Interactive BRDF Estimation for Mixed-Reality Applications

Martin Knecht
Vienna University of Technology
knecht@cg.tuwien.ac.at

Georg Tanzmeister
Vienna University of Technology
georg.tanzmeister@tuwien.ac.at

Christoph Traxler
VRVis Zentrum für Virtual Reality und Visualisierung Forschungs-GmbH
traxler@vrvis.at

Michael Wimmer
Vienna University of Technology
wimmer@cg.tuwien.ac.at

ABSTRACT

Recent methods in augmented reality allow simulating mutual light interactions between real and virtual objects. These methods are able to embed virtual objects in a more sophisticated way than previous methods. However, their main drawback is that they need a virtual representation of the real scene to be augmented in the form of geometry and material properties. In the past, this representation had to be modeled in advance, which is very time consuming and only allows for static scenes.
We propose a method that reconstructs the surrounding environment and estimates its Bidirectional Reflectance Distribution Function (BRDF) properties at runtime without any preprocessing. By using the Microsoft Kinect sensor and an optimized hybrid CPU & GPU-based BRDF estimation method, we are able to achieve interactive frame rates. The proposed method was integrated into a differential instant radiosity rendering system to demonstrate its feasibility.

Keywords

BRDF estimation, reconstruction, augmented reality

1 INTRODUCTION

Many mixed-reality applications require or at least desire a consistent shading between virtual and real objects. Examples are product presentations, virtual prototyping, architectural and urban visualizations and edutainment systems. Here virtual objects should smoothly blend into the real environment and provide a plausible illusion for users. They need to be rendered in a way that makes them hard to distinguish from real objects. Some recently published methods [14, 7] consider the mutual light interaction between real and virtual objects, so that they indirectly illuminate or shadow each other.

Beside the geometry of the scene and the real lighting conditions, the BRDFs of real objects are needed to simulate these mutual shading effects. Acquiring this data in a pre-processing step would diminish the dynamic and interactive nature of mixed-reality systems, and would also make it necessary to track the previously modeled movable real objects. In this paper, we introduce a BRDF estimation method that runs at interactive frame rates. It is based on real-time reconstruction using the structural light scanner provided by Microsoft's Kinect sensor [11]. The real lighting conditions are captured by a camera with a fish-eye lens from which light sources are derived.

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

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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

2 RELATED WORK

BRDF estimation has a long history of research and a variety of methods have been presented. Our approach belongs to the class of image-based methods, which are sometimes synonymously called *Inverse Rendering*. These methods try to fit parameters of an underlying, sometimes rather simple, BRDF model, like

the Phong [15] or Ward model [20], from images of a scene. Yu et al. introduced *Inverse Global Illumination* [23], where reflectance properties are derived from a sparse set of HDR images considering also indirect illumination. The geometry is pre-modeled and partitioned into surfaces with similar materials. The direct light sources must also be known. An optimization algorithm then calculates diffuse and specular components separately. Although the concept is sound and forms the basis of newer algorithms, it needs a lot of manual pre-processing. Sato et al. [17] presented a method that also performs a reconstruction of the object's geometry from range images, which is then used to estimate diffuse and specular parameters from the same images.

Boivin and Gagalowicz [2] use a single LDR image in addition to a geometric model including light sources. Starting with a Lambertian model, they iteratively compare renderings with the original image and consider more and more complex reflectance models as long as the difference is too large. Though their solution is scalable with regard to accuracy, it is still time consuming and requires pre-processing. Mercier et al. [10] were the first to present a fully automatic method to recover the shape and reflectance properties of a single object and the position of light sources from a set of calibrated images. For that purpose, the object and light sources are fixed on a turntable, and photographs are taken every 5 degrees. The geometry is approximated by *Shape From Silhouette* (SFS) from Szeliski [19]. The method is very accurate and does not need any preprocessing, but the special setup makes it unsuitable for mixed-reality. Xu and Wallace [22] used a depth sensor and a stereo intensity image to acquire an object's reflectance properties and parameters for multiple light sources. Although using a depth map comes close to our approach, their method is restricted to a single object. Furthermore, calculating light source parameters from intensity images introduces inaccuracies for flat surfaces.

Zheng et al. [25] presented a solution that is similar to that of Mercier et al. [10]. One big difference is that they use measured lighting conditions instead of deriving this information from the images, which minimizes the estimation error. They then apply the highlight removal algorithm from Ortiz and Torres [13] before clustering images into regions with similar diffuse materials using K-Means. The parameters of the Ward model are then obtained for each cluster by non-linear optimization. Their algorithm is very robust, since after estimating specular factors, diffuse factors are re-estimated in order to compensate for errors caused by wrong clustering or inaccurate geometry.

Like Mercier's method, the approach is based on a controlled setup, which does not meet our requirements. This especially concerns reconstruction by SFS and measurement of the light source. Their estimation pipeline however is very efficient and so we based our work on it. For example we also use an adaptation of the highlight removal technique from Ortiz and Torres [13] and we also use K-Means [9] for clustering.

Several efficient implementations of the K-Means algorithm on the GPU already exist. Almost all of them use a hybrid GPU/CPU approach, where the new cluster centers in each iteration are either entirely or at least partially calculated on the CPU [5, 8, 24, 21]. In all of the aforementioned papers CUDA is used to perform the calculations on the GPU.

To our knowledge there is only one method which was proposed by Dhanasekaran and Rubin [4] where the whole K-Means algorithm is done entirely on the GPU eliminating the need of continuously copying data via the PCIe bus. However, in contrast to Dhanasekaran and Rubin's work which relies on OpenCL, we use a different approach that utilizes mipmaps to calculate the center of each cluster using DirectX.

Generally speaking, all these previous image-based BRDF estimation methods work off-line and have running times ranging from a couple of minutes to several hours. Furthermore they are restricted to static scenes. Mixed-reality applications are highly interactive and dynamic according to Azuma's definition [1]. Hence our motivation was to design and develop a method that runs at interactive frame rates and can thus handle highly dynamic scenes.

3 OVERVIEW

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

Our BRDF estimation algorithm is mainly influenced by the ideas of Zheng et al. [25]. Their method was designed to work offline and had thus different requirements. As an adaption we modified their approach where necessary and made extensive use of the GPU

to gain interactive frame rates. The presented method can be divided into two main parts:

- Data acquisition: Capture data from the Microsoft Kinect sensor and a fish-eye lens camera to obtain color, geometry and lighting information.
- BRDF Estimation: Enhance the RGB input data and estimate diffuse and specular material characteristics.

Figure 1 illustrates the separate steps in the context of the two main parts of the method. Section 4 describes the data acquisition. Here normals are obtained with the use of the depth map we get from the Kinect and the lighting environment is approximated with the input stream of the fish-eye lens camera. The BRDF estimation is described in Section 5 where after intermediate steps the final diffuse and specular reflectance parameters are estimated. The output of our method is integrated into a DIR rendering system [7] and we directly render the fully defined geometry (including material characteristics) into a G-Buffer which stores the 3D position, the normal, the color and the material parameters needed for Phong shading.

4 DATA ACQUISITION

The Microsoft Kinect sensor is a relatively cheap device to capture a video and a depth stream simultaneously. The resolution of both streams is 640×480 pixels at a frame rate of 30Hz. Surrounding objects can be captured in a range between 0.45 and 10 meters. Figure 2 shows the input data provided by the Kinect sensor. The other source of input data is a IDS uEye camera with a fish-eye lens attached to capture the incident illumination.

4.1 Normal Estimation

We implemented two methods for normal estimation. The first one uses the Point Cloud Library (PCL) [16]. While the normals are of high quality, their computation takes too much time. The PCL functions need 223 milliseconds for one normal estimation step with a smoothing factor of 10. The estimation is performed on the CPU and therefore we implemented our own GPU normal estimation method that exploits temporal coherence (TC) between adjacent frames in a similar way as done by Scherzer et al. [18].

Our normal estimation is based on two render passes. The first pass performs subsampling and averaging of the normals from the previous frame. Furthermore, a curvature coefficient is calculated. The subsampling causes a smoothing on the normals of the previous frame. Let (i, j) be the row and the column of a given pixel in the previous frame. The average normal is

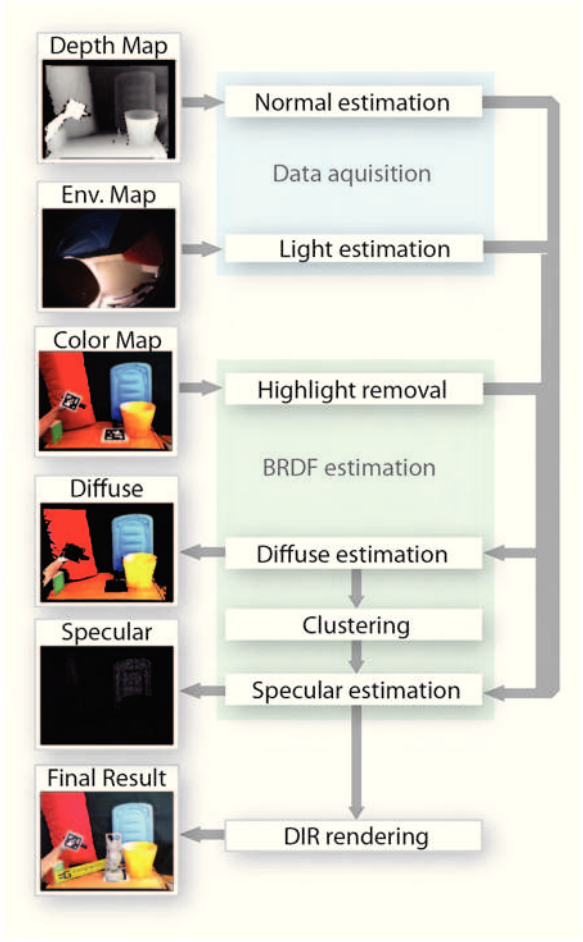

Figure 1: Shows the main steps in the BRDF estimation pipeline. Operations related to data acquisition are shown in the blue box (Section 4). Steps belonging to BRDF estimation and are shown in the green box (Section 5).

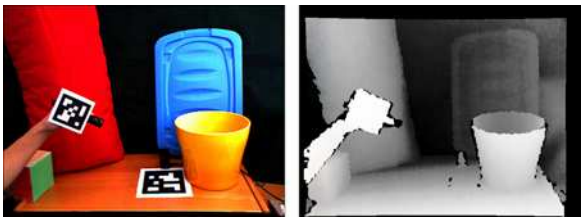

Figure 2: The left image shows the video input stream. The right image shows the normalized depth input stream. Both streams have a resolution of 640×480 with a frame rate of 30Hz.

then calculated by averaging over $(i-1, j)$, $(i+1, j)$, $(i, j-1)$ and $(i, j+1)$. Note that if no normal is available at a given pixel location, it will be discarded from the calculation. The curvature coefficient is calculated as follows:

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

$$curv_V(i,j) = N_{i,j-1} \cdot N_{i,j+1} \quad (2)$$

$$curv(i,j) = \min\left[curv_H(i,j), curv_V(i,j)\right]^{128} \quad (3)$$

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

The second rendering pass consists of two steps. In the first one a new normal is calculated from the point cloud delivered by the Microsoft Kinect sensor. We look up the 3D position $p_{i,j}$ at the current pixel (i,j) and two neighboring positions in horizontal $(i,j+4)$ and vertical $(i+4,j)$ direction. A distance value of four pixels showed good smoothing characteristics while edges were still preseverd. From these values, we can set up a surface normal as follows:

$$d_{i+4,j} = \frac{p_{i+4,j} - p_{i,j}}{|p_{i+4,j} - p_{i,j}|} \quad (4)$$

$$d_{i,j+4} = \frac{p_{i,j+4} - p_{i,j}}{|p_{i,j+4} - p_{i,j}|} \quad (5)$$

$$normal_{i,j} = d_{i+4,j} \times d_{i,j+4} \quad (6)$$

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

$$c_N = |N_p \cdot N| \quad (7)$$

$$c = c_B * c_N + (1 - c_N) \quad (8)$$

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

While the quality of the normals is not that high compared to the results of the PCL, our proposed method runs on the GPU and is thus faster (see Section 6). Furthermore note that the reprojection quality heavily depends on the tracking quality.

4.2 Light Estimation

The incoming light position must be known in order to be able to estimate a BRDF. The fish-eye camera captures the environment map and the DIR rendering system creates a dome of virtual point light (VPL) sources above the scene. We select a subset of these VPLs from the dome and use them for BRDF estimation. The selection criteria are that the VPLs have a high intensity and that there is no other selected VPL within certain distance.

5 BRDF ESTIMATION

Similar to Zheng et al. [25] highlights in the input image are removed and afterwards inverse diffuse shading is applied. However, in their approach the resulting buffer was just used for clustering. In contrast we also use this buffer as a diffuse reflectance map to keep the computation time low.

5.1 Highlight Removal

To estimate specular reflectance values similar colors need to be clustered since they are assumed to belong to the same material. However, specular highlights would form a separate cluster due to saturation, which is not desired. Our highlight removal is based on the work of Ortiz and Torres [13]. Instead of transforming the camera color image into the L1-Norm, we use the Hue Saturation Intensity (HSI) color space. Highlights should be detected at pixels where the color has high brightness but low saturation. As thresholds we set the minimum brightness to 0.9 and the maximum saturation to 0.1. In a first pass, the highlight detection result is written into a binary mask with a one where the brightness and saturation criteria are met and a zero otherwise. Then a morphological dilation with a disk (radius of 4 pixels) is performed. While Ortiz and Torres [13] perform a *Morphological Vectorial Opening by Reconstruction*, we use a rather simplistic reconstruction method. For each pixel that is masked as a highlight, a new color has to be found that ideally matches surrounding colors. We do this by iterating through neighboring pixels in an increasing circular manner until a pixel is found that is not masked as belonging to a highlight anymore. Then the color of the found pixel is used to substitute the color of the masked pixel. In this way, all highlights can be canceled out. Note that due to this highlight removal process, bright and weakly saturated objects may get misinterpreted as highlights. The results of the highlight removal operation are shown in Figure 3.

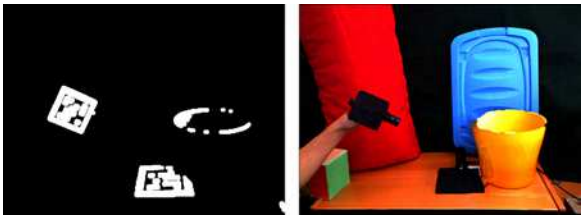

Figure 3: The left image shows the highlight mask. In a second step the masked pixels are filled as shown in the image on the right.

5.2 Diffuse Reflectance Estimation

After highlight removal we estimate the diffuse parameters per pixel by rephrasing the diffuse illumination equation. We end up with the following formula for the diffuse reflectance estimation k_d:

$$k_d = \frac{I}{\sum_{l=1}^{n} I_l (N \cdot L_l)}, \tag{9}$$

where I is the input intensity of the current pixel, I_l the intensity of the lth light source, L_l the direction towards the lth light source and n is the number of lights that are used for the BRDF estimation. Zheng et al. [25] estimate the diffuse parameters at a later stage because they used multiple RGB samples per vertex. We use the resulting buffer as diffuse reflectance map and as input for the clustering. The estimated diffuse reflectance map is shown in Figure 4. In the ideal case the different objects would have completely flat colors. However, this is not the case due to several simplifications that introduce consecutive errors in the pipeline. First, the environmental light is represented by only a few virtual point lights. Second, no shadows or indirect illumination are taken into account and third, the normal estimation is not absolutely correct.

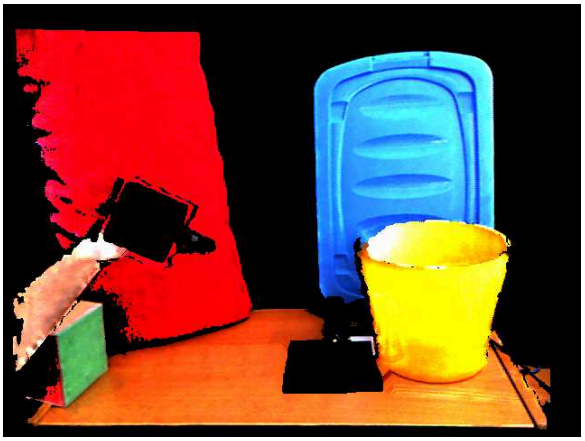

Figure 4: This image shows the estimated diffuse material component k_d. In the ideal case the objects would look perfectly flat and no variations due to different normals and thus illumination would be visible.

5.3 Clustering

Pixels with similar RGB colors in the diffuse reflectance map are assumed to have the same material and therefore need to be clustered. A novel K-Means implementation that is executed on the GPU performs the clustering. K-Means was introduced by Stuart P. Lloyd [9] and it consists of the following steps:

1. Randomly choose k cluster centers.
2. Assign each data element to the nearest cluster center using Euclidean distance.
3. Calculate new cluster centers by calculating the centroid over all data elements assigned to a specific cluster.
4. Repeat steps 2 & 3 until termination criteria are met.

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

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

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

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

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

$$cluster_c(T_{RGBD}, T^*) = \frac{avg(T_{RGBD})}{\sum_{i=0}^{n} \sum_{j=0}^{n} T^*_{i,j}} \tag{10}$$

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

Step 4: Repeat steps 2 & 3: In the original K-means method [9], the second and third steps are repeated until no data element changes the cluster anymore. This stopping criteria is too conservative for our needs. We need a fast clustering algorithm and thus have lowered the stopping criteria: First only a maximum number of 20 iterations are performed defining the upper bound for the computation time. Second if the variance change from one iteration to another drops below 10^{-4}, no further iterations are executed. By exploiting temporal coherence a low variance solution may be available after the first iteration and no new cluster center set needs to be processed. Note that the variance is calculated in a similar way to the cluster centers by exploiting mipmapping. As the squared distances for each pixel to the cluster centers are already calculated in the shader, the variance can be calculated nearly for free in step 2.

If the first run with the old cluster centers as initial values does not converge, the second run with random cluster centers gets executed. Then the cluster centers with the lower variance value are used for BRDF estimation. However, always using just the previous cluster centers could lead to a local minimum for clustering and there would be no way out to maybe find the global one. For this reason, in every fifth frame, the second iteration with random cluster centers will be executed anyway. Figure 5 shows the resulting clusters after K-Means is applied on the diffuse reflectance map.

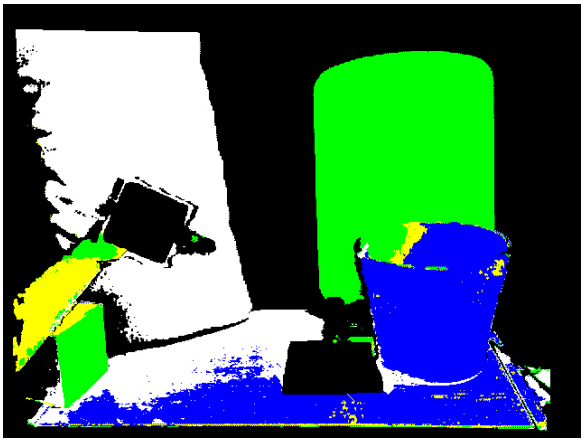

Figure 5: This figure shows the resulting clusters after K-Means is applied.

5.4 Specular Reflectance Estimation

One of the last steps needed in the BRDF estimation pipeline is the estimation of the specular intensity k_s and specular power n_s values per cluster. We assume white highlights and thus k_s is reduced to a scalar value. The paramters are estimated similar as proposed by Zheng et al. [25]. However, there are two main differences in our method. First, the solver works partly on the GPU and thus gains more speed than just a plain CPU implementation. Second, the positions of the light sources are not chosen to be fixed variables. The reason for this is that the positions are evaluated using importance sampling and thus can vary over time and furthermore need not to be at the exact position where a small light source is placed. However, the position of a light source highly influences the position of the specular reflection and therefore small variations of the initial positions are allowed to the solver.

For the non-linear optimization, a Nelder-Mead algorithm was used [12] with the following objective function evaluated on the GPU:

$$F_j = \sum_i \left[I_i - \sum_{l=1}^{n} I_l k_d (N \cdot L_l) + I_l k_s (V \cdot R_l)^{n_s} \right]^2 \quad (11)$$

where i iterates over all pixel intensities I_i which are related to cluster j. I_l is the intensity of the lth light source and k_d is the diffuse intensity vector of a cluster, which is set to the cluster center color. Note that for the specular component estimation, k_d is fixed and only the light source positions as well as k_s and n_s can be varied by the solver. N is the normal vector of the surface and R_l the reflection vector of the lth light source. V is a view vector pointing towards the camera. The result of the specular reflectance estimation is shown in Figure 6. Figure 7 shows a simple Phong-illuminated rendering on the left using the estimated k_d, k_s and n_s Phong illumination parameters. In this case, the same VPLs are used for illumination that are also used to estimate the BRDFs. In the image on the right side a rendering using DIR with an additional virtual pocket lamp is shown. Note the yellow indirect illumination on the real desk and on the virtual Buddha.

Figure 6: This image shows the result of the specular component estimation.

Figure 7: The left image shows a simple Phong rendering with the VPLs used for BRDF estimation. In the right image a virtual pocket lamp illuminates the real scene. Note the yellow color bleeding on the real desk and on the virtual Buddha.

6 RESULTS

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

6.1 Normal estimation

The two methods used for normal estimation differ in quality and computation time. Figure 8 shows a side-by-side comparison of the normal maps. The left normal map is calculated with the PCL library and a smoothing factor of 10. The average computation time is 223 milliseconds. The right normal map is computed with our proposed method in about 0.57 milliseconds. The PCL based normal map has a lot of holes, shown in grey color. In our method these holes are filled with the normals of the neighboring pixels. Even though these normals are not correct from a reconstruction point of view, they reduce the visible artifacts a lot. Furthermore note that our method produces sharper edges.

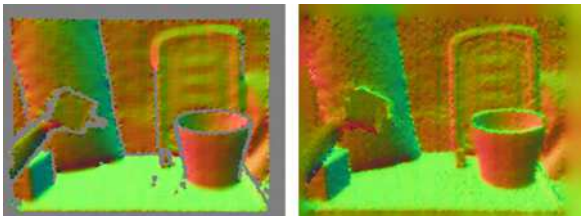

Figure 8: Comparison of the two implemented normal estimation methods. Left: Normals estimated using the PCL library [16] in 223 milliseconds. In grey areas no normal could be estimated by the PCL. Right: Our proposed method which takes 0.57 milliseconds.

6.2 K-Means clustering

We compared the proposed K-Means clustering implementation against the OpenCV library [3]. In the test setup a data set of 640×480 3-vector elements needed to be clustered. We ran both algorithms 5 times each time with different initial cluster centers. The interaction count of each run was set to 20. Note that no temporal coherence was exploited in order to get comparable results. The measured times include all the 5 runs and do not include the setup of the random data elements. Table 1 shows the execution times in seconds for 6 and 12 clusters.

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

Table 1: Shows a comparison between the K-Means implementation from OpenCV [3] and our GPU implementation. Both algorithms ran 5 times with 20 iterations. Timings show the total execution of the 5 runs in seconds.

Table 2 shows the average iteration count needed for the first 50 frames to get below the variance change threshold. The columns show the average number of iterations for 6 clusters (6C) and for 12 clusters (12C). The rows show if the cluster centers from the previous frame were used (frame^{-1}) or if the cluster centers where chosen randomly (random). We set the maximum iteration count to 30, which was never reached during this test.

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

Table 2: Shows the average iteration count when reusing the cluster centers from the previous frame or taking random new ones.

6.3 Performance Analysis

The BRDF estimation pipeline has several steps. Table 3 gives an overview on the time spent for each one. In this setup, 6 clusters and 4 VPLs were used to approximate the BRDFs.

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

Table 3: Shows the time spent on each pipeline stage.

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

Two main parameters can be tweaked to get a better performance for a given scenario. One parameter is the number of materials that are estimated every frame. The

second parameter is the number of virtual point lights that are used to approximate the surrounding illumination. Table 4 shows the impact of different cluster and VPL settings with a fixed maximum iteration count for the specular estimation set to 50.

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

Table 4: Shows the average fps with different VPL and cluster settings.

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

Furthermore it turned out that the solver has difficulties in finding appropriate values in certain cases. Sometimes there is simply no highlight due to a given VPL. We therefore introduced a threshold value for a maximum error. If the error is too large, we set the specular intensity k_s to zero. Another problem could be that the solver just has one single point of view per frame whereas Zheng et al. [25] used several photographs to perform a specular estimation. Recently upcoming techniques, however, promise to greatly improve the problem of temporal coherent BRDF estimation (see Section 8).

6.4 Critical Scenario

A critical scenario is shown in Figure 9 on the left. It shows a blue notebook and a horse made from porcelain placed on a wooden box. The light is mainly coming from the direction of the spotlight partially visible on the left side of the image, causing specular highlights on the blue notebook envelope and the wooden box. The number of clusters used in this scenario is six, and four virtual point lights are used to estimate the surrounding illumination. The average frame rate is 2.3 fps.

In this scenario, the clustering is not as distinct regarding the objects compared to the first scenario. Due to the highlights caused by the spotlight, the clustering step creates different clusters for the same material as shown in Figure 9 (right). Furthermore, the Kinect sensor has troubles finding depth values at the white borders of the tracking marker, resulting in holes in the estimations (see Figure 10 (left)). Figure 11 shows the resulting diffuse (left) and specular (right) estimations. The Phong-shaded result is shown in Figure 12.

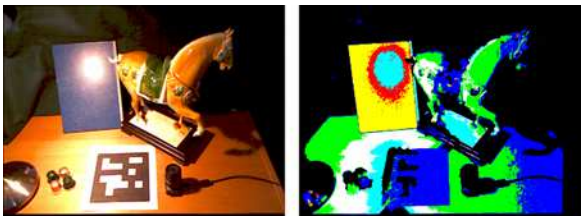

Figure 9: The left image shows the video input captured by the Kinect sensor. On the right side, the clustering result is shown.

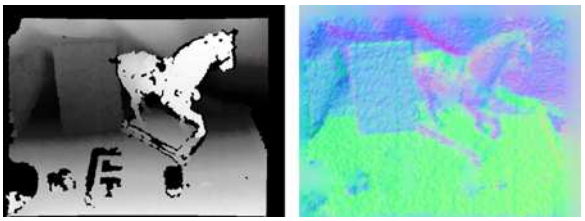

Figure 10: This figure shows the depth values acquired by the Kinect sensor on the left. Note that it failed to measure depth at the white borders of the tracking marker and the black fish-eye lense camera. On the right side the normal estimation from our proposed method is shown.

Figure 11: This figure shows the scene rendered with just the diffuse estimation (left) and specular estimation (right).

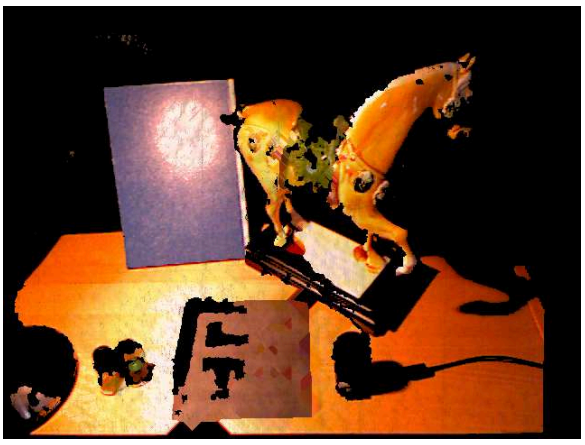

Figure 12: This figure shows the Phong-shaded result combining the diffuse and specular estimations.

7 LIMITATIONS AND DISCUSSION

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

happens when objects are very glossy, reflective, transparent or absorb too much of the infrared light. The infrared pattern also vanishes in direct sunlight, making the approach unsuitable for outdoor mixed-reality. Furthermore, the border of curved objects is also often missing from the depth map because the projected light pattern is too distorted there.

Since bright pixels are assumed to be highlights due to specular reflections, bright and weakly saturated objects may be misinterpreted as highlights. Furthermore, shadows are not considered directly in the current implementation. Pixels with a brightness value below a certain threshold are simply discarded.

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

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

Due to the real-time constraints several simplifications are introduced. The environmental illumination is approximated using a few virtual point lights, the normals have a lower quality compared to the PCL library and the clustering therefore also introduces some errors. All these simplifications lower the quality of the final BRDF estimation. However, since DIR mainly tries to compute visually plausible results rather than being physically correct, the estimated BRDFs should have a sufficient quality for mixed-reality scenarios.

8 CONCLUSION AND FUTURE WORK

We introduced a method to estimate the BRDFs of an augmented scene at interactive frame rates. The method does not need any precomputation, which makes it suitable for mixed-reality applications. The Microsoft Kinect sensor serves as a data input source to reconstruct the surrounding environment in the form of geometry and material properties. First, normals are estimated using a screen-space method exploiting temporal coherence. In the next pipeline stage we propose an adapted K-Means implementation that is specially tailored towards BRDF estimation and fast execution on the GPU. Temporal coherence is exploited here too, which allows us to find clusters faster than with a conventional implementation. The Phong parameter estimation is performed using a hybrid CPU / GPU variation of the Nelder-Mead optimization algorithm. The results demonstrate the feasibility of this method for mixed-reality applications.

In the future, we plan to enhance the quality and speed of this BRDF estimation method. It should be possible to gain a lot of speed by porting the Nelder-Mead optimization method to the GPU. Furthermore, recent techniques like KinectFusion [6] could greatly enhance the quality of the BRDF estimation.

9 ACKNOWLEDGEMENTS

This work was supported by a grant from the FFG-Austrian Research Promotion Agency under the program “FIT-IT Visual Computing” (project no. 820916). Studierstube Tracker is kindly provided by Imagination Computer Services GmbH.

10 REFERENCES

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

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

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

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

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

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

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

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

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

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

[11] Microsoft. Kinect sdk: research.microsoft.com/en-us/um/redmond/projects/kinectsdk/.

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

[13] 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.

[14] Saulo A. Pessoa, Guilherme de S. Moura, Veronica Teichrieb, and Judith Kelner. Photorealistic rendering for augmented reality: A global illumination and brdf solution. In *Virtual Reality*, pages 3–10, 2010.

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

[16] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

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

[18] Daniel Scherzer, Stefan Jeschke, and Michael Wimmer. Pixel-correct shadow maps with temporal reprojection and shadow test confidence. In Jan Kautz and Sumanta Pattanaik, editors, *Rendering Techniques 2007 (Proceedings Eurographics Symposium on Rendering)*, pages 45–50. Eurographics, Eurographics Association, June 2007.

[19] Richard Szeliski. Rapid octree construction from image sequences. *CVGIP: Image Underst.*, 58:23–32, July 1993.

[20] Gregory 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.

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

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

[23] Yizhou Yu, Paul Debevec, Jitendra Malik, and Tim Hawkins. Inverse global illumination: recovering reflectance models of real scenes from photographs. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 215–224, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

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

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