for the estimation. On the one hand, Phong allows for a slight speed-up since it is simpler than Ward and on the other hand, it directly fits the needs of our Differential Instant Radiosity renderer described in Section 2.1.

Putting the contribution of this thesis with respect to the work of Zheng et al. in relationship to the components in Figure 3.1 this means that input data acquisition and in the following the data extraction is completely different. Highlight removal was modified as well to allow for a faster computation. Additionally, the specularity estimation was slightly extended beyond the necessary adaptation to the Phong model and multiple light sources.

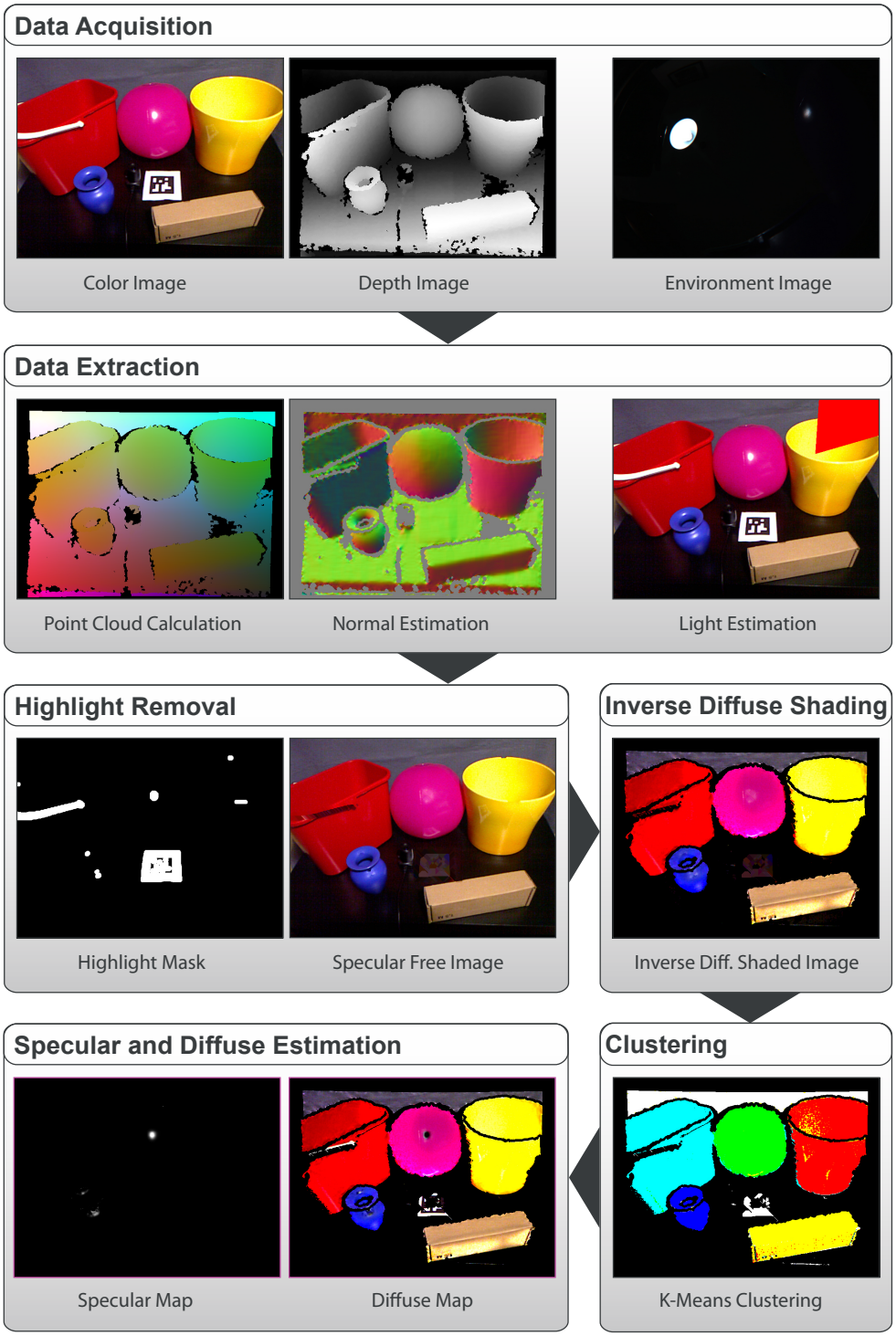


Figure 3.1: This figure gives an overview of the algorithm and shows the individual steps.

Figure 3.2: Scenario used to test the individual components of our technique. Different objects with different materials can be observed.

3.3 Test Scenario

The test scenario that we used to evaluate individual parts of our algorithm is shown in Figure 3.2. We used different objects with different materials. From left to right: A red plastic bucket, a blue vase, a pink inflatable rubber ball, a cardboard box and a yellow plant pot. The scenario was illuminated with a single 250 Watts daylight bulb. The black fish-eye camera, which can be seen in the middle of the image captures the environment and provides the data to estimate the light sources. Next to the camera is a marker, which is used to align the imaging camera and the fish-eye camera in a common world coordinate system. We use Studierstube Tracker [16] to acquire an up-to-date camera pose estimation.

3.4 Color and Geometry Acquisition Using the Kinect Sensor

Apart from the data we get by the fish-eye camera for the lighting estimation, all information is provided by the Kinect sensor [68]. There are several reasons why we chose to use the Kinect for the estimation process. On the one hand it is cheap compared to other range scanners and stereo systems. And on the other hand, we simultaneously get a color image stream so we can also use the Kinect as video see-through device in a mixed reality system. This is particularly useful, since a range scanner does not provide a whole model of the scene, but only depth values of point locations which are visible to the scanner. Hence, if we have two different cameras,

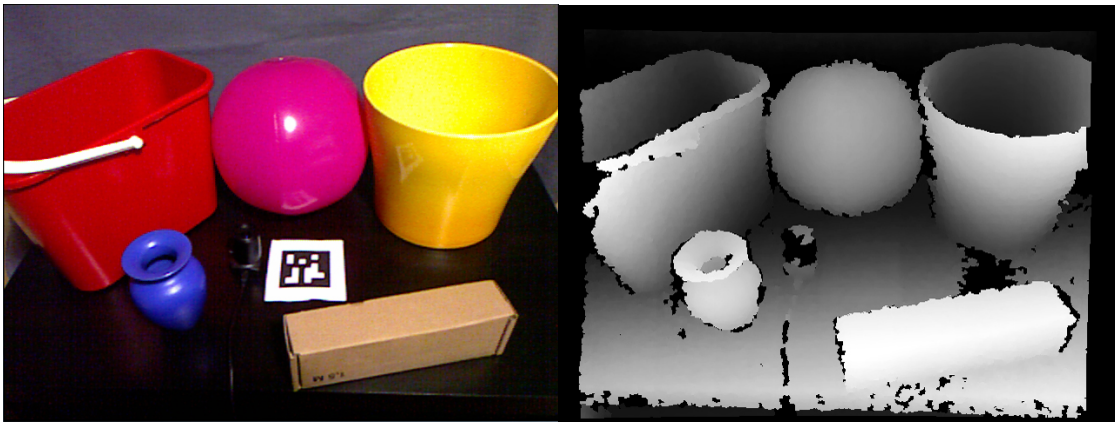
one providing the depth and the other the video, we always have to make sure that both cameras are close together and point into the same direction to combine the 3D location and the color information. If two separate cameras are used, they both need to be tracked so that the depth image and the color image can be aligned. By using the Kinect sensor, we are able to avoid these problems leading to a simpler and more robust solution. In the following sections we give detailed information about the data streamed by the Kinect and how we calculate 3D point clouds from depth maps.

3.4.1 The Data

The Kinect sensor has an RGB camera, a depth sensor and a multi-array microphone. Its depth sensor belongs to the class of active range sensors, where light is actively projected onto the scene, in contrast to passive range sensors, which only rely on intensity images to reconstruct depth [63]. More precisely, Kinect's depth sensor is a structured-light 3D scanner. It projects a specific pattern of points into the scene. From another point of view than the projector's, the pattern will be distorted and out of this deformation, depth can be calculated. The reason why the point pattern is not visible to the human visual system is because it is emitted and captured in the infrared spectrum.

Kinect streams both, color information from the RGB camera and depth information from the depth sensor at a resolution of 640×480 and at a framerate of 30Hz. The depth information is provided in form of a depth map, thus giving a depth value for every pixel of a 640×480 image. A convenient feature is that we already get the depth data in real world units, more precisely we get it in millimeters, as the internal calibration parameters seem to be stored. Due to the technology of structured-light scanning the working range is limited. For the Kinect sensor the minimum distance lies at around 45cm, whereas the maximum distance is at 10m. The practical range, though, where good depth estimation are achieved is shorter.

In Figure 3.3a and 3.3b you can see an example of a color image and a depth image, respectively. The two images are aligned in a way that the color at pixel position (x, y) corresponds to the depth at position (x, y) and vice-versa. A relationship between the color and the corresponding depth value is needed for our algorithm and by aligning the images this way, we can achieve a fast and easy mapping. The non-uniform black margins at the border of the depth image appear because the field of view of the depth camera is slightly narrower than the one of the color camera. This is negligible and we could have easily cropped the color image to get rid of them. Way more critical is however that no depth value can be obtained at the border of objects. On the one hand this is due to the fact that the infrared point projector, the infrared camera and the color camera are located at different physical positions and on the other hand because the infrared



(a) Color Image (b) Depth Image

Figure 3.3: Color and depth data from the Kinect sensor.

points get distorted too much at the boundaries and cannot be retrieved anymore. Additionally, at some points the Kinect sensor simply fails to retrieve a depth value, like at the fish-eye lens camera or at parts of the table. The missing values of the depth image are indeed a major problem since they are propagated throughout the whole algorithm and even get worse since some parts rely on neighboring information, like the normal estimation described in Section 3.5. This means that on all those points the material cannot be estimated and is therefore left out in the algorithm. 3D Reconstruction using the Kinect sensor is a very hot research topic and there will be several publications coming out in the next months, which wait for acceptance at the major conferences. Especially the work on Kinect Fusion [35] sounds particularly promising since it provides high quality 3D surface reconstruction without holes. For these reasons and because the reconstruction was not the main focus of this work we skipped this problem and hope to integrate some of the work once it comes out.

To visualize the depth map we use histogram equalization. We map the depth values from the range $[0, 10000]$ to $[0, 255]$ and invert their value, so that high depth values correspond to low grayscale values. A black pixel, i.e. one whose grayscale value equals 0, denotes a location where the depth could not be calculated.

3.4.2 From Depth Values to Point Clouds

Since the Kinect sensor is only providing us with a depth map in image space, we need to calculate the actual 3-dimensional point locations in the camera coordinate system by ourselves. This is a common problem in computer vision. In the geometric image formation process a 3D

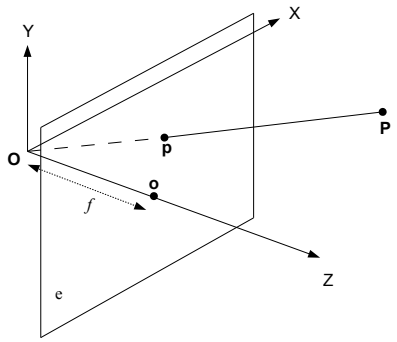


Figure 3.4: The perspective camera model. Point $\mathbf{P}$ gets projected onto the image plane e.

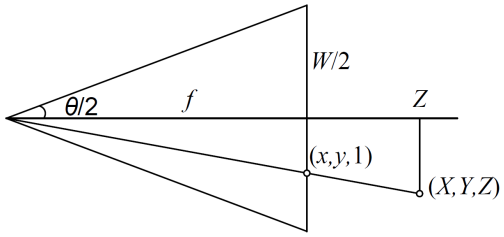


Figure 3.5: Central projection from top-down-view, showing the relationship between the focal length, the image width and the field of view. Figure taken from [60].

point $\mathbf{P} = (X, Y, Z)$ in the camera reference frame whose origin is the center of projection $\mathbf{O}$ is projected onto the image plane e yielding an image point $\mathbf{p} = (x, y)$, see Figure 3.4. If we neglect radial distortions, then this relationship can be described by the following equation [63]:

$$\begin{aligned} x &= -f_x \frac{X}{Z} + o_x \\ y &= -f_y \frac{Y}{Z} + o_y, \end{aligned} \tag{3.1}$$

where $f_x = f/s_x$ and $f_y = f/s_y$ with s_x and s_y being the effective horizontal and vertical pixel size and f being the focal length. The variables f_x and f_y describe the focal length in pixels, whereas f is usually described in millimeters. The image center $\mathbf{o} = (o_x, o_y)$, also called principal point, is the intersection of the optical axis, the Z-axis in our case, with the image plane.

What we actually want now, is the opposite of Equation 3.1. We want to infer the point $\mathbf{P}$ from the pixel coordinates $\mathbf{p}$. Solving for (X, Y, Z) yields:

$$\begin{aligned} X &= -\frac{Z}{f_x}(x - o_x) \\ Y &= -\frac{Z}{f_y}(y - o_y) \\ Z &= Z \end{aligned} \tag{3.2}$$

The only unknown variables up to now are f_x, f_y, o_x and o_y, as the Kinect sensor provides us with a depth map, thus obtaining a Z value at each pixel position $\mathbf{p}$.

Fortunately, the Kinect sensor has both, the horizontal and the vertical field of view in pixels, denoted as θ_x and θ_y respectively, already calculated and stored in its memory and it is possible to access this information via the driver. Using the following equation we can infer the focal length [60]:

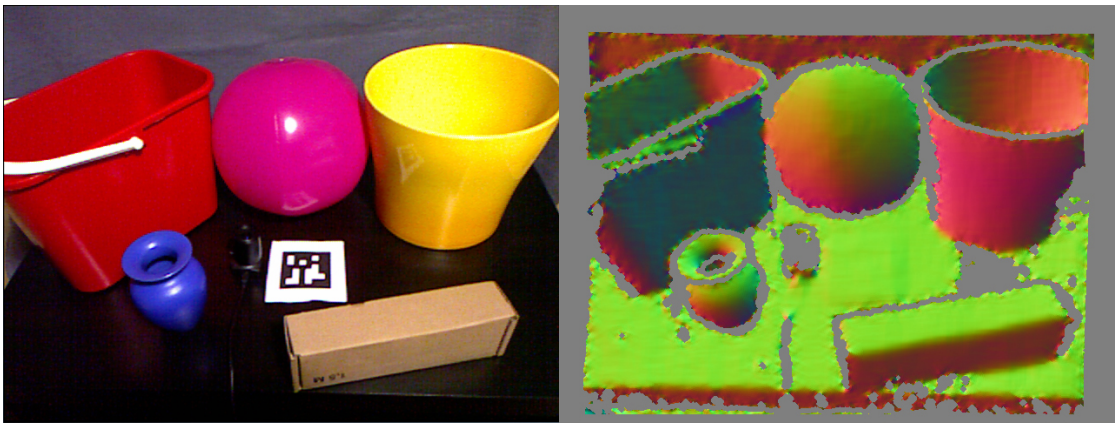
$$\begin{aligned} f_x &= \frac{w}{2\tan\frac{\theta_x}{2}} \\ f_y &= \frac{h}{2\tan\frac{\theta_y}{2}}, \end{aligned} \tag{3.3}$$

where w is the image width and h the image height. In Figure 3.5 we can see the geometric relationship between the focal length, the image width and the field of view leading to this equation. Finally, we assume that the image center $\mathbf{o}$ is very close to the center of the image, and therefore we use the values $(320, 240)$.

3.5 Normal Estimation

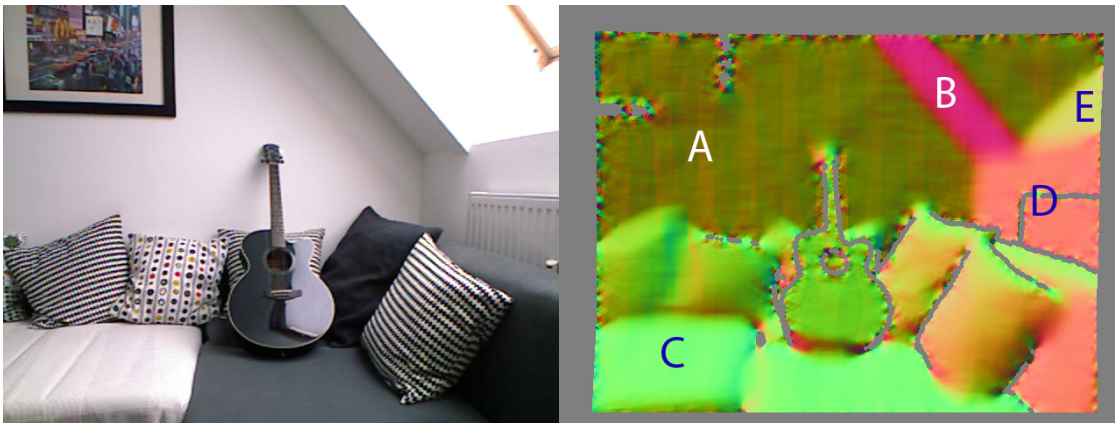
Once the 3D point location is obtained, the next step is to calculate the corresponding normal at this position, since the normal is another crucial unknown in the illumination calculation, see Section 2.3. A library is used for the task of estimating the normals, which is described in Section 4.2.3. Here, only the results which are given by the library are discussed and presented. The point normals are visualized by mapping the normalized vectors $\mathbf{n} = (n_x, n_y, n_z)$, where $n_x, n_y, n_z \in [-1, 1]$, to RGB colors $\mathbf{c} = (c_r, c_g, c_b)$, where c_r, c_g, c_b are in the range $[0, 1]$. This is done by adding 1 to each vector element and dividing by factor 2. In Figure 3.6 you can find a visualization of the results of the estimation in the test scenario. Similarly to the depth map visualization in Figure 3.3b, gray pixels correspond to the zero normal, which come from the fact that no depth value could be recovered at this location. The region where no normals could be calculated however is slightly larger than the region where no depth value was obtained, since the normal estimation works on neighboring points and therefore needs a certain number of neighbors.

In order to understand the meaning of the colors and to ensure the correctness of the estimation, Figure 3.7 demonstrates a mapping of colors at different positions to the corresponding normals. The normals are defined in a right-handed coordinate system, where the x-values grow from right to left, the y-values from bottom to top and the z-values from front to back. We can see that e.g. the white wall facing the observer turns into an olive green, which has the RGB values $(0.5, 0.5, 0)$, representing the normal vector $(0, 0, -1)$. Since the positive z-axis points into the image, this is correct. Another example is the sitting-area of the couch which turns into



(a) Color image (b) Normal map

Figure 3.6: Visualization of the estimated normals in our test scenario.



(a) Color image (b) Normal map

	Normalized point normal	Normal in RGB space
A	$(0, \quad 0, \quad -1)$	$(0.5, \quad 0.5, \quad 0)$
B	$(0.71, \quad -0.71, \quad 0)$	$(0.85, \quad 0.15, \quad 0.5)$
C	$(0, \quad 1, \quad 0)$	$(0.5, \quad 1, \quad 0.5)$
D	$(1, \quad 0, \quad 0)$	$(1, \quad 0.5, \quad 0.5)$
E	$(0.71, \quad 0.71, \quad 0)$	$(0.85, \quad 0.85, \quad 0.5)$

(c) Mapping between normals and colors

Figure 3.7: Test of the correctness of the normal estimation.

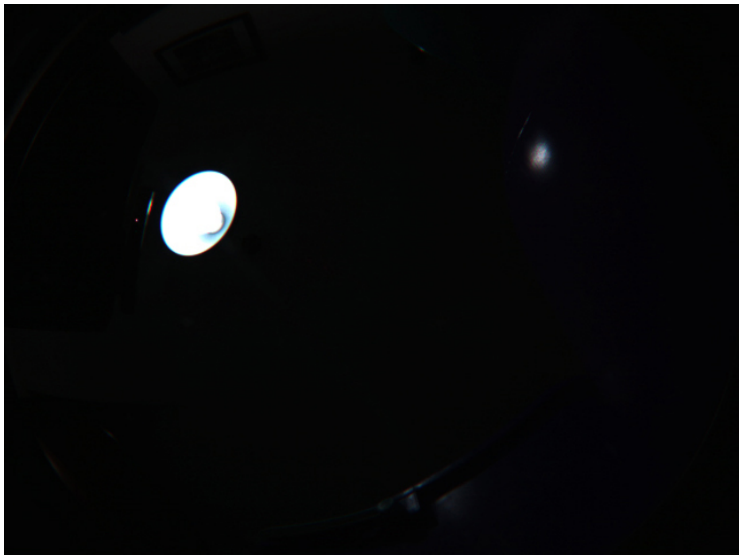


Figure 3.8: The environment map obtained from the test scenario streamed by a fish-eye lens camera.

a bright low-saturated green with RGB values $(0.5, 1, 0.5)$ corresponding to the normal $(0, 1, 0)$. The table in Figure 3.7c shows additional correspondences.

3.6 Light Estimation

The lighting environment is acquired by using a fish-eye lens camera that captures the surroundings online. In Differential Instant Radiosity by Knecht et al. [28] importance sampling is used to place *Virtual Point Light*s (VPL)s on a hemisphere according to the pixel intensities in the environment map. In the technique presented in this thesis, the importance sampling is directly used by taking the first n positions of the light sources corresponding to the VPLs with the highest illumination strength. Those are considered as the n estimated point light sources.

Typically it is not possible that the see-through camera and the fish-eye camera see a common marker, since the environment camera is usually placed to point upwards like in Figure 3.2. Therefore the fish-eye camera is assumed to be very close to the marker which defines the origin of the world coordinate system. This allows to transform the light positions that we get from the environment map in the Kinect view space.

Figure 3.8 shows the environment map captured in our test scenario. A small aperture was used to facilitate the estimation of the brightest VPL. Only a single point light source was used in this scenario. The small bright spots not belonging to the illuminant are due to reflections in the environment.

Looking back at the image presented in the overview diagram in Figure 3.1 we can see a red cone painted on top of the color image. It represents the estimated position of the light source.

3.7 Highlight Removal

Highlight removal is the process of eliminating specular reflections on objects, as described in Section 2.4. Removing highlights is crucial for estimating the specular reflection coefficients in materials, if individual objects are identified by clustering and the specular coefficients are estimated per cluster. The problem is that specularities represent the illuminant's color, which may be very different from the object's color, leading to a strong cut in the intensity function. Hence, the highlights will most likely be identified as individual objects rather than to be clustered together with the object on which they occur. However, if an object is specular and a highlight is visible on the image, the correct specular reflection coefficients for that object can only be estimated if the highlight *is* clustered together with the object.

Similarly to Zheng et al. the highlight removal technique by Ortiz and Torres [45] presented in Section 2.4.2 is used for this task. However, only the main idea behind the algorithm is used and the way how the highlights are actually removed is quite different than the way presented in the work of Ortiz and Torres. While they use Serra's L1-Norme as color space, which is a variant of the *Hue Saturation Value* (HSV) model, here the *Hue Saturation Intensity* (HSI) color model is used, since „the HSI model is an ideal tool for developing image-processing algorithms based on color description" [15]. We do not use the HSV model since „its focus is on presenting colors that are meaningful when interpreted in terms of a color artist's palette" [15]. The following conversion from RGB to HSI is used [14]:

$$
\begin{aligned}
\theta &= \cos^{-1} \frac{\frac{1}{2}(2r - g - b)}{\sqrt{(r-g)^2 + (r-b)(g-b)}} \\
H &= \begin{cases} \theta, & \text{if } b \leq g \\ 360 - \theta, & \text{if } b > g \end{cases} \\
S &= 1 - \frac{3}{r+g+b} \min(r, g, b) \\
I &= \frac{1}{3}(r + g + b)
\end{aligned}
\tag{3.4}
$$

Specular pixels are detected as those exhibiting high brightness and low saturation values. We achieve good results by using 0.9 as brightness threshold and 0.1 as saturation threshold,

assuming the colors are normalized in the range $[0, 1]$. To be more specific: A pixel is marked as a highlight pixel if and only if $I > 0.9 \wedge S < 0.1$.

In contrast to Ortiz and Torres it is taken into account that the color of the specularity depends on the color of the light source. Since the Kinect sensor comes with auto white-balancing on board it is assured that specular highlights are really those where the saturation is low, if the light color is uniform across the scene.

The output after the HSI transformation and the specularity detection is a binary image containing 1s at specular highlights and 0s elsewhere. We then follow Ortiz and Torres suggestion to dilate the specularity mask in order to include the transition between specular and diffuse areas as well. A circle of radius 4 is used as structuring element for the Morphological Dilation.

The next step of Ortiz and Torres' algorithm is to apply Morphological Vectorial Opening by Reconstruction on the pixels, which correspond to highlights. However, this requires a Vectorial Erosion first and then a rather time-consuming Reconstruction. The result of applying Opening by Reconstruction is that first, all the highlight pixels get replaced by the minimum value of the neighborhood defined by a structuring element which has to be larger than the largest highlight (Erosion). In the following step the maximum value of those eroded highlight pixels is spread across the highlight area for a uniform inpainting (Reconstruction), see Section 2.4.2 for further details. The specular pixels are thus replaced by the surrounding diffuse pixels in a way that they all exhibit the same RGB-value. Note that the true object color is not known at highlights and the operation is simply a heuristic to eliminate highlights by using the surrounding pixels.

In order to speed things up and to be able to do the process in one pass instead of one Erosion pass and n Reconstruction passes, the following strategy is applied to the pixels that have previously been detected as highlight:

1. Starting from the current pixel, process the neighborhood pixel by pixel in a circular way, until a non-specular pixel is found.

2. Replace the value of the current pixel with the diffuse one from the last step.

In Figure 3.9 a sketch of the pixel traversal can be found.

In Figure 3.11 the result of our highlight removal on different test scenes is shown, together with the original color image and the specularity mask. Figure 3.11a shows a scenario containing juggling balls. Light comes through a window and leads to specular highlights. In Figure 3.11b we can see the results when applied to the test scenario which is used in this chapter. Note that some regions are wrongly classified as highlight, like the optical marker or the handle of the bucket, since they are also barely saturated and very bright.

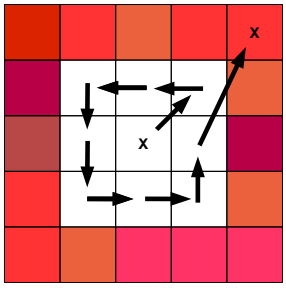


Figure 3.9: Demonstration of the order in which the neighboring pixels are processed to find a diffuse pixel. The current specular pixel, which is in the middle in this figure, gets replaced by the first non-specular pixel found, marked by the x.

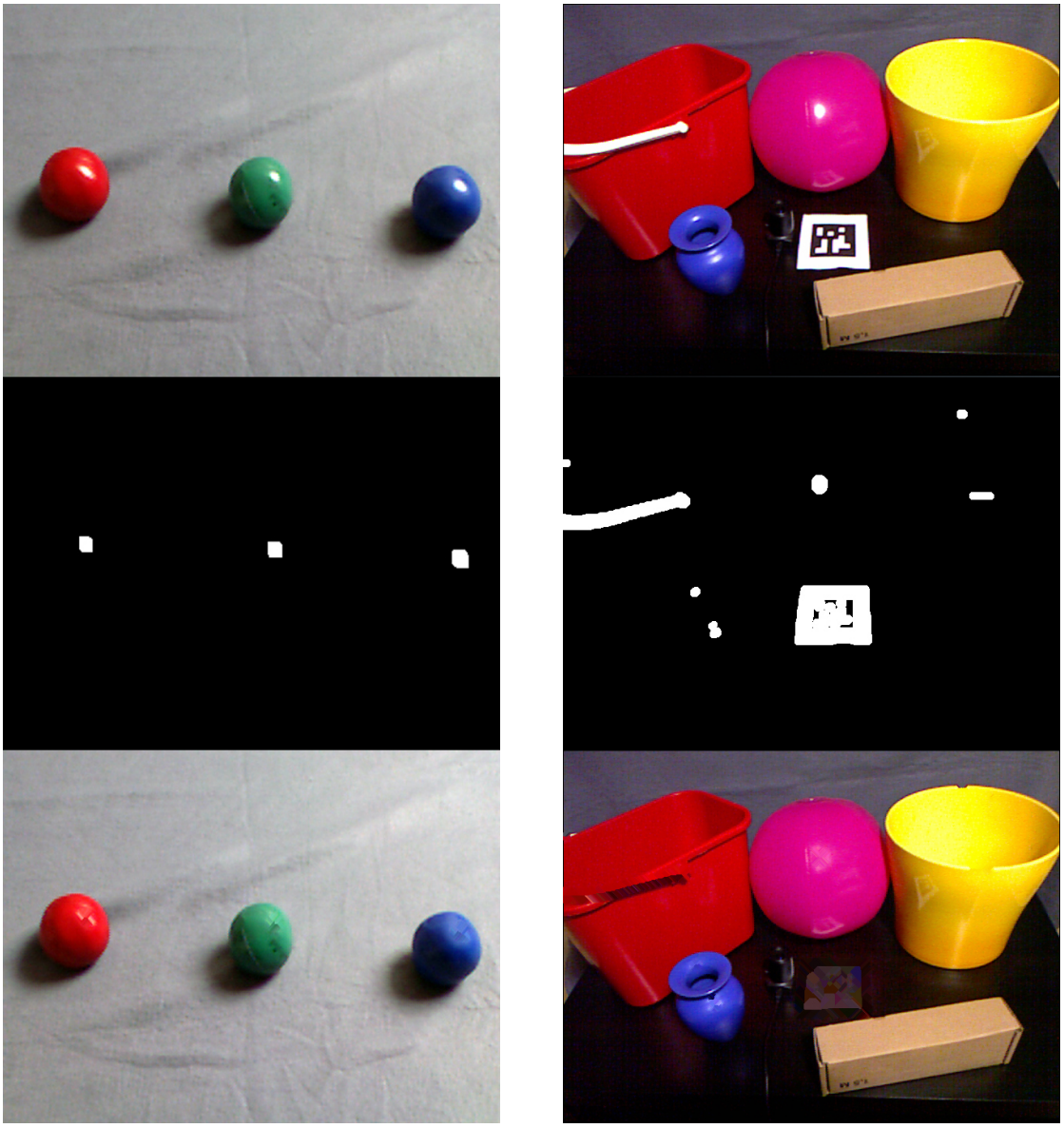
(a) Original image (b) Ortiz [44] (c) Ours

Figure 3.10: Comparison of the highlight removal of Ortiz [44] and our modification.

3.7.1 Comparison to the Original Version of Ortiz and Torres

Figure 3.10 shows a comparison of the original algorithm of Ortiz and Torres [45] and our modification. Since the original paper was only available in gray scale, we used the images of a later publication by Ortiz [44], where he presents a more efficient implementation of the algorithm. In both images the inpainted regions are clearly visible, although in the original version by Ortiz the inpainted regions are smoother than in our version. This is however neglectable because it merely serves as input for the clustering.

The processing time of the optimized highlight removal by Ortiz [44] is 1090 milliseconds for the image in Figure 3.10. In contrast, our implementation only needs 0.94 milliseconds on a Nvidia GTX 580. Unfortunately the author does not specify on which CPU the timing was measured, although since the paper is from 2007 and we are 3 orders of magnitudes faster, it does not really matter to show the huge performance gain.



(a) (b)

Figure 3.11: From top the bottom: The color image, the specularity mask and the resulting image with removed highlights of (a) a test scenario containing juggling balls and (b) the test scenario used throughout this chapter.

3.8 Inverse Diffuse Shading

After highlight removal another pre-clustering step is applied to further improve the identification of objects, or rather groups of similar materials since one object may exhibit different materials. The shading of an object at one point depends on the angle between the normal at that point and the vector to the light source(s). This fact however poses a problem to clustering algorithms that operate in RGB space, since the values between a strongly illuminated material of a specific color and a barely illuminated one cover a wide range. Hence, Zheng et al. propose to only assume diffuse objects, which is partly true since specular highlights have already been removed in the previous step, and solve the illumination equation for the diffuse parameters. Here, the equations adapted to the Phong model and to multiple light sources are given. Starting with the rendering equation given in Equation 2.3 from Section 2.2.1 we derive the diffuse component as:

$$\mathbf{k_a I_a} + \sum_{l=1}^{n} \mathbf{I}_l[\mathbf{k_d}(\mathbf{N} \cdot \mathbf{L}_l) + \mathbf{k_s}(\mathbf{V} \cdot \mathbf{R}_l)^{n_s}] = \mathbf{I} \tag{3.5}$$

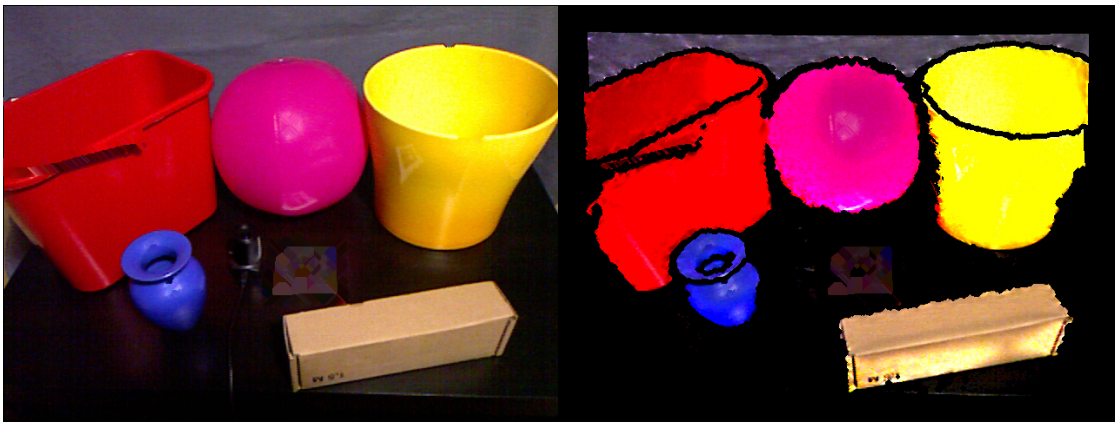
$$\mathbf{k_d} \sum_{l=1}^{n} \mathbf{I}_l(\mathbf{N} \cdot \mathbf{L}_l) = \mathbf{I} - \mathbf{k_a I_a} \tag{3.6}$$

$$\mathbf{k_d} = \frac{\mathbf{I} - \mathbf{k_a I_a}}{\sum_{l=1}^{n} \mathbf{I}_l(\mathbf{N} \cdot \mathbf{L}_l)}, \tag{3.7}$$

where $\mathbf{I}$ is the pixel color given by the color image and therefore known.

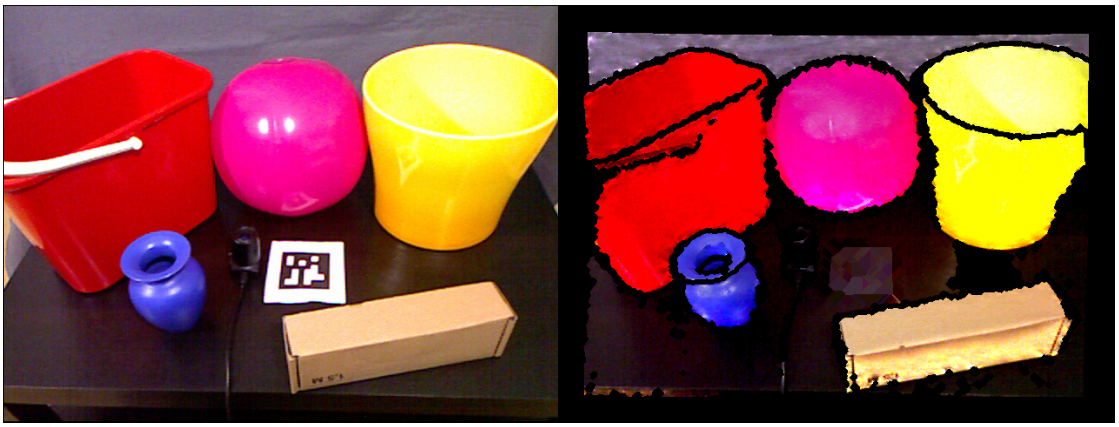
In Figure 3.12 you can see the, what we call, *inverse diffuse shaded* image together with the highlight removed image. The benefits for the clustering are directly visible. We can see that especially the yellow plant pot and the blue vase show a more uniform color distribution. The strong variations on the curtain in the background are due to the fact that it is wrinkled and therefore the computed normals vary.

Although the equations above are valid for an arbitrary number of point light sources, we get better results if, for every point, we only take the one light source into account, which contributes the most to the point's illumination. This is strongly encouraged due to the fact that a real light source may be modeled by several virtual point light sources and that the number of VPLs per real light source can easily be unbalanced, since they are assigned randomly. In Figure 3.13 you can see the inverse diffuse shaded image of the same scene using two light sources.



(a) Highlight removed image (b) Inverse diffuse shaded image

Figure 3.12: Results of the inverse diffuse shading stage operating on the highlight removed image. Note the smoother color distribution along the objects, which facilitates the clustering.



(a) Color image (b) Inverse diffuse shaded image

Figure 3.13: Inverse diffuse shading in a scene with two light sources. (a) shows the color image and (b) the resulting inverse diffuse shaded image.

3.9 Clustering

Similarly to Zheng et al. [75], the clustering of surfaces with similar characteristics is done by using K-Means, which was described in Section 2.6. K-Means has the advantage of being a fast and simple approach of data clustering in any domain. The drawback however is that the number of clusters needs to be specified manually, although there are possibilities to automatically choose k but requiring multiple iterations of K-Means, see Section 2.6. We also implemented

a version which automatically chooses the number of clusters but removed it again because of performance, so like Zheng et al. it needs to be specified manually.

The clustering operates on the inverse diffuse shaded image presented above. Clustering is one of the key parts of the algorithm, since the specular parameters are calculated per cluster. It is also one of the steps, which has the biggest performance impact, because K-Means requires multiple iterations, where each iteration itself requires *width* $\times$ *height* $\times$ k distance calculations, with k being the number of clusters. Although there exist several improvements to reduce the number of calculations and well-engineered libraries providing efficient CPU implementations, we make use of the inherent potential of parallelization and developed a GPU implementation to speed up K-Means clustering.

Unlike the approaches presented in Section 2.6.2 we do not use CUDA or OpenCL but use DirectX and HLSL. This is beneficial for us since using CUDA in conjunction with a conventional graphics library typically requires a context switch, which is expensive. Additionally, in contrast to most GPU K-Means implementations we developed a GPU-only technique, which does not require to copy lots of data between the RAM and the video memory in every iteration. Typically the GPU is used to calculate for every data point its nearest cluster center and the CPU is used to calculate the update of the cluster centers, which requires to calculate k-centroids. We implemented the centroid calculation also on the GPU by using MIP-maps.

Below we present the details of our novel K-Means GPU implementation. Later in Chapter 4 we compare the performance of our implementation with a well-known computer vision library and give further implementation details.

3.9.1 GPU-based K-Means

The main performance potential of graphics processing units lies in concurrency. What GPUs are good at and what they are designed for is to do the same calculations for many different elements at the same time, rather than doing it sequentially in an element-by-element way. Thus, we can expect a boost in performance if we find a way to parallelize the calculations in an algorithm. Looking back at the standard K-Means algorithm presented in Section 2.6, we can see that in step 2 lies the main potential. For every element, in our case there are 307200 as we are dealing with images of size 640×480, it is necessary to calculate the distance to every cluster center and choose the closest one. This way the k clusters are formed. Note that the clusters describe the k different classes and hence we will use both terms synonymously.

Once for every data point its nearest cluster center is known, the new cluster centers μ_j have to be computed as the centroid of its data points, i.e. $\mu_j = \frac{1}{n} \sum_i x_{ji}$. This is the part where it gets tricky. Calculating the average requires to compute the sum of a number of elements,

which is not something that is particularly well suited for parallelism. The problem is that the calculations are dependent from each other as obviously the sum of n data points is 1 element and not n independent new elements as it is in the previous step. Although one could do the concurrent distance calculations on the GPU, copy the output from the memory of the graphics card to the RAM, compute the new cluster centers using the CPU and transferring the results back to the video memory, the performance gain is low compared to a high performance CPU-only implementation.

One solution to the problem of efficiently calculating the aforementioned average is by using a feature called MIP-mapping. It is commonly used when textured objects are rendered and is therefore provided by GPUs in a fast way. MIP maps are texture reduction patterns. Starting with the original texture of size $2^n \times 2^n$ the texture size is continuously reduced to $1/4$, yielding first a texture of size $2^{n-1} \times 2^{n-1}$, then of size $2^{n-2} \times 2^{n-2}$,... until eventually it reaches the size of one texel [18]. Let level 0 denote the original texture of size $2^n \times 2^n$ and level n denote the texture of size 1×1. Then every texel of level i is the average of the four corresponding texels at level $i-1$. Thus, the texture at the highest level, which consists only of 1 texel holds the average value of all the texels of the original texture. In the following we formalize this finding. For simplicity we take an 4×4 texture, for which we show that its average texel value is the same as the texel value of its top-most MIP map level, but the proof can easily be extended to bigger textures. Left is the average value of the 16 texels of the original texture and right is the outcome after 2 MIP map operations:

$$\frac{1}{16}\sum_{i=1}^{16} x_i = \frac{\frac{1}{4}\sum_{i=1}^{4} x_i + \frac{1}{4}\sum_{i=5}^{8} x_i + \frac{1}{4}\sum_{i=9}^{12} x_i + \frac{1}{4}\sum_{i=13}^{16} x_i}{4} \tag{3.8}$$

$$\frac{1}{16}\sum_{i=1}^{16} x_i = \frac{\frac{1}{4}(\sum_{i=1}^{4} x_i + \sum_{i=5}^{8} x_i + \sum_{i=9}^{12} x_i + \sum_{i=13}^{16} x_i)}{4} \tag{3.9}$$

$$\frac{1}{16}\sum_{i=1}^{16} x_i = \frac{1}{16}(\sum_{i=1}^{4} x_i + \sum_{i=5}^{8} x_i + \sum_{i=9}^{12} x_i + \sum_{i=13}^{16} x_i) \tag{3.10}$$

$$\frac{1}{16}\sum_{i=1}^{16} x_i = \frac{1}{16}\sum_{i=1}^{16} x_i \tag{3.11}$$

There is still one problem to conquer however. The average needs to be calculated per cluster and the data stored in one texture gives us one average value. Hence we render the data points belonging to the different classes in separate textures and calculate the k new cluster centers by applying MIP-mapping to all those textures. In order to correctly calculate the centroid of each class, we need to know the number of elements per class since the texture size will usually be

bigger than the number of elements of a class. If s is the texture size, then the highest MIP map of one of the k textures containing the data points of a class is:

$$\frac{1}{s}\sum_{i} x_i, \tag{3.12}$$

if all other texture entries not belonging to a valid entry are set to 0. The final cluster center is then calculated by multiplying with s and by dividing by the number of data points of this class.

What we are still missing are those numbers of data points per class. To calculate them, we apply the same trick using MIP maps again. For every cluster we have an additional texture - in practice one channel of a texture is sufficient - where we place a „1“ at the current position if the data point belongs to this cluster. All other entries are „0“. The actual number of entries is then the value of the one texel at the highest MIP map position multiplied by the size of the texture at level 0.

From Section 2.6.1 we know that the result of K-Means depends on the initial cluster centers, which are chosen randomly in our implementation. Thus, we applied the enhancement to run the algorithm several times, each time with different randomly chosen initial values and choose the run which exhibits the lowest variance. The variance of one run is calculated by taking the average of the k variances of the individual cluster. To calculate the variance of one cluster we use the total variation, which is the trace of the covariance matrix of the cluster's data points [54]. It can be calculated as [67]:

$$\sum_{i=1}^{p} Var(X_i) = \sum_{i=1}^{p} \frac{1}{q}\sum_{l=1}^{q}(x_{li}-\mu_i)^2 = \frac{1}{q}\sum_{l=1}^{q}\Big[\sum_{i=1}^{p}(x_{li}-\mu_i)^2\Big], \tag{3.13}$$

where p is the number of dimensions, which is 3 in our case and q is the number of data points in the cluster. $\sum_{i=1}^{p}(x_{li}-\mu_i)^2$ is the squared euclidean distance between a data point x_l and its cluster center μ. These distances for all the 307200 points have already been calculated at the time the data points get assigned to a cluster center and thus do not require any additional computations. From Equation 3.13 we observe that the total variation can be calculated as the average value of the squared euclidean distances between the data points and their cluster center. Once again MIP-mapping is exploited to quickly acquire this average.

Before diving into the pseudocode we want to point out that the average is only correctly computed using MIP-mapping if the texture is of size $2^n \times 2^n$. Since the image from the Kinect sensor is of size 640×480 some kind of upsampling is therefore needed. In practice we only process the 640×480 image and write the data where the average is to be calculated in 1024×1024 textures. Algorithm 3.1 shows the pseudocode of our K-Means implementation. The algorithm is divided into two passes. The first pass is used to calculate the nearest cluster center

for every input texel and to separate the data points into different textures according to the cluster they belong to. After generating MIP maps for all textures, the second pass is used to compute the new cluster centers along with the variance of each class. After a couple of iterations the process is stopped and repeated with new randomly chosen initial clusters. The final output are the cluster centers with the lowest mean total variation over all cluster centers.

3.9.2 Visualization

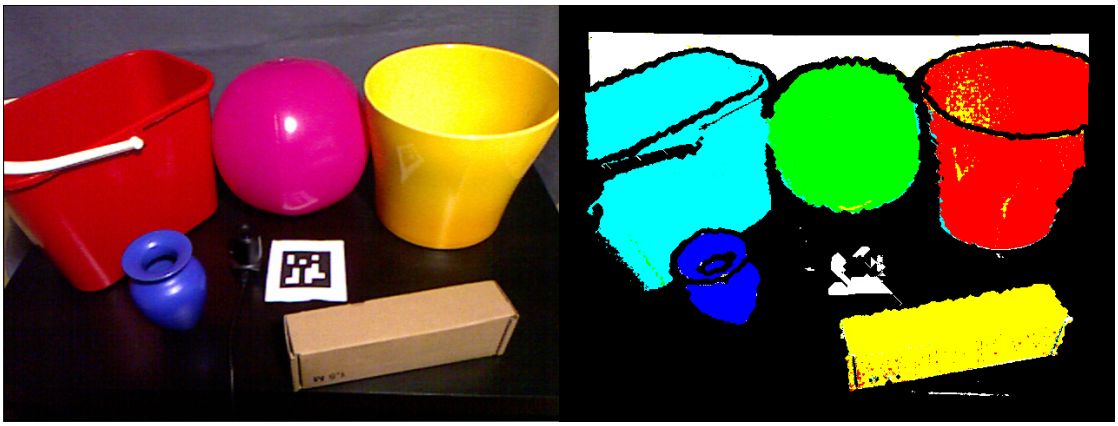
As our implementation of the K-Means clustering algorithm runs at interactive frame-rates, it is possible to cluster consecutive frames provided by a video camera. Visualizing the results however, is not as easy as one would think at first glance, since the initial cluster center are randomly chosen. It is not a good choice to assign, let us say, cluster 1 to color 1, cluster 2 to color 2 and so on, as this results in a disturbing flickering in the resulting video stream. Even if the result on two consecutive images leads to similar clustered regions, it is very unlikely that the corresponding clusters have the same color, because any ordering of the cluster centers is lost in the randomization.

The way we are dealing with this problem is that we save the final cluster centers c_{j0} together with the corresponding visualization colors of the first frame as reference values. For all the cluster centers in the consecutive frames we then simply find the closest cluster center c_{j0} and assign it the same color that c_{j0} has been assigned. Using this simple heuristic, the flickering can be reduced dramatically.

3.9.3 Clustering Results

Figure 3.14 shows a visualization of the clustering in the test scenario from Section 3.3. Different colors have been assigned to different clusters. Note that the blue vase happens to be visualized also in blue, although this is just a coincidence. Every color except black corresponds to one cluster. There are multiple different reasons why a pixel gets assigned to the class of black or zero pixels. Either one of the following may occur:

- no depth value could be generated or
- no normal could be calculated or
- there is no direct illumination at this point or
- the RGB-values are very dark and are therefore considered to be shadowed.



(a) Original color image (b) Cluster image

Figure 3.14: Visualization of the K-Means clustering.

In this scenario k was set to 6, so apart from the class of zero pixels 6 cluster were used. Note that except for a couple of pixels which are falsely classified, the clustering provides good material detection results. Due to the highlight removal the pink plastic ball as well as the blue vase were clustered correctly, whereas if you look at the original color image, which is also shown in Figure 3.14, strong highlights are visible. Additionally, the red bucket and the yellow plant pot were clustered consistently, although they exhibit varying colors due to varying surface normals. This is due to the inverse diffuse shading.

Although not clearly visible from the cluster image in Figure 3.14 we want to point out that one class of material must not only consist of neighboring pixels. With the use of K-Means only the neighborhood in the color space is used for the clustering. This way, two objects with the same material will be clustered together even if they are not overlapping in the image.

3.10 Specular Reflectance Estimation

The estimation of the specular parameters is done using Zheng et al.'s [75] suggestion to apply a non-linear optimization at every cluster j using as objective function:

$$F_j = \sum_i \left[\mathbf{I}_i - [\mathbf{k_a} \mathbf{I_a} + \sum_{l=1}^{n} \mathbf{I}_l \mathbf{k_d} (\mathbf{N} \cdot \mathbf{L}_l) + \mathbf{I}_l \mathbf{k_s} (\mathbf{V} \cdot \mathbf{R}_l)^{n_s}] \right]^2, \tag{3.14}$$

where $\mathbf{I}_i$ denotes all the pixel intensities of the current cluster j. As diffuse parameters $\mathbf{k_d}$ the cluster centers are taken for each class and they are set to be fixed.

input : An image I stored in a 2D Texture of size $x \times y$, number of clusters k
output: k cluster center c_j

Create the render target textures $clusters_j$ and $numEntries_j$ of size $2^n \times 2^n$ such that $x, y < 2^n$ and $x, y > 2^{n-1}$;
$variance = \infty$;
for 1 **to** *NUMRUNS* **do**
    Randomly choose k initial cluster center $currentc_j$;
    for 1 **to** *NUMITERATIONS* **do**
        `// first GPU pass`
        for *every texel* t *of* I **do**
            Find the nearest cluster center $currentc_j$;
            Write the RGB values of t in the R-, G-, and B-channel of $clusters_j$;
            Write the squared distance to $currentc_j$ in the A-channel of $clusters_j$;
            Write 1 in $numEntries_j$;
        end
        Generate MIP maps for all textures;
        `// second GPU pass`
        for $j \leftarrow 1$ **to** k **do**
            $avgColor =$ R-, G- and B-channel of highest MIP map of $clusters_j$;
            $avgVar =$ A- channel of highest MIP map of $clusters_j$;
            $actualNumEntries =$ highest MIP map of $numEntries_j$ * $(2^n * 2^n)$;
            $currentc_j = \frac{avgColor*(2^n*2^n)}{actualNumEntries}$;
            $currentVariance_j = \frac{avgVar*(2^n*2^n)}{actualNumEntries}$;
        end
    end
    if $\frac{1}{k}\sum_{j=1}^{k} currentVariance_j < variance$ **then**
        $variance = \frac{1}{k}\sum_{j=1}^{k} currentVariance_j$;
        for $j \leftarrow 1$ **to** k **do**
            $c_j = currentc_j$;
        end
    end
end

Algorithm 3.1: K-means GPU implementation.

There are still a couple of things which we do differently than Zhen et al. though. While they use the Levenberg-Marquardt non-linear optimization, we use the Nelder-Mead algorithm [38], since we empirically achieve good results with respect to both quality and speed. Another difference is that since we do not have multiple observations of a single surface point but work on a per image basis, we do not need to sum over all those observations. And thirdly, we also let the light position vary within a small region around its estimated location.

We discovered that due to errors in the estimation of the 3D points, the normals, the light sources and the camera pose it is rather unlikely that the specular highlight will be exactly at the same place as it is on the original image. This is however mandatory for a good estimation. Imagine the calculated specular highlight is slightly off and only covers half of the original highlight. Then it is much weaker since it also covers many non-specular pixels on the real image. Thus, if the light positions can be varied by the optimization, it will automatically correct the inaccuracies.

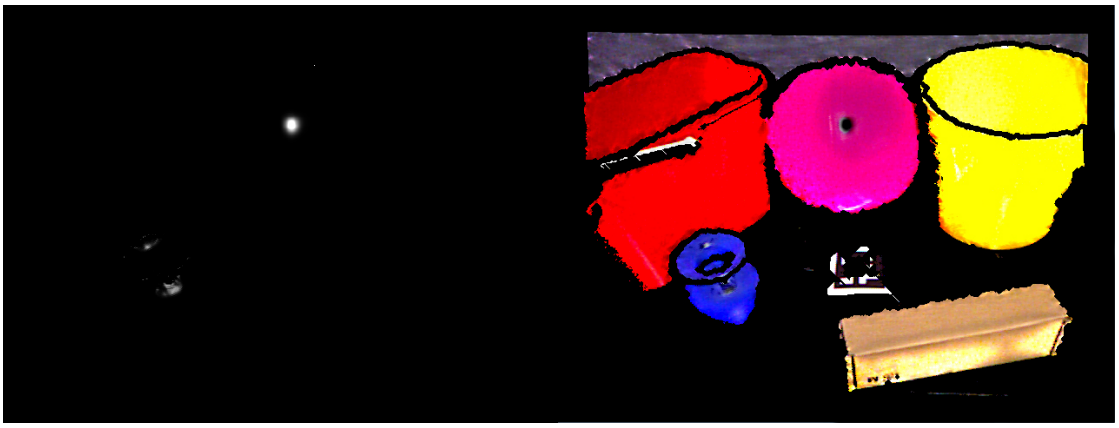
In Figure 3.15a you can see the result of the specularity estimation. The point cloud was rendered using Equation 2.3 and the diffuse parameters as well as the ambient term was set to zero. To simplify the non-linear estimation and to make it more robust we restricted ourselves to white highlights. Hence we only estimate one value for $\mathbf{k_s}$ and used it for all the three components. As we mentioned before the Kinect sensor does white-balancing on the color image and hence if the light color is uniform across the scene, we can assume that specular highlights will indeed be white.

To prove that the specular parameters of the Phong model are truly estimated and not simply given as diffuse texture, we provide in the following the values of the estimated parameters for the plastic ball and the vase:

Object	$\mathbf{k_s}$	n_s
Pink plastic ball	1.0	39.60
Blue vase	0.6839	20.55

3.11 Diffuse Reflectance Estimation

After the specular parameters have been estimated per cluster, the diffuse parameters are re-calculated to allow diffuse textures, as presented by Zheng et al. [75]. Similarly to the inverse diffuse shading we can calculate the diffuse parameters using the rendering equation. This time we include the specular term as well:



(a) Estimated specular component (b) Estimated diffuse component

Figure 3.15: Results of the material estimation.

$$\mathbf{k_a I_a} + \sum_{l=1}^{n} \mathbf{I}_l[\mathbf{k_d}(\mathbf{N} \cdot \mathbf{L}_l) + \mathbf{k_s}(\mathbf{V} \cdot \mathbf{R}_l)^{n_s}] = \mathbf{I} \tag{3.15}$$

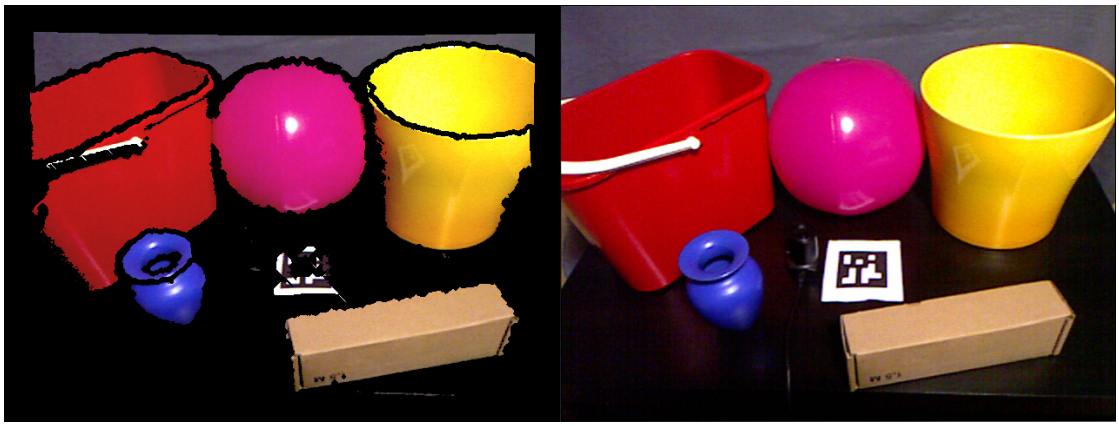
$$\sum_{l=1}^{n} \mathbf{I}_l \mathbf{k_d}(\mathbf{N} \cdot \mathbf{L}_l) + \sum_{l=1}^{n} \mathbf{I}_l \mathbf{k_s}(\mathbf{V} \cdot \mathbf{R}_l)^{n_s} = \mathbf{I} - \mathbf{k_a I_a} \tag{3.16}$$

$$\mathbf{k_d} \sum_{l=1}^{n} \mathbf{I}_l(\mathbf{N} \cdot \mathbf{L}_l) = \mathbf{I} - \mathbf{k_a I_a} - \sum_{l=1}^{n} \mathbf{I}_l \mathbf{k_s}(\mathbf{V} \cdot \mathbf{R}_l)^{n_s} \tag{3.17}$$

$$\mathbf{k_d} = \frac{\mathbf{I} - \mathbf{k_a I_a} - \sum_{l=1}^{n} \mathbf{I}_l \mathbf{k_s}(\mathbf{V} \cdot \mathbf{R}_l)^{n_s}}{\sum_{l=1}^{n} \mathbf{I}_l(\mathbf{N} \cdot \mathbf{L}_l)} \tag{3.18}$$

Since we have 3 unknowns and 3 equations we can directly calculate $\mathbf{k_d}$. Figure 3.15b shows the diffuse parameters per point. Note that $\mathbf{k_d}$ goes towards zero at highlights, as the diffuse information is lost there, since we only use the information of one image. An alternative is to use the inverse diffuse shaded image from Section 3.8 if a more complete diffuse map is desired.

Finally, we present a rendering of the point cloud using the estimated parameters and using the Phong model in Figure 3.16. An interesting observation is that the reflections on the yellow plant pot, which are visible on the original color image, are still visible in the rendering of the estimated parameters, although they are not due to the specular parameters. Those reflections are baked into the diffuse texture similar to an environment map.



(a) (b)

Figure 3.16: Phong rendering of the point cloud using the estimated parameters (a), and the original color image (b). Note that the reflections on the yellow plant pot are automatically baked into the diffuse texture.

3.12 Limitations

Our BRDF estimation combines techniques from different areas of computer vision and computer graphics and limitations therefore arise due to several reasons. Almost every individual step presented in this chapter has its own limitations and requirements and since they are combined, the whole algorithm demands that all the requirements from all the individual steps are fulfilled. In the following we only want to present the ones with the most impact.

The most natural limitation comes with the way how we acquire the geometry. The Kinect sensor does a great job in providing depth values, but there are certain materials where no depth can be obtained, e.g. very black objects or very specular, almost mirror-like objects. Additionally since the Kinect uses infrared light for the depth estimation, it is unusable in daylight outdoor scenes, as the point pattern is not recognized anymore.

Particularly problematic is the absence of depth information at the border of objects. Due to the normal estimation these regions are further enlarged since it operates on neighboring points. Especially for occlusion this is problematic because if we use the original color image for the mixed reality rendering, the occlusion will partially look wrong at the borders of real objects, where no depth value could be obtained. As described in Section 3.4.1 we wait with excitement on the current research on Kinect Fusion [35] which promises to provide detailed reconstruction without any holes.

The color image stream from the Kinect sensor also poses some limitations. With the driver we used, which will be named in the next chapter, it is not possible to control brightness sensitiv-

ity. This is done completely automatically and sometimes quite radically making the image look overexposed. If the color image is overexposed however, specular highlights will look blurred and the estimation has problems in reproducing it.

Furthermore, the way how the light estimation is done in our mixed reality renderer is not particularly well suited for BRDF estimation in scenes with multiple light sources. As mentioned earlier, the VPLs are placed using importance sampling and we take the first n VPLs which correspond to the brightest pixels. Since importance sampling is used, the strongest VPLs do not automatically correspond to the brightest pixels but it can only be said that they are likely to be near them. The light positions have a strong impact on the illumination though and we cannot simply increase the number of light sources to increase the accuracy, as this would have a crucial impact on the performance.

Apart from inaccuracies in the light position estimation, another problem is that VPLs may be distributed unevenly along equally strong physical light sources. If we have two physical light sources and we use the strongest two VPLs it can happen that both of them are close to the same physical light source position. An improvement in the light source estimation is therefore definitely part of future work.

The restriction to white specular highlights may sound like a huge restriction, but in reality due to the human perception it is not. A white object will, after a certain adaptation time, look white even if the light color is not exactly white. The human visual system adapts to the lighting environment so that the color stays approximately constant. In our system we somehow model this effect by applying white balancing on the color image and therefore assume white highlights. However, we restrict ourselves to a uniform light color across the scene as otherwise the white balancing would fail.

Finally, there are some limitations of the materials of objects due to the highlight removal. Highlights are detected as those pixels that are lowly saturated and very bright. Hence, white objects are very likely to be classified as highlight and need to be omitted in the estimation. In the next chapter we will present implementation related information and give details about the hardware and software configuration used.

CHAPTER 4

Implementation

In this chapter we present the hardware and software components that we used to develop and test the algorithm. We mention the libraries and if applicable discuss certain parameters that influence the library functions' behaviors. We also give further details about how we used the GPU to perform calculations, especially in the K-Means and the specularity estimation steps. Later in this chapter, we present further details about our novel K-Means implementation, show the running times on different numbers of clusters and compare it to the performance of OpenCV [22].

4.1 Hardware Configuration

In Table 4.1 you find a summary of the hardware configuration. As already mentioned several times in this thesis, the Kinect sensor is used for capturing depth and video data. Additionally it is used as video see-through camera in the mixed reality application. The uEye camera is mounted with a Fujinon YV2.2x1.4A-2 fish-eye lens and is used as environment camera.

CPU	Intel Core i7-2600K @ 3.4 GHz
GPU	Nvidia GTX 580 3 GB
RAM	8 GB DDR3-1600
Camera 1	Microsoft Kinect sensor
Camera 2	uEye UI-1645LE-C-HQ

Table 4.1: The hardware configuration of the test system.

4.2 Software Configuration

We integrated our BRDF estimation algorithm into the framework that was developed for Differential Instant Radiosity. We used Microsoft Visual Studio 2008 and developed in C#. In order to access the graphics card we used DirectX 10 together with the SlimDX library. The operating system was Microsoft Windows 7 64-bit and we also compiled for a 64-bit target platform.

4.2.1 How GPU Implementations were done

Whenever we talk about that we made an implementation on the GPU, it always means that we wrote HLSL shader code to perform the operations. Since all the data we receive from the Kinect sensor is in form of 640×480 images, we almost always proceeded in the following way:

1. Set the view port to 640×480.
2. Set all the uniform shader input variables, e.g. the color image or light positions.
3. Render a screen quad, i.e. a quad containing 4 vertices in the range $[-1, 1]$ and 4 texture coordinates in the range $[0, 1]$.
4. In the vertex shader simply pass the data to the pixel shader.
5. In the pixel shader do the specific operations.
6. Render the result into a texture and if needed create MIP maps.

Every time MIP maps were used to calculate the average or the sum of a texture, like we do in our K-Means implementation, the render target is a texture of size 1024×1024 in order to have a squared texture of size $2^n \times 2^n$, where the original 640×480 fits completely. The view port is still set to 640×480 so that we do not make any additional unnecessary calculations.

4.2.2 The Kinect Sensor in a 64-bit Windows Environment

The Kinect sensor was originally developed for the Microsoft XBox360 and designed to be a novel game controller. As described in Section 3.4, the Kinect combines a depth and an RGB camera to recognize human motion to control an Xbox game. However, due to its attractiveness in both quality and price, many computer graphics and computer vision researchers soon began to experiment with the Kinect in tasks for which it was not originally designed. So did we in order to investigate in real-time BRDF estimation.

Since no official Microsoft driver was available at the time the research started (February 2011), we used the OpenNI [42] driver. More specifically, we used the latest unstable branch [41]. In addition to OpenNI we also used avin2's driver [1]. OpenNI already comes with C# support, which was very useful for our framework and hence we could directly integrate it. Unfortunately we cannot use the Microsoft Kinect SDK [36] at the moment, because it currently does not support x64 target platforms.

To communicate with the uEye camera, which serves as environment camera, we simply used the original driver [21]. It also comes with a C# API and hence allowed easy integration. Since the two cameras need to be aligned in a common world reference frame and in order to geometrically register the virtual content in the scene some form of tracking of the video see-through camera needs to be done. We used Studierstube Tracker [16] for this task.

4.2.3 Normal Estimation using PCL

Normal Estimation of the point clouds is done by using the *Point Cloud Library* (PCL) [47]. Since PCL is a C++ library, we could not directly use it in our framework. Therefore we wrote a wrapper in C++/CLI to access its functionality. In version 1.0 PCL offers two different normal estimation methods. We first tested the standard normal estimation

`pcl::NormalEstimation`,

which computes the normals by using a least-squares plane fitting approach. Unfortunately, with this method we were far away from interactive frame rates and an e-mail from the author, Radu Bogdan Rusu, confirmed our assumption that it is meant to work offline as it uses exact covariance computation schemes.

PCL also offers another normal estimation feature called

`pcl::IntegralImageNormalEstimation`,

which we will abbreviate by IINE in the following. It works on structured point clouds that come in form of an image, i.e. where the pixel position corresponds with the projection of the 3D point onto the image plane. This constraint is then used to speed up the calculations. Unfortunately the paper of the developer of IINE, Stefan Holzer, was not published then (as of July 2011) and hence we cannot give further details about this method.

In IINE there is especially one parameter, which influences the result the most: The smoothing size s. The smoothing size is the size of the area used to smooth the normals [46]. To demonstrate the effect of the smoothing size on the normals we generated a series of images

with different smoothing sizes, shown in Figure 4.1. We can see that the bigger the smoothing size, the less noisy the normals get. The drawbacks however are that the effective size of the normal map shrinks and that edges will also get smoothed out. The size shrinking is only visible at higher smoothing sizes, since the effective size of the depth map is smaller than the one of the color map (refer to Figure 3.3) and all the pixels having a color but not depth information will simply be set to the zero normal. Note that in Figure 4.1 all the images were cropped for visualization reasons.

4.2.4 Specularity Estimation using DotNumerics

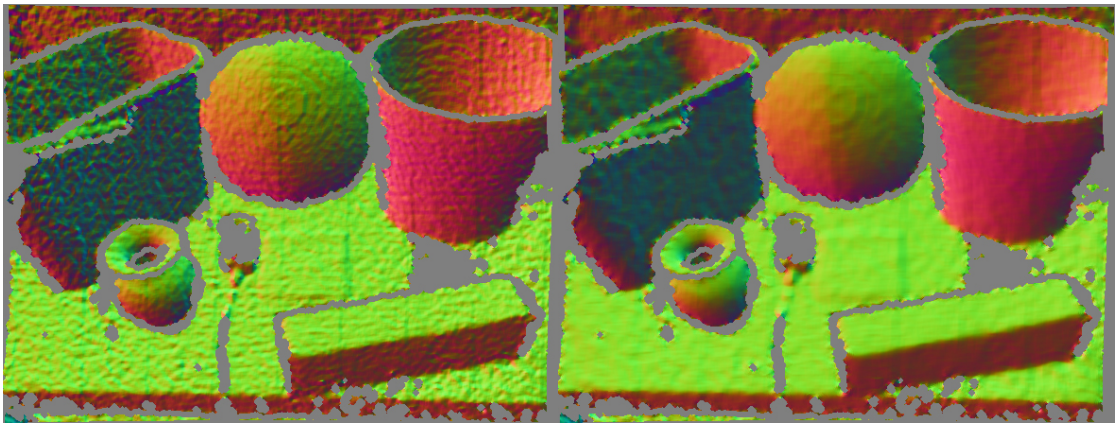
The specularity estimation described in Section 3.10 is done by using non-linear least squares. We used DotNumerics [25] for the optimization. More specifically, we used the Nelder-Mead Simplex implementation. The optimization methods of DotNumerics all work in a way that you specify a certain function that is to be minimized. You also specify the function's variables and provide an initial solution. The optimization method then minimizes the function by searching for the *best* parameters.

Remember that we estimate common specular parameters for every cluster. For every pixel, we calculate the squared difference between the color image RGB values and the result of the rendering equation using the current guess of the specular parameters and sum over all those differences, see Equation 3.14. Hence, the optimization involves a lot of computations, since the sum of squared differences has to be calculated for each guess of the parameters and for each cluster.

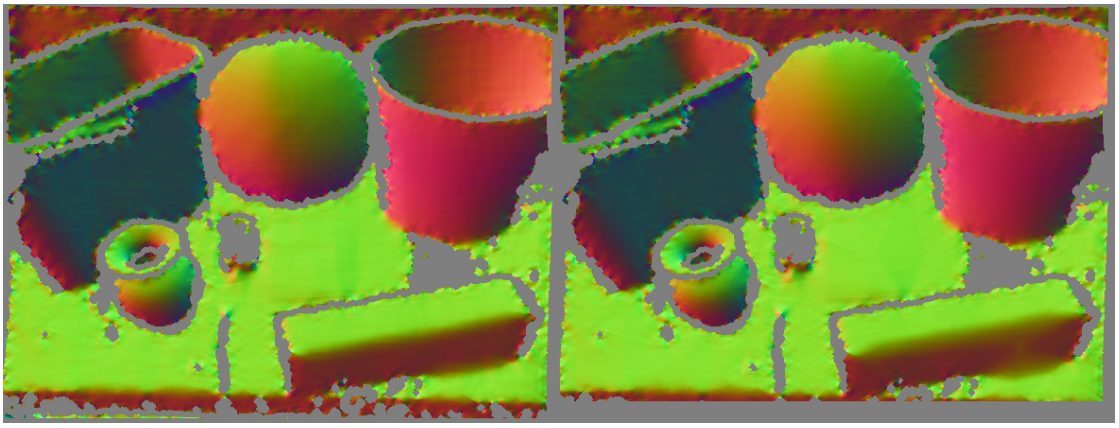
A convenient feature of DotNumerics is that the function to be optimized can contain arbitrary user defined code as long as the signature meets the requirements. Thus, we implemented the sum of squares calculation on the GPU as described in Section 4.2.1. We calculate the squared distance for every individual pixel of the current cluster, write the results into a texture and obtain the sum by using MIP maps, similarly as we do for our K-Means algorithm.

The estimation time for 6 clusters, allowing each cluster optimization up to 100 iterations is around 300 milliseconds. Comparing our specularity estimation with the one of Zheng et al. [75] we observe an enormous speed-up since they need 22 to 36 minutes. It should be mentioned here that they proceed slightly different because they have multiple observations per vertex. They have a total of 36 images of the object as they make one image every 10 degrees, so in average they will have 18 samples per vertex some of which will not be used because they are shadowed since the camera and the light source are at different angles. As Zheng et al. use between 20K and 40K vertices to represent an object, the total number of samples is approximately similar to the 300K samples that we use and the runtime is thus comparable apart from CPU differences.

(a) Color image (b) Depth map



(c) Normal map, $s = 2 \times 2$ (d) Normal map, $s = 5 \times 5$



(e) Normal map, $s = 12 \times 12$ (f) Normal map, $s = 30 \times 30$

Figure 4.1: Test of PCL's `IntegralImageNormalEstimation` for different smoothing sizes s.

4.3 K-Means

We already provided a detailed description of our K-Means implementation in Section 3.9.1. Here we want to present the running times of our clustering algorithm with different numbers of clusters and compare our results with the implementation in OpenCV.

OpenCV [22] is a high-performance computer vision library. Therefore a search for a high-performance robust K-Means implementation naturally leads to a more thorough investigation of this library. Additionally to the classic OpenCV library, which only uses the CPU for processing, there also exists OpenCV_GPU [23], which contains a subset of the functions implemented in OpenCV using NVidia CUDA [40] to do the calculations on the GPU. Since OpenCV_GPU does not provide a GPU implementation of K-Means (as of July 2011), we evaluated the CPU version.

4.3.1 Results and Comparison with OpenCV

A robust K-Means implementation requires multiple iterations. In order to test the performance of OpenCV and compare it to our K-Means implementation, we decided to run the algorithm 5 times, each time with different random initial cluster centers, since this is also the strategy we use in our material estimation technique. Each run consists of 20 iterations. We generated 307200 3-component vectors randomly as test data. We only measured the time for the actual K-Means algorithm and did not include the time for data generation and preparation. Using 12 clusters the OpenCV implementation needs around 4 seconds to complete on our test system described above. Obviously this is not fast enough for interactive frame-rates.

In Table 4.2 a more detailed list of the running times of the OpenCV and our implementation is given. Additionally to the implementation described in Chapter 3, we implemented a hybrid CPU/GPU technique where only the Euclidean distance calculations to determine the nearest cluster center for every data point is done on the GPU and the cluster center updates are done using the CPU. Since many GPU implementations of K-Means in the literature are based on a hybrid CPU/GPU approach, we decided that it is very useful for performance comparison to also have the running time of such a technique.

We tested the clustering algorithms on 6 and 12 clusters respectively. Our GPU implementation is almost 12 times faster than OpenCV when using 6 clusters and more than 10 times faster when using 12 clusters. Although the hybrid CPU/GPU implementation already performs 3 times faster than OpenCV when using 12 clusters, having a processing time of over one second is still not satisfying.

Clusters	OpenCV	Hybrid CPU/GPU	GPU-only using MIP maps
6	2.2227	1.3175	0.1875
12	3.9344	1.3457	0.3634

Table 4.2: Comparison of the running time in seconds averaged over 5 trials of the OpenCV K-Means implementation, the hybrid CPU/GPU approach and our novel GPU-only technique based on MIP maps. The test system was the one listed in Table 4.1. The input were 307200 3-dimensional data points (a 640×480 RGB image) and the algorithm ran for 5×20 iterations.

4.3.2 Structure of the Render Targets in the GPU Implementation

From Table 4.2 we can also observe that the processing time of our GPU implementation for 12 clusters is approximately twice as high as for 6 clusters, while the one of the CPU/GPU implementation seems to grow very slowly with respect to the number of clusters. As described in Chapter 3, we need to write the data points of every class into a separate texture in order to calculate the new cluster centers correctly. Current graphics hardware supports up to 8 render targets simultaneously. In order to maximize the number of clusters that can be processed in one shader pass we use 8 RGBA floating point textures and the following texture layout: The R-, G- and B-channel of the first 6 textures hold the 3-dimensional data points. Hence we can process 6 clusters in one pass. The A-channel of the first 6 textures is used to calculate the number of data points per class, which is needed to calculate the centroid, see Section 3.9. The remaining 2 textures are used to calculate the variance. The total variance of one cluster is the average value of all the distances between its data points and its cluster center. Thus, we need 6 channels to store these distances for 6 classes. Hence, if e.g. 12 clusters are required we need 2 pixel shader passes, which is why the processing time approximately doubles every 6 clusters in our K-Means implementation.

In the hybrid CPU/GPU implementation only the Euclidean distances and the nearest cluster centers are calculated on the GPU and the centroid and the variance are calculated using the CPU. The distance calculations to the k cluster centers for every data point are done in a loop in the pixel shader. Since the running time of the CPU/GPU approach is dominated by the CPU, the additional calculations on the GPU are almost negligible.

4.3.3 Final Thoughts

We want to point that in fact the performance comparison is not exactly fair and in favor of OpenCV. In our implementation, we simultaneously calculate the average total variance of each run in order to choose the run, which exhibits the lowest variance, while in the OpenCV implementation only the clusters are computed. Additionally, to measure the performance of our

implementation we started the timer before the first DirectX API call and stopped it after the result was copied into the RAM to ensure that the calculation terminated.

The low number of clusters, where our novel K-Means implementation is feasible may pose restrictions to certain applications. However, in our case it is very unlikely that we want an estimation of more than 12 materials at the same time, since indirect illumination only makes sense if the material has a certain size on the image plane and with a maximum resolution of 640×480 this is hardly achievable having more materials. Another point to consider is that the video memory of the graphics card is continuously getting bigger and the graphics card we used already has 3 GB. If the number of simultaneous render targets gets bigger too, more clusters can be used without much additional computational cost.

CHAPTER 5

Results

In this chapter we want to present the final results of our technique. Since the motivation of this thesis was an interactive BRDF estimation method for mixed reality systems, we want to examine its use in such a system. More specifically, we will use the material estimation in our Differential Instant Radiosity renderer, where previously geometry and materials had to be manually created as a pre-processing step. Especially, we want to show the following two effects, which rely on material and geometry information of the real environment:

- Color bleeding - visible indirect illumination from real to virtual objects and vice-versa.
- Occlusion - a real objects occluding a virtual one.

A virtual object occluding a real object is not very exciting, since in typical mixed reality scenarios the virtual content is simply drawn on top of the original image and hence the virtual objects always occlude real ones. In the following we will first discuss errors and quality concerns and then present two scenarios, where the aforementioned effects will be visible.

Additionally, we will present the running time of our estimation and compare it with the running times of the techniques reviewed in Chapter 2.

5.1 Error Measurements

Our technique relies on concepts that had been demonstrated in the past to produce good results and we modified these concepts to work interactively. Since the basis of our estimation process is the same as the one from Zheng et al. [75], we assume that our approach performs similarly in terms of the relative error between ground truth data and the estimation, see the next section.

Error measurements turned out to be particularly problematic because on the one hand, we would need an object where the ground truth parameters are known for the Phong model, which is going to be very hard since it does not have any physical foundation. And on the other hand, using the OpenNI [42] driver it is neither possible to adjust the image brightness nor the white balancing and both is done automatically and we do not know the algorithms behind these operations. Especially the second point demonstrates that any error measurement would have very low significance since the size and the strength of a highlight highly depends on the image brightness which can heavily vary in low dynamic range images. In addition, the motivation of our technique is not to have material parameters for photo-realistic rendering, but to incorporate indirect illumination in mixed reality environments and due to all these reasons we omit error measurements.

5.2 Quality Comparison

In this section we want to say a couple of words to the quality of our estimation with respect to the work of Zheng et al. [75]. As we did not have access to their code and did not have time to re-implement their algorithm and we additionally do not have the same test objects it is hard to compare the quality of the results. Nevertheless, we still want to mention the major differences and discuss their effects. Many of the major differences have already been discussed in Section 3.2. First of all is the geometry acquisition. As mentioned before, a problem of our approach lies in an incomplete depth map. Since there is a lot of research going on in reconstruction using the Kinect sensor, we strongly assume that this will not be a problem in the future any more and thus the geometry will be at least as good as the geometry acquired by Shape From Silhouette.

Light estimation also works differently. Zheng et al. estimate the lighting as a manual preprocessing step using a shiny steel ball of known size and taking pictures with fixed aperture and exposure time. In contrast, we estimate the lighting dynamically at run-time using the image stream of a fish-eye lens camera. Hence, the light estimation will be less accurate than the one of Zheng et al.

Although we modified the highlight removal from its original version, we did not discover any loss in quality. Highlight removal simply serves for the clustering and if an object's pixels are clustered together correctly, the inpainted regions where the highlight was will also be clustered together with them because the neighboring pixel values are used in the filling process. Another difference lies in the illumination model used, but as discussed in Section 2.2.3 this is negligible.

The biggest quality concern in our opinion is that Zheng et al. use multiple samples per vertex and hence have more data in the fitting process, which makes the optimization more robust. Since they turn the object around on a turntable while taking pictures, the highlights also

move and they have thus some samples of the same vertex that are specular and some that are not. Due to the controlled setup they also do not use any tracking of the camera which further reduces errors.

5.3 Scenario 1

The first scenario we present shows a plant in a yellow pot and a beige decoration vase. Again we use Studierstube tracker to put the environment camera and the Kinect camera into a world coordinate system and also to provide a location, where the virtual content is placed. The virtual object is the Stanford Buddha, which has a light gray color tone. In Figure 5.1a you can see the result of the Differential Instant Radiosity renderer using our interactive geometry and BRDF estimation technique.

The only physical light source is one 250 Watts daylight bulb. The lighting environment is captured by a fish-eye lens camera. In all of the following examples the number of clusters, k, was manually set to 6. In Figure 5.1b and 5.1c you can see the original color and depth image respectively. In addition to the marker that is placed on the red carpet to define the origin of the world coordinate system, we use a second marker to track a pocket lamp. The real pocket lamp is turned off as can be seen in the original image. It is tracked and used as an interaction device to illuminate the scene with pure virtual light. On the resulting image we can see a spot on the yellow plant pot cast from the pocket lamp. This spot causes indirect illumination, which is visible as yellow color bleeding on the Buddha, on the beige vase and on the floor.

In Figure 5.2 a similar image is shown, although this time the pocket lamp points towards the red carpet on the floor. This causes red color bleedings on the Buddha and the real objects. Figure 5.2b and 5.2c again show the original color and depth image. In both images the black boundaries can be observed, which were already discussed several times in this thesis. Although we could have used the original color image to minimize the effect, occlusion, which is demonstrated in the next section, will look terribly wrong then and hence we consistently left it black.

5.4 Scenario 2

The second scenario we present is similar to the first one although here we want to focus on occlusion. In Figure 5.3a we see a virtual object, the Stanford Dragon in green color. We can see that it is partially occluded by a real object. This time the pocket lamp is also virtually turned off to demonstrate the difference to the previous images.

Scenario	*k*	K-Means iter.	Spec. iter.	*s*	Running time
Scenario from Chapter 3	6	5 runs à 20 iter.	100	35	0.5378
Image from Figure 5.1	6	3 runs à 20 iter.	60	15	0.3799
Image from Figure 5.2	6	3 runs à 20 iter.	60	15	0.3876
Image from Figure 5.3	6	3 runs à 20 iter.	60	15	0.3764

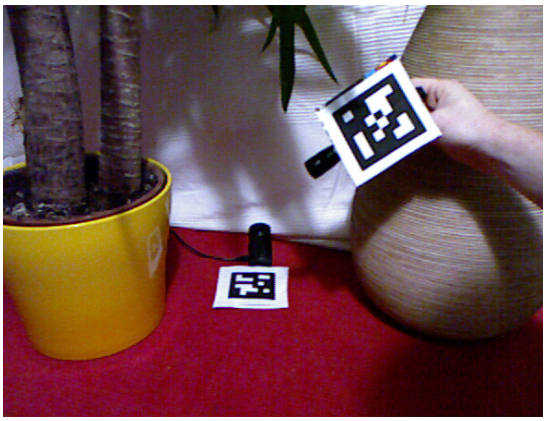
Table 5.1: The table above shows the parameters used in our tests consisting of: The number of clusters k, the number of K-Means iterations, the number of iterations in the specularity estimation step per class and the normal smoothing size s. In the last column the running time of the total estimation is given. For the test scenario from Chapter 3 we allowed more iterations to get more accurate results.

In this scenario we can clearly see the problem of missing depth values at the border of objects and its consequences. The silhouette of the beige vase contains many holes and seams and does not resemble its original silhouette from the color image. Therefore also the occlusion does not happen at a smooth edge. As described in Section 3.4.1 the work on Kinect Fusion [35] promises to solve this problem.

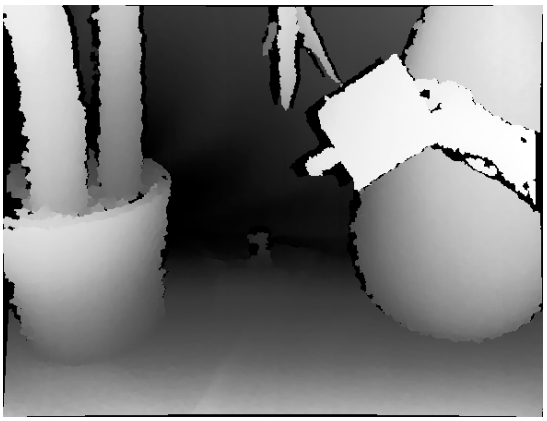
5.5 Performance

Finally, in Table 5.1 we give details about the parameters used in the estimations of the scenarios in this chapter as well as of the test scenario from Chapter 3 and present the running times. The time for the estimations was between 0.37 and 0.54 seconds and our approach comes thus very close to interactive frame rates. Comparing our results with the results of the methods in the literature presented in Table 2.1 from Section 2.3.3 we can clearly see the huge performance gain. The fastest approach of the techniques we reviewed was the one of Zheng et al. [75] with a running time of at least 22 minutes. Hence, with our technique we are around 3000 times faster.

(a) Result of the Differential Instant Radiosity renderer using the estimated geometry and materials.



(b) Original color image

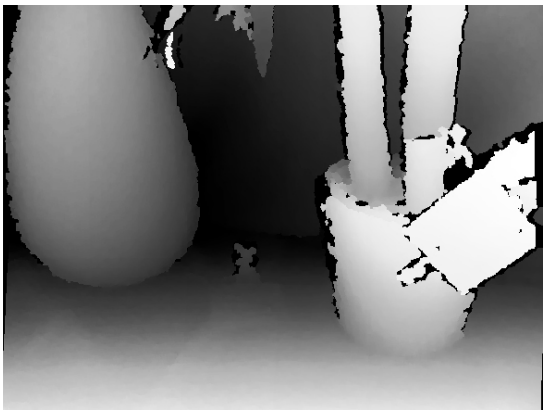


(c) Original depth image

Figure 5.1: A pocket lamp points towards a real object causing yellow color bleeding on both virtual and other real objects. The image was rendered at 2.2 frames per second.

(a) Result of the Differential Instant Radiosity renderer using the estimated geometry and materials.

(b) Original color image

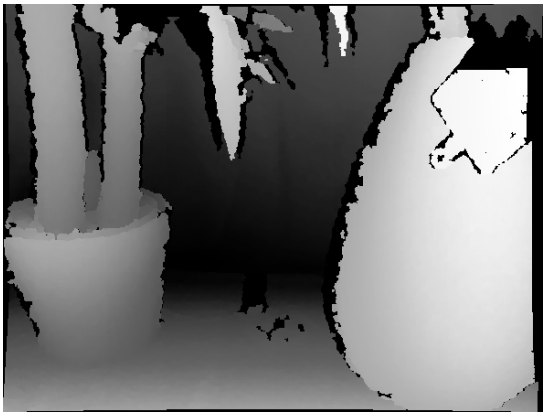


(c) Original depth image

Figure 5.2: The pocket lamp points toward the red carpet causing red color bleeding on virtual and real objects. The image was rendered at 2.2 frames per second.

(a) Estimated parameters rendered using Differential Instant Radiosity.

(b) Original color image



(c) Original depth image

Figure 5.3: A virtual object, the Standford dragon, is partially occluded by a real one. This time the pocket lamp is turned off. The image was rendered at 2.2 frames per second.

CHAPTER 6

Conclusion

To conclude this thesis, in Section 6.1 we summarize our work, emphasize the biggest achievements and compare it to related work, which was presented in Chapter 2. Finally, in Section 6.2 we discuss future work.

6.1 Summary and Comparison with Related Work

In this thesis an interactive image-based BRDF and geometry estimation algorithm for the use in mixed reality environments was presented. Until now, all material estimation techniques in the literature work offline, having processing times ranging from half an hour up to several hours. Our technique relies on concepts that have been shown very useful for surface reflectance estimation in the past and adapts and modifies these concepts to work interactively using the power of concurrency provided by current graphics cards. Hence, the major contribution of this work is a massive speed-up, reducing the processing time to under a second per frame.

Due to the fact that we achieve interactive frame rates, we are able to handle dynamically changing scenes without the need of any pre-processing. This means that e.g. new objects may be inserted into the scene or old objects may be removed while the system still manages to hold an up-to-date data model of the scene's geometry, its reflectance characteristics and its lighting environment. This is a major advancement for the use in mixed reality systems, since in such systems physical interaction with real objects is one of the major aspects in contrast to purely virtual systems and due to this interaction the scene gets almost always modified.

Amongst other things, this speed-up was made possible due to a novel GPU implementation of the K-Means clustering algorithm, which is done entirely on the GPU. To our knowledge only one other GPU only implementation of K-Means exists in the literature, which is the one of

Dhanasekaran and Rubin [11]. We use MIP maps to calculate the cluster center updates instead of using the CPU as in many K-Means GPU implementations and hence we do not need to copy data between the video memory and the RAM. We have shown that our GPU implementation outperforms conventional hybrid CPU/GPU implementations, when the number of clusters is relatively low and it is thus an optimal solution for interactive BRDF estimation.

6.2 Future Work

There is plenty of room for future work in our approach. One is definitely additional speed-up. Especially the parts, where we used libraries can most likely be enhanced, like the normal estimation, which is done by using PCL. Since the normal estimation feature of the PCL that we used runs on the CPU, the processing time is not optimal and a GPU implementation would definitely increase the speed. To further increase the performance it is possible to exploit temporal coherence in the clustering by saving the cluster centers from the first frame and by using them as start values for the clusterings in the following frames.

Apart from the running time, the light estimation can be enhanced. Currently we are using the same light estimation as we use in Differential Instant Radiosity, i.e. we assign VPLs on an environment map using Halton sequences. We achieved good results when using one point light source, however multiple point light sources may be problematic. As mentioned in Section 3.12, the light estimation is not optimal for BRDF estimation, since it is not very accurate and it is by no means guaranteed that the VPLs sufficiently approximate the true light sources. One approach to improve it could be to cluster all the VPLs and to detect the true light sources as clusters centers.

Finally another big improvement will be some form of depth map hole filling. Especially at the border of objects the Kinect has problems to calculate depth values. This is however problematic, since then we cannot recover any information at those points, which leads to strong seams, noticeable as black regions in the resulting image. There are several ways to deal with this problem. Either one could try to fill the holes on a per-image basis using an image inpainting approach or one could exploit temporal coherence and use the information from the previous frames to fill the missing values.

With our interactive BRDF and geometry estimation technique we hope to open up new possibilities and to come one step closer to achieve believable renderings in mixed reality environments without the need of pre-modeling.

List of Figures

1.1 Left: A virtual object, the teapot, rendered into a real scene. Right: The outcome we would expect in the real world. 2

2.1 (a) Virtual object shadows a real one. (b) A pocket lamp points towards a virtual Cornell box causing red color bleeding through indirect illumination towards the desk and the cardboard box. (c) A pocket lamp illuminates the real cardboard box. Indirect illumination via VPLs causes the green wall of the Cornell box to appear brighter. Figures taken from [28] 7

2.2 The geometry of the incident ray ω_i, defined by the angles (θ_i, ϕ_i), and the reflected ray ω_r,defined by the angles (θ_r, ϕ_r). The $\mathbf{Z}$-axis corresponds with the normal at the surface point. Figure courtesy of Nicodemus et al. [39]. 8

2.3 Acquisition system of Mercier et al. [34]. Figure also courtesy of Mercier et al. [34]. 13

2.4 Highlight removal by Tan and Ikeuchi [62]. Figures also courtesy of Tan and Ikeuchi [62]. 18

2.5 Morphological Dilation and Erosion. 20

2.6 Demonstration of Morphological Reconstruction. Original image (a), the mask G. Initial image (b), the marker image F. Note that the blue borders are added just for visualization purposes, the true marker image only consists of the two black dots! Intermediate result (c), after a couple of iterations. The blue borders are again added for visualization. The final reconstructed image (d). 22

3.1 This figure gives an overview of the algorithm and shows the individual steps. . . . 32

3.2 Scenario used to test the individual components of our technique. Different objects with different materials can be observed. 33

3.3 Color and depth data from the Kinect sensor. 35

3.4 The perspective camera model. Point $\mathbf{P}$ gets projected onto the image plane e. . . . 36

3.5 Central projection from top-down-view, showing the relationship between the focal length, the image width and the field of view. Figure taken from [60]. 36
3.6 Visualization of the estimated normals in our test scenario. 38
3.7 Test of the correctness of the normal estimation. 38
3.8 The environment map obtained from the test scenario streamed by a fish-eye lens camera. 39
3.9 Demonstration of the order in which the neighboring pixels are processed to find a diffuse pixel. The current specular pixel, which is in the middle in this figure, gets replaced by the first non-specular pixel found, marked by the **x**. 42
3.10 Comparison of the highlight removal of Ortiz [44] and our modification. 42
3.11 From top the bottom: The color image, the specularity mask and the resulting image with removed highlights of (a) a test scenario containing juggling balls and (b) the test scenario used throughout this chapter. 43
3.12 Results of the inverse diffuse shading stage operating on the highlight removed image. Note the smoother color distribution along the objects, which facilitates the clustering. 45
3.13 Inverse diffuse shading in a scene with two light sources. (a) shows the color image and (b) the resulting inverse diffuse shaded image. 45
3.14 Visualization of the K-Means clustering. 50
3.15 Results of the material estimation. 53
3.16 Phong rendering of the point cloud using the estimated parameters (a), and the original color image (b). Note that the reflections on the yellow plant pot are automatically baked into the diffuse texture. 54

4.1 Test of PCL's `IntegralImageNormalEstimation` for different smoothing sizes s. 61

5.1 A pocket lamp points towards a real object causing yellow color bleeding on both virtual and other real objects. The image was rendered at 2.2 frames per second. . . 69
5.2 The pocket lamp points toward the red carpet causing red color bleeding on virtual and real objects. The image was rendered at 2.2 frames per second. 70
5.3 A virtual object, the Standford dragon, is partially occluded by a real one. This time the pocket lamp is turned off. The image was rendered at 2.2 frames per second. . . 71

List of Tables

2.1 Overview of the running times of some of the algorithms presented in Section 2.3.1. 17

4.1 The hardware configuration of the test system. 57

4.2 Comparison of the running time in seconds averaged over 5 trials of the OpenCV K-Means implementation, the hybrid CPU/GPU approach and our novel GPU-only technique based on MIP maps. The test system was the one listed in Table 4.1. The input were 307200 3-dimensional data points (a 640×480 RGB image) and the algorithm ran for 5×20 iterations. 63

5.1 The table above shows the parameters used in our tests consisting of: The number of clusters k, the number of K-Means iterations, the number of iterations in the specularity estimation step per class and the normal smoothing size s. In the last column the running time of the total estimation is given. For the test scenario from Chapter 3 we allowed more iterations to get more accurate results. 68

Bibliography

[1] Avin2. https://github.com/avin2/SensorKinect. Accessed: 2011-07-22.

[2] R. Azuma. A survey of augmented reality. *Presence*, 6(4):355–385, 1997.

[3] J. F. Blinn. Models of light reflection for computer synthesized pictures. In *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, SIGGRAPH'77, pages 192–198, New York, NY, USA, 1977. ACM.

[4] S. Boivin and A. 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*, SIGGRAPH'01, pages 107–116, 2001.

[5] G. Bradski and A.Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 1. edition, 2008.

[6] C. Buchart, D. Borro, and A. Amundarain. Gpu local triangulation: an interpolating surface reconstruction algorithm. *Computer Graphics Forum*, 27:807–814, 2008.

[7] Computer Graphics Group. http://www.cg.tuwien.ac.at/projects/RESHADE. Accessed: 2011-03-24.

[8] Computer Graphics Group. http://www.cg.tuwien.ac.at/research/publications/2010/knecht_martin_2010_DIR/. Accessed: 2011-07-5.

[9] R. L. Cook and K. E. Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics*, 1:7–24, January 1982.

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

[11] B. Dhanasekaran and N. 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*, GPGPU-4, pages 2:1–2:8, New York, NY, USA, 2011. ACM.

[12] Z. Dong, T. Grosch, T. Ritschel, J. Kautz, and H.P. Seidel. Real-time indirect illumination with clustered visibility. In *Proceedings of the international workshop on Vision, Modeling and Visualization*, VMV'09, 2009.

[13] A. Fournier, A. S. Gunawan, and C. Romanzin. Common illumination between real and computer generated scenes. In *Proceedings of Graphics Interface '93*, pages 254–262, Toronto, ON, Canada, May 1993.

[14] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Prentice Hall, 2. edition, 2002.

[15] R. C. Gonzalez, R. E. Woods, and S. L. Eddins. *Digital Image Processing Using MATLAB*. Prentice Hall, 2003.

[16] Graz University of Technology. `http://studierstube.icg.tugraz.at/handheld_ar/stbtracker.php`. Accessed: 2011-01-07.

[17] X. D. He, K. E. Torrance, F. X. Sillion, and D. P. Greenberg. A comprehensive physical model for light reflection. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, SIGGRAPH'91, pages 175–186, New York, NY, USA, 1991. ACM.

[18] D. Hearn and P. Baker. *Computer Graphics with OpenGL*. Pearson Prentice Hall, 3. edition, 2004.

[19] B. Hong-tao, H. Li-li, O. Dan-tong, L. Zhan-shan, and L. He. K-means on commodity gpus with cuda. In *Proceedings of the 2009 WRI World Congress on Computer Science and Information Engineering - Volume 03*, CSIE'09, pages 651–655, Washington, DC, USA, 2009. IEEE Computer Society.

[20] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, SIGGRAPH'92, pages 71–78, New York, NY, USA, 1992.

[21] IDS Imaging Development Systems GmbH. http://www.ids-imaging.de/drivers.php?cat=2. Accessed: 2011-07-22.

[22] Intel and Willow Garage. http://opencv.willowgarage.com/wiki/. Accessed: 2011-05-02.

[23] Intel and Willow Garage. http://opencv.willowgarage.com/wiki/OpenCV_GPU. Accessed: 2011-07-23.

[24] T. Jie, W. Gangshan, X. Bo, and G. Zhongliang. Interective point clouds fairing on many-core system. In *Proceedings of the International Symposium on Parallel and Distributed Processing with Applications*, ISPA'10, pages 557–562, 2010.

[25] José Antonio De Santiago-Castillo. http://www.dotnumerics.com/. Accessed: 2011-07-23.

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

[27] Klaas Klasing, Daniel Althoff, Dirk Wollherr, and Martin Buss. Comparison of surface normal estimation methods for range sensing applications. In *Proceedings of the IEEE international conference on Robotics and Automation*, ICRA'09, pages 1977–1982, Piscataway, NJ, USA, 2009.

[28] M. Knecht, C. Traxler, O. Mattausch, W. Purgathofer, and M. Wimmer. Differential instant radiosity for mixed reality. In *Proceedings of the 9th IEEE International Symposium on Mixed and Augmented Reality*, ISMAR'10, 2010.

[29] P. Kühtreiber. Brdf approximation and estimation for augmented reality. Technical report, Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2010.

[30] Y. Li, K. Zhao, X. Chu, and J. Liu. Speeding up k-means algorithm by gpus. In *Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology*, CIT'10, pages 115–122, Washington, DC, USA, 2010. IEEE Computer Society.

[31] S. Lin and H. Y. Shum. Separation of diffuse and specular reflection in color images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'01, pages 341–346, Los Alamitos, CA, USA, 2001. IEEE Computer Society.

[32] S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982.

[33] J. Lu and J. Little. Reflectance function estimation and shape recovery from image sequence of a rotating object. In *Proceedings of the 5th International Conference on Computer Vision*, pages 80–86, June 1995.

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

[35] Microsoft. `http://research.microsoft.com/en-us/projects/surfacerecon/`. Accessed: 2011-10-24.

[36] Microsoft. `http://research.microsoft.com/en-us/um/redmond/projects/kinectsdk/`. Accessed: 2011-07-22.

[37] S. K. Nayar, X. S. Fang, and T. Boult. Separation of reflection components using color and polarization. *International Journal of Computer Vision*, 21:163–186, February 1997.

[38] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

[39] F. Nicodemus, J. Richmond, J. Hsia, I. Ginsberg, and T. Limperis. Geometrical Considerations and Nomenclature for Reflectance. Washington. *National Bureau of Standards*, 1977.

[40] NVidia. `http://developer.nvidia.com/category/zone/cuda-zone`. Accessed: 2011-05-02.

[41] OpenNI. `https://github.com/OpenNI/OpenNI/tree/unstable`. Accessed: 2011-07-22.

[42] OpenNI. `http://www.openni.org/`. Accessed: 2011-07-22.

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

[44] F. Ortiz. Real-time elimination of brightness in color images by ms diagram and mathematical morphology. In *Proceedings of the international workshop on Computer Analysis of Images and Patterns*, CAIP'07, pages 458–465, 2007.

[45] F. Ortiz and F. 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, 2006.

[46] PCL community. `http://docs.pointclouds.org/trunk/classpcl_1_1_integral_image_normal_estimation.html`. Accessed: 2011-07-22.

[47] PCL community. `http://pointclouds.org/`. Accessed: 2011-07-22.

[48] B. T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18:311–317, June 1975.

[49] R. Ramamoorthi and P. Hanrahan. A signal-processing framework for inverse rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH'01, pages 117–128, New York, NY, USA, 2001. ACM.

[50] T. Ritschel, T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz. Imperfect shadow maps for efficient computation of indirect illumination. *ACM Transaction on Graphics*, 27:129:1–129:8, December 2008.

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

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

[53] T. Scheuermann and J. Hensley. Efficient histogram generation using scattering on gpus. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, I3D'07, pages 33–37, New York, NY, USA, 2007. ACM.

[54] R. Schlittgen. *Multivariate Statistik*. Oldenbourg Wissenschaftsverlag GmbH, 2009.

[55] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, 1982.

[56] J. Serra. *Image Analysis and Mathematical Morphology, Volume 2: Theoretical Advances*. Academic Press, 1988.

[57] S.A. Shafer. Using color to separate reflection components. *Color Research & Application*, 10:210–218, 1985.

[58] H. L. Shen and Q. Y. Cai. Simple and efficient method for specularity removal in an image. *Applied Optics*, 48(14):2711–2719, 2009.

[59] R. Szeliski. Rapid octree construction from image sequences. *CVGIP: Image Understanding*, 58:23–32, July 1993.

[60] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010.

[61] P. Tan, S. Lin, L. Quan, and H. Y. Shum. Highlight removal by illumination-constrained inpainting. In *Proceedings of the 9th IEEE International Conference on Computer Vision*, pages 164–169, 2003.

[62] R. T. Tan and K. Ikeuchi. Separating reflection components of textured surfaces using a single image. *IEEE Transactions on Pattern Analysis Machine Intelligence*, 27:178–193, February 2005.

[63] E. Trucco and A. Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall, 1998.

[64] G. J. Ward. Measuring and modeling anisotropic reflection. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, SIGGRAPH'92, pages 265–272, New York, NY, USA, 1992. ACM.

[65] G. J. Ward. The radiance lighting simulation and rendering system. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH'94, pages 459–472, New York, NY, USA, 1994. ACM.

[66] S.H. Westin, H. Li, and K.E. Torrance. A comparison of four brdf models. *Research Note PCG0402 Cornell University Program of Computer Graphics*, pages 1–10, 2004.

[67] Wikipedia Community. `http://de.wikipedia.org/wiki/Totale_Varianz`. Accessed: 2011-07-20.

[68] Wikipedia Community. `http://en.wikipedia.org/wiki/Kinect`. Accessed: 2011-07-26.

[69] E. Wu, Q. Sun, and X. Liu. Recovery of material under complex illumination conditions. In *Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, GRAPHITE'04, pages 39–45, New York, NY, USA, 2004. ACM.

[70] R. Wu, B. Zhang, and M. Hsu. Clustering billions of data points using gpus. In *Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop*, UCHPC-MAW'09, pages 1–6, New York, NY, USA, 2009. ACM.

[71] S. Xu and A. M. Wallace. Recovering surface reflectance and multiple light locations and intensities from image data. *Pattern Recognition Letters*, 29:1639–1647, August 2008.

[72] Q. Yang, S. Wang, and N. Ahuja. Real-time specular highlight removal using bilateral filtering. In *Proceedings of the 11th European conference on Computer vision: Part IV*, ECCV'10, pages 87–100, 2010.

[73] Y. Yu, P. Debevec, J. Malik, and T. 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*, SIGGRAPH'99, pages 215–224, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[74] M. Zechner and M. Granitzer. Accelerating k-means on the graphics processor via cuda. In *First International Conference on Intensive Applications and Services*, INTENSIVE'09, pages 7–15, 2009.

[75] Z. Zheng, L. Ma, Z. Li, and Z. Chen. Reconstruction of shape and reflectance properties based on visual hull. In *Proceedings of the 2009 Computer Graphics International Conference*, CGI'09, pages 29–38, New York, NY, USA, 2009. ACM.