AUTHOR's VERSION, To appear in IEEE Transactions on Visualization and Computer Graphics, DOI 10.1109/TVCG.2017.2734426

Natural Environment Illumination: Coherent Interactive Augmented Reality for Mobile and non-Mobile Devices

Kai Rohmer, Johannes Jendersie, and Thorsten Grosch

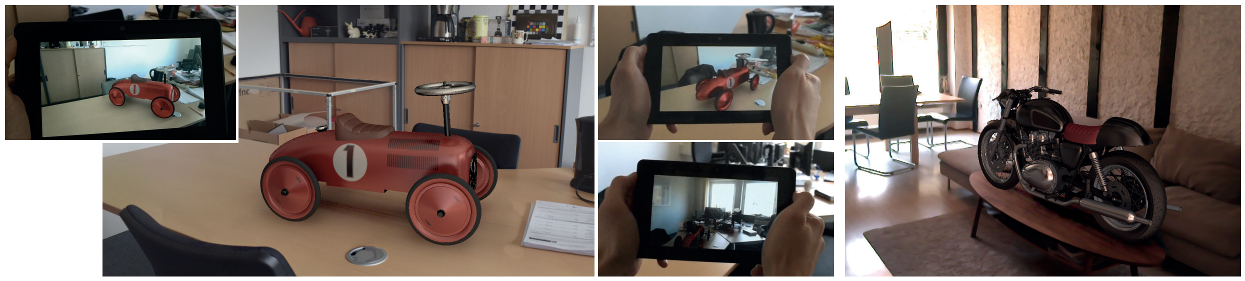

Fig. 1. Rendering results of our entire pipeline. A toy car with small metal applications rendered using different quality settings: low *(top left)*, high *(left)*, medium *(top center)* and *(bottom center)* (model courtesy of VILAC). A motor bike *(right)* with mirroring chrome elements (model courtesy of Maciek Ptaszynski).

Abstract—Augmented Reality offers many applications today, especially on mobile devices. Due to the lack of mobile hardware for illumination measurements, photorealistic rendering with consistent appearance of virtual objects is still an area of active research. In this paper, we present a full two-stage pipeline for environment acquisition and augmentation of live camera images using a mobile device with a depth sensor. We show how to directly work on a recorded 3D point cloud of the real environment containing high dynamic range color values. For unknown and automatically changing camera settings, a color compensation method is introduced. Based on this, we show photorealistic augmentations using variants of differential light simulation techniques. The presented methods are tailored for mobile devices and run at interactive frame rates. However, our methods are scalable to trade performance for quality and can produce quality renderings on desktop hardware.

Index Terms—Augmented Reality, Mixed Reality, Differential Rendering, Color Compensation, Impostors Tracing, GPU-Importance Sampling, Mobile AR, Scene Reconstruction, Light Estimation, Material Estimation, Depth-Sensing, Point Clouds, Global Illumination

1 Introduction

For many years, visual coherence in Augmented Reality (AR) is one of the research topics in focus of the computer graphics community. With the farther goal of photorealistic augmentation, various methods have been developed to insert virtual objects into a view of the real world so that they blend in seamlessly. Most of them require a complex hardware setup [38] or computation times that do not allow for interactivity. Our goal is to achieve plausible and compelling results with a mobile device and no other hardware at interactive frame rates.

To achieve coherent augmentations, there are three major topics we need to deal with: geometric registration, photometric registration and camera simulation [41]. Geometric registration of the virtual content and the real world is the foundation of coherent AR applications. It involves the pose estimation of the camera in the real world and the projection parameters of the camera – denoted as extrinsic and intrinsic parameters. We assume that this information is provided for each point in time and that they are synchronized with the virtual camera used for rendering virtual objects. This automatically provides us with several important visual cues, e.g.: relative size of objects and perspective, so we can focus on occlusions, shadows and shading to achieve a seamless blend of virtual and real objects. For more details and an introduction to tracking and registration, see the book of Schmalstieg and Höllerer [41].

The focus of our work are the other aspects: photometric registration and camera simulation. The first one describes the interaction of light between the real world and the augmentations. It involves the acquisition of the real environment including geometry, materials and light – ideally, the plenoptic function introduced by Adelson and Bergen [1]. Based on the captured data, light simulation approaches can be used to compute the appearance of the virtual objects in the real environment and their virtual influence back on the real scene.

Camera simulation takes the physical behavior of the camera into account. Unlike professional DSLR cameras for which exposure, shutter speed, sensitivity, white balance and other options can be specified, simple mobile cameras lack of these options and often provide only an exposure compensation factor and the ability to select one of a few predefined white balance settings. Mobile sensors and their APIs are designed to allow fast and nice-looking snapshots but provide no basis for measuring radiance. Even their image processing pipeline and parameters are unknown, such that generating low dynamic range (LDR) images that could be merged into high dynamic range (HDR) is cumbersome. To overcome this limitation, we present an approach to estimate these unknown adjustments with respect to a reference image. This estimation is successively applied to new LDR images to transform them into the reference space for further processing. In the context of AR, we use the approach to create a reconstruction of the environment, superimpose live camera images by virtual objects plus one of multiple differential rendering methods, and eventually transform the composed image back into the original color space of the input image.

• *Kai Rohmer is part of the Graphical Data Processing and Multimedia Group at the TU Clausthal, Germany. E-mail: kai.rohmer@tu-clausthal.de.*
• *Johannes Jendersie is part of the Graphical Data Processing and Multimedia Group at the TU Clausthal, Germany. E-mail: johannes.jendersie@tu-clausthal.de.*
• *Thorsten Grosch is head of the Graphical Data Processing and Multimedia Group at the TU Clausthal, Germany. E-mail: thorsten.grosch@tu-clausthal.de.*

The contributions of this paper are:

- An estimation of the unknown image optimizations performed by mobile camera sensors.
- Usage of this estimation to acquire the HDR radiance of an entire scene consistently with respect to a reference frame.
- A GPU-optimized surface normal estimation by local least squares plane fitting.
- Introduction of Impostor Tracing [42] for photorealistic AR rendering, combined with GPU Importance Sampling [2].
- Usage of a new sampling strategy tailored for AR.
- Using the inverse operation of the estimated camera optimizations to reduce notable differences between virtual and real objects in terms of perceived color and brightness.

2 Related Work

AR with correct illumination goes back to the differential rendering introduced by Fournier et al. [9] and Debevec [6]. The idea is to first reconstruct 3D geometry, camera parameters, materials and light sources. Then, two global illumination simulations are computed, one which contains virtual objects (LS_{Obj}) and one which does not (LS_{Env}). The (masked) difference between both, $\Delta LS = LS_{Obj} - LS_{Env}$, contains shadows and indirect illumination, i.e., the impact of virtual objects on the scene. Adding the virtual objects and ΔLS to the original photograph yields the augmented image. This is also the basis for the rendering in our pipeline.

Interactive AR with Consistent Illumination. Several solutions were presented which compute this difference in radiance at interactive to real-time speed, based on Instant Radiosity [27], Light Propagation Volumes [10], Irradiance Volumes [12,14,15], Irradiance Caching [23], Path Tracing [24] and Spherical Harmonics (SH) [34]. An overview of the recent techniques is given by Kronander et al. [29]. Rohmer et al. [37–39] introduced a hybrid approach that allows to display the differential rendering on a mobile device using a distributed solution with tablet PCs and a stationary PC. They require an external tracking system and HDR video cameras.

Photorealistic AR with a Consumer 3D Scanner. Due to the availability of consumer 3D scanning hardware, some solutions were presented which do no longer require a manual reconstruction of the 3D geometry. Lensing et al. [30] show photorealistic augmentations based on the depth values of a Kinect camera. Franke [11] introduced Delta Voxel Cone Tracing, based on a reconstructed 3D voxel model and Voxel Cone Tracing [3]. Gruber et al. [17] showed an image-space approach which directly works on pixel colors and the depth image for near-field illumination. Earlier, Gruber et al. [16] estimated a distant monochromatic EM in SH basis from observations of many surface points on arbitrary geometry. Differential rendering based on PRT and ray-casting for occlusion detection is applied to augment the scene. In accordance to us, the authors assume a Lambertian reflectance for real surfaces, but they also assume distant light, while we are able to compute colored near-field illumination. Zhang et al. [45] presented a differential rendering approach for several Mixed Reality operations that runs only on a tablet PC. This approach requires a semi-automatic mesh reconstruction from the recorded point data and a non-linear optimization (15-30 min). We consider the extension of our approach to use such a global exposure compensation as part of future research.

Reconstructing 3D Geometry. There are different ways to create a 3D model using a moving depth camera. Kinect Fusion [20] uses a volumetric data structure and a signed distance function to reconstruct a 3D mesh. Alternatively, a 3D voxel model can be constructed using a Hashing approach [33]. Several extensions exist for these methods [5, 43, 44]. Our method does not build a new geometry representation but instead works on the 3D point cloud with HDR color values, acquired by looking around with the tablet camera, without any limitations on the scene extent.

Color Compensation. For most mobile devices, exposure time and white balance of the camera change automatically without manual control. Therefore, a color compensation must be included when recording the real radiance values by moving the mobile camera. Possible solutions for this problem are the exposure and vignetting calibration for stitching panoramic images [13] and the radiometric calibration for videos [18, 26]. For stitching an HDR environment map, Kán [22] showed that this be can recorded with a mobile device if the exposure time of the camera is known. lIKE Kán, Meilland et al. [32] assume that all camera parameters can be fixed. Allowing shutter speed to be adjusted automatically results in a single degree of freedom that can easily be estimated or found in the meta-data of the images. In this paper, we present an approach that deals with devices that allow no control of the acquisition parameters and offer no meta-data per image. The video preview mode that is used for AR applications is subject to these restrictions [22].

To address this issue, Knecht et al. [28] adaptively map the colors of virtual objects to the colors present in the current camera image. The general idea is to create a mapping between the camera image and the partial solution LS_{Env} of the differential rendering. Applying the map to virtual objects yields coherent colors. However, to compute LS_{Env}, a scene reconstruction containing material properties is required. Our goal is to find a mapping that allows to create such a reconstruction.

Recently, exposure correction methods were used for the correction of recorded color values of a Kinect camera [36] and for the color values of a tablet camera [45]. While these solutions search for a global optimum of a given set of images, our method works progressively for each new camera image on the tablet PC.

Capturing Illumination. Several tools exist for quickly capturing the real illumination, like light probes [6] or fish eye cameras [38]. As an alternative, inverse rendering techniques can be used to reconstruct the incoming illumination from the visible radiance values in the camera image [35,36]. Meilland et al. presented an approach to create an image-based representation of the scene that is then used for AR rendering [32]. While mainly focusing on the pose estimation for proper spatial registration between key frames, the registration of the image intensities relies on the relative change in shutter speed. This works well if all other camera settings can be fixed. If not, the compensation presented in this paper can help as it deals with a general linear color transformation. Storing radiance directly in the point cloud is easier but requires a good pose estimation. For rendering, Meilland et al. create virtual light probes and extract a few discrete light sources for shadow computations. We also render virtual light probes (EM/DIT), but we consider all directions for light computations and shadows respectively. Eventually, theirs and our approach are following the same goal of capturing the environment scene including geometry and radiance. At the point where the virtual light probes are created, both approaches can be exchanged.

3 Overview

We are using the *Google Tango Development Kit*[1] which provides an LDR image, a 3D pose estimation of the camera, and a sparse depth image as well as timestamps for each of them. Based on this input the following steps are performed:

Color Compensation Estimation. To cope with unknown color compensation applied by mobile camera drivers, we estimate this compensation with respect to the already captured scene (see Sec. 4). The previous captured parts are rendered from the new camera position and the resulting reference image is compared to the current input image. Pixels that probably show the same real-world geometry are selected and used in a system of equations to estimate the transformation that maps the color of the input pixels to the color of the reference pixels. This transformation is applied to all pixels of the most recent LDR input image which yields a corrected input image for further processing and for augmentations later.

[1]Google. *Project Tango Developer Website*. 2017

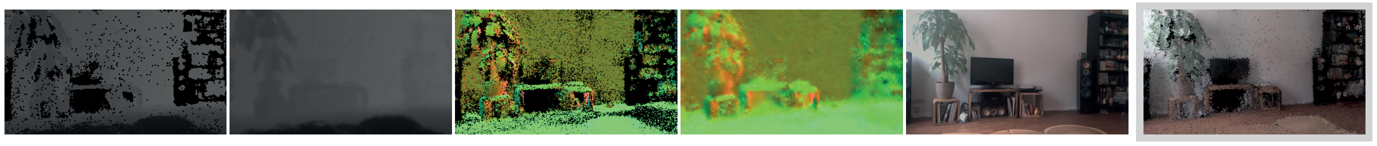
Fig. 2. G-Buffer. *Left to right*: splatted environment sample depths, filled depths, sample normals, filled normals, corrected color image. The last image shows the radiance of the samples. It is not used for rendering but as reference image during the compensation estimation.

Capturing the Environment. The captured scene is stored as an unstructured point cloud, which is updated every time a new set of samples is provided by the depth sensor. These samples are back-projected from image space into 3D world space and the resulting position is stored with further information like corrected color and estimated normal. We call these small structs *Environment Samples* and store them in a probabilistic hashmap [19]. The normal estimation, insertion and deletion of samples is detailed in Sec. 5.

G-Buffer as AR basis. To allow for basic occlusion between virtual and real objects a z-buffer containing the real-world objects is required. Splatting the point cloud into screen space yields the z-buffer but also normals and colors – which are actually used as reference image in the color compensation step and for material estimation during rendering. After filling holes by push pull steps [31], we have a valid G-Buffer [40] that can be used for a basic and more advanced AR rendering (see Figure 2). As the G-Buffer generation is straightforward we will not provide further details. Note that we do not need the depth sensor anymore. So, as long as the camera pose is provided, any device can be used. However, depth sensing could be used to improve the G-Buffer in future extensions.

Coherent Rendering. For the rendering of the virtual objects, we suggest using an image-based ray tracing variant. Therefore, we combine GPU Importance Sampling [2] with Impostor Tracing [42]. A *Distance Impostor (DI)*, basically a RGB-Depth environment map, centered around the virtual object, serves as input. This structure is similar to Reflective Shadow Maps [4] and is created just like the G-Buffer before. We apply three alternative implementations for differential rendering [6]: the suggested *Distance Impostor Tracing (DIT)*, standard *Environment Mapping (EM)* and *Voxel Cone Tracing (VCT)*; for comparison and to emphasize the flexibility of the approach. We also present an improved sampling strategy tailored for differential rendering (see Sec. 6 and 7).

Inverse Color Compensation. Since the unknown compensation, estimated in the first step, was introduced by the vendors to look pleasant to the user, we transform the augmented image back into the original input image color space assuming that this results in an adequate tone mapping.

4 Color Compensation Estimation

Assuming we already captured parts of the environment and these parts are at least partially visible in the current camera image, we aim for estimating a transformation to map between the visible colors of the overlapping areas. Figure 3 illustrates the basic idea. The inputs at time t are a camera image in an unknown color space and a rendering of the partially acquired scene in the reference color space. We select pairs of pixels from the images that probably show the same real-world geometry. The pixels selected from the input image form the set $\mathcal{U}_t = \{\mathbf{u}_i\}$ and pixels from the reference image the set $\mathcal{R}_{t-1} = \{\mathbf{r}_i\}$, respectively. After estimating the compensation function c_t that maps the color of the pixels in set $\mathcal{U}_t$ to the colors in $\mathcal{R}_{t-1}$, we can apply the function to all pixels of the input image to transform them into the reference color space. Additionally, we obtain the inverse function to map augmented images back into the original camera color space:

$$c_t : \mathcal{U}_t \to \mathcal{R}_{t-1} \qquad\qquad c_t^{-1} : \mathcal{R}_{t-1} \to \mathcal{U}_t$$

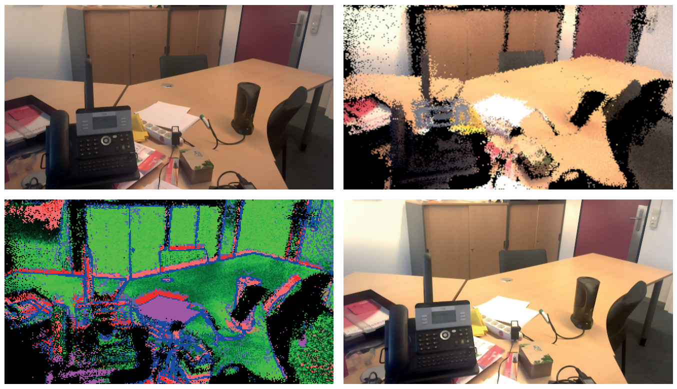
Fig. 3. Color Compensation. *Top left*: A camera input image in an unknown color space. *Top right*: The corresponding reference image rendered from the point cloud (see Sec. 6.1). *Bottom left*: The color-coded selection of sample pairs (green = valid). *Bottom right*: The corrected input image in the reference color space. Note that this process can output HDR values outside the [0,1] range.

4.1 Selecting Sample Pairs

The selection of pixels that are used in the linear system decides about the robustness of the result. On one hand, we need a large set of samples to get a good estimation. On the other hand, we need to avoid wrong pairs, for instance based on occlusion of not yet scanned geometry. In the first step, we remove all pixels that are easy to reject. This includes pixels that:

- are under or over-saturated in the camera image,
- have no data in the rendering of the point cloud,
- have strong color gradients in either of the images.

Strong gradients are likely to contain edges that could be matched to the wrong surface because of inaccurate tracking.

In the next filtering step, the pixel colors are transformed into HSV color space. After mapping hue and saturation into $[0,1]$, we reject pairs that show a very strong discrepancy in color based on a coarse filter applied on the Euclidean distance between hsv$(\mathbf{u}_i)$ and hsv$(\mathbf{r}_i)$ in the hue-saturation-plane $(\cdot)_{hs}$. Note that the brightness value has no influence here.

Assuming that changes in the input are small between two frames, we can transform the input set $\mathcal{U}_t$ by the estimated function of the last frame c_{t-1}. Hence, $c_{t-1}(\mathbf{u}_i)$ will be close to $\mathbf{r}_i$ and stronger filtering parameters can be used than before. This time, we reject pairs based on their Euclidean distance in HSV space including the brightness dimension. Figure 3 shows the input, the output and a color coding of the filtering process. Green pixels are accepted with a certainty weight w_i according to their brightness, see Eq. (1). Blue ones are discarded because of gradients, purple pixels were rejected due to saturation and the red ones have a too large distance in HSV.

At this point, different models can be fitted to estimate c_t. We decided to use a linear transformation in linear RGB space to describe c_t. We also fitted a coarse approximation a_t that simply aligns the average intensity per color channel. It serves as backup in case of failure, e.g., because of too few samples. Depending on the sensor, other color spaces and models might work better.

We choose a linear transformation here because we cannot make assumptions about the actual type of correction. However, we want to find a robust estimation that can deal with basic operations like a scale in brightness, small color shifts introduced by changing white balance and simple contrast enhancements. To allow any of these operations, the input colors need to be in a linear color space. Ideally, this requires a transformation of the real input using the camera response curve which is unknown, too. Debevec [7] shows how to reconstruct such a camera curve for an exposure series. Unfortunately, it is not clear how to reconstruct a camera response curve in a system without exact control over the camera parameters. Based on the measurements in [45], we achieved good results by assuming a default gamma curve of 2.2.

4.2 Color Map Approximation

In both, the approximation and the linear estimation, the samples are weighted based on their similarity in the hue-saturation-plane:

$$w_i = \left(1 - \min\left(1, \|\mathrm{hsv}\left(\mathbf{u}_i\right)_{hs} - \mathrm{hsv}\left(\mathbf{r}_i\right)_{hs}\|_2\right)\right)^{\beta}. \tag{1}$$

The exponent β allows to further penalize larger distances. In all our scenes, $\beta = 2$ was chosen for a_t and $\beta = 8$ for c_t. To get a coarse approximation of the unknown color compensation, an average scaling per color channel, $\mathbf{s}$, is computed:

$$\mathbf{s} = \frac{\bar{\mathbf{r}}}{\bar{\mathbf{u}}} = \frac{\sum_i^n \mathbf{r}_i \cdot w_i}{\sum_i^n \mathbf{u}_i \cdot w_i},$$

where n the number of pixel pairs. The per color channel scaling results in the approximation matrix A_t:

$$A_t = \begin{bmatrix} \mathbf{s}_r & 0 & 0 \\ 0 & \mathbf{s}_g & 0 \\ 0 & 0 & \mathbf{s}_b \end{bmatrix}.$$

4.3 Linear Map Estimation

To setup the linear system, the sample sets are written as matrices. The estimation function c_t is described as a linear matrix C_t:

$$U_t = \begin{bmatrix} \mathbf{u}_{1_r} & \mathbf{u}_{2_r} & \cdots & \mathbf{u}_{n_r} \\ \mathbf{u}_{1_g} & \mathbf{u}_{2_g} & \cdots & \mathbf{u}_{n_g} \\ \mathbf{u}_{1_b} & \mathbf{u}_{2_b} & \cdots & \mathbf{u}_{n_b} \end{bmatrix} \text{ with } \mathbf{u}_i \in \mathcal{U}_t$$

$$R_{t-1} = \begin{bmatrix} \mathbf{r}_{1_r} & \mathbf{r}_{2_r} & \cdots & \mathbf{r}_{n_r} \\ \mathbf{r}_{1_g} & \mathbf{r}_{2_g} & \cdots & \mathbf{r}_{n_g} \\ \mathbf{r}_{1_b} & \mathbf{r}_{2_b} & \cdots & \mathbf{r}_{n_b} \end{bmatrix} \text{ with } \mathbf{r}_i \in \mathcal{R}_{t-1}$$

$$C_t \cdot U_t = R_{t-1}.$$

To solve for the compensation matrix C_t we minimize the following energy function:

$$\underset{C_t}{\arg\min} \ \|W_t \left(C_t \cdot U_t - R_{t-1}\right)\|_2,$$

where the diagonal matrix $W_t \in \mathbb{R}^{n_b \times n_b}$ holds the weights of the sample pairs $W_{ii} = \sqrt{w_i}$ computed by Equation (1). Solving in the sense of least squares yields:

$$C_t = \left(\left(U_t \cdot W_t^2 \cdot U_t^T\right)^{-1} \left(U_t \cdot W_t^2 \cdot R_{t-1}^T\right)\right)^T.$$

Further, we can incorporate knowledge about the expected transformation. We know that the change in exposure is a strong influence factor that basically scales the brightness, and this scaling factor is already computed by the approximation A_t. To improve the robustness, a regularization can be applied to force the solutions closer towards A_t. Therefore, the energy to minimize is changed to:

$$\underset{C_t}{\arg\min} \ \|W_t \left(C_t \cdot U_t - R_{t-1}\right)\|_2 + \gamma \|C_t - A_t\|_2,$$

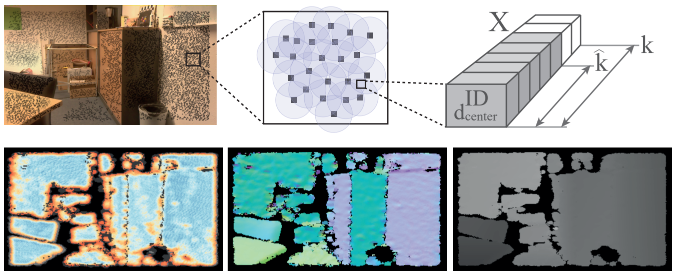

Fig. 4. Normal Estimation. *Top:* Sparse depth sample given as input are spanned to overlapping disks. All disks that overlap a certain pixel are added into a list X at that pixel. The $\hat{k}$ elements in the list form the local neighborhood of the pixel. *Bottom left:* Color coded number of elements used to fit a plane, surface normals *(center)* and improved depth *(right)*.

Now, solving in the sense of least squares yields:

$$C_t = \left(\left(U_t \cdot W_t^2 \cdot U_t^T + \gamma I\right)^{-1} \cdot \left(U_t \cdot W_t^2 \cdot R_{t-1}^T + \gamma A_t\right)\right)^T, \tag{2}$$

where I is the identity matrix and γ a small scalar factor, set to $5 \cdot 10^{-2}$ in all our examples.

Note that both matrices in the inner braces of Equation (2) are of size 3×3 so they can easily be computed on the GPU. In fact, the entire process of estimating A_t and C_t requires only one iteration over all sample pairs and a parallel reduce operation to compute the sums and the matrix products.

For a large number of sample pairs, usually $n > 2000$ in our experiments, and input images that do not lack information in one or more channels completely, the system is nonsingular and can be inverted. This is important to transform rendering results back into the original color space of the input image, as described in Sec. 6.6.

If the sample count is below this threshold, the approximation A_t is used and the frame is marked as approximated to skip the capturing of environment samples for this frame. In the first frame we initialize the system by using the identity, $C_0 = I$, which renders the color space of the first frame the reference color space. While moving in the scene, darker and brighter areas are observed with automatically selected appropriate camera settings. Hence, floating point precision images of different levels of brightness are provided for further HDR processing.

This approach is applicable to any camera system, independent of the availability of a depth sensor. However, depth information could be used in pair filtering.

5 Capturing the Environment

The goal of this stage of the pipeline is to create a reconstruction of the real environment. For basic augmentations, the position of the real surface is sufficient, whereas for coherent rendering, normals are required as well. As input serve the sparse depth image delivered by the sensor (see Figure 4, gray dot over the color image) and the corrected camera image to gather corresponding color values. Each measured sample is back-projected to world space coordinates and stored in a buffer. Its buffer index is denoted as unique sample id i. Normals could be approximated by computing the gradient of the interpolated positions. To focus on smooth normals, we address the problem by fitting a plane into the samples of the local neighborhood of each pixel in the sparse depth image.

Figure 4 illustrates this process step by step. First, we identify the samples of the local neighborhood by using the rasterization pipeline of the GPU. Therefore, we span a disk around each sample. The radius depends on the resolution of the depth image and the density of the measured samples. It is a device specific constant that must be specified such that the number of overlaps per pixel equals the size of the desired neighborhood k. For each pixel of the depth image, a list of size k is used to store the ids of the overlapping disks and thereby the pointers to all samples of the local neighborhood.

In the second step, a plane is fitted to the world space positions of each pixel's list. We now consider an arbitrary pixel and denote the respective list as set $\mathcal{X}$ containing the samples $\mathbf{x}$. Generally, this list is not entirely filled, so we define the number of samples in the list as $\hat{k} \leq k$. Even though plane fitting in 3D is a common problem, we need to perform this task for each pixel as fast as possible on the GPU. Since it is an eigenvalue and eigenvector problem, the common solution requires a Singular Value Decomposition, which is not feasible in our context. By reformulating the problem, the solution can be found easier. Like in a Principal Component Analysis we first shift the mean of all samples into the origin of a local coordinate system, so the resulting plane will contain the origin:

$$\hat{\mathbf{x}}_i = \mathbf{x}_i - \frac{1}{\hat{k}} \sum_j \mathbf{x}_j \qquad \text{for } i = 1, \ldots, \hat{k}$$

Each vector $\hat{\mathbf{x}}$ defines a direction from the local origin. The normal $\mathbf{n}$, we are looking for, is perpendicular to that direction:

$$\hat{\mathbf{x}}_i{}^T \mathbf{n} = x_i \cdot \mathbf{n}_x + y_i \cdot \mathbf{n}_y + z_i \cdot \mathbf{n}_z = 0$$

The trick to avoid the SVD is to split the problem into three smaller ones that are easy to solve. After shifting by the mean, we are now looking for a plane that intersects the origin. There are two degrees of freedom describing the rotation of the plane, but we have three unknown components of $\mathbf{n}$. We now assume that the normal does not lie in the xy-plane. In that case, we can set $\mathbf{n}_z = 1$ such that $x_i \cdot \mathbf{n}_x + y_i \cdot \mathbf{n}_y = -z_i$ and solve the overdetermined system using normal equations for the other two components:

$$\begin{bmatrix} \sum x_i x_i & \sum x_i y_i \\ \sum y_i x_i & \sum y_i y_i \end{bmatrix} \begin{bmatrix} \mathbf{n}_x \\ \mathbf{n}_y \end{bmatrix} = - \begin{bmatrix} \sum x_i z_i \\ \sum y_i z_i \end{bmatrix}.$$

This small system can be solved efficiently by using the determinant and Cramer's rule which yields the normal direction $\begin{bmatrix} \mathbf{n}_x & \mathbf{n}_y & 1 \end{bmatrix}^T$ that just must be normalized.

However, this does not work if the normal lies in the xy-plane in which case the determinant gets zero. Therefore, the assumption is altered and the determinant is computed for all three cases. Eventually, only the case with the largest determinant is evaluated to compute the plane normal $\mathbf{n}$. In combination with the mean sample position that was used to shift the samples into the local origin, a world space plane is defined. This approach to plane fitting in 3D is not new[2], but it is well suited for a GPU implementation.

To avoid fitting planes across discontinuities in depth we search for the sample with the smallest screen space distance to the considered pixel and discard samples based on their depth distance to this reference sample. In our experiments, a threshold of 0.1 m shows good results even when the surface is seen from a grazing angle.

Intersecting the fitted plane with the view ray from the camera through the considered pixel yields a distance that is used to refine the depth image and thereby the position of the samples.

Figure 4 shows the resulting depth and normal image as well as a visualization of $\hat{k}$ for each pixel where red means a small number and dark blue the maximum number. As can be seen, most of the pixels show a medium blue so the lists are not entirely filled. This is intended as the radius, that is used during the filling of the list, needs be chosen conservatively. If the radius is too large it is possible to miss neighbors because the list is already full. In case with two or less samples in a list, we cannot estimate a normal, which is visualized by dark gray color or black if there was no sample at all. We also stop the normal computation if the largest determinant is close to zero which means that the system is of bad condition. This case is illustrated by a light gray.

After creating the depth and normal image, both are sampled at the locations of the input samples and the gathered information are added as *Environment Sample* to the point cloud describing the scene. This point cloud is stored in a probabilistic hash grid with space for 4M

[2]Emil Ernerfeldt. *Fitting a plane to many points in 3D*. Blog, 2015

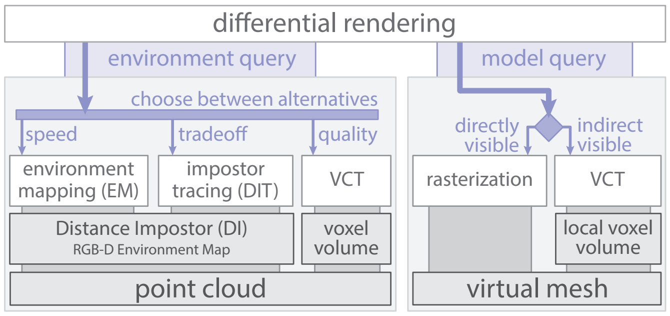

Fig. 5. Overview of Techniques and Data Structures. The differential rendering queries radiance of the environment and the virtual object. Different implementations can be used to answer the request. In general, such a query can be interpreted as ray or cone cast.

elements and a virtual cell size of 0.05 m. If there already is an element at the insertion address, it is replaced randomly with a probability of the inverse number of collisions in that cell. Therefore, a counter per cell keeps track of the collisions.

Besides adding elements to the cloud, samples can be removed, too. This is important to handle movements of real objects and erroneous samples. An old *Environment Sample* can be removed if it has a smaller depth than the current depth image at the same location. Obviously, it is not present in the current view and therefore should not exist. The removed samples are often those, which have been captured over a larger distance. Nevertheless, these far samples cannot be ignored in the first place, because they are valuable for generating the reference image.

6 Interactive Coherent Rendering

As illustrated in Figure 5, our differential rendering system uses an interface to query radiance from the scene and from the virtual object. By providing alternative implementations for the environment queries we can offer different quality/performance options. However, we are also able to compare the proposed *Distance Impostor Tracing (DIT)* to the commonly used *Environment Mapping (EM)* (related to [6]) and *Voxel Cone Tracing (VCT)* (related to [3, 11]).

In Sec. 6.1 we describe how the captured point cloud is processed into the data structures required by the different implementations. We introduce the representation of the virtual object in Sec. 6.2 and show how to use the scene representation to illuminate this object in Sec. 6.3 and Sec. 6.5. To compute the influence of the object onto the environment we use differential lighting as explained in Sec. 6.4. Details to the final composition are covered in Sec. 6.6.

Details on how to solve equations in Sec. 6.3 and 6.4 by using *Monte Carlo* integration are provided in the following Sec. 7. Here we introduce an idea to improve the rendering for few samples incorporating knowledge of the object influence.

6.1 Scene Representation

The differential rendering, including the rendering and shading of the virtual object, requires to query the environment illumination multiple times. Each of these queries is a ray or cone cast from the surface position $\mathbf{x}$ into the environment to look up the radiance incident from the direction of the ray, the direction of solid angle $\boldsymbol{\omega}_i$. Instead of dealing with large sets of points, we use dedicated representations for the techniques to realize this query. For EM and DIT this representation is a *Distance Impostor (DI)*, whereas for VCT a voxel volume is used.

(Scene) Voxel Volume Cone tracing requires a dense regular voxel volume with a filled mipmap chain [3, 11]. Therefore, all samples are added to the most detailed mipmap level of the volume where each voxel stores the RGB radiance of the environment samples and an opacity value of 1.0. After generating the mipmap chain, the voxels in the higher levels store averaged radiance and opacity values for lookups with larger cone radii. Assuming a static scene, the volume is filled only

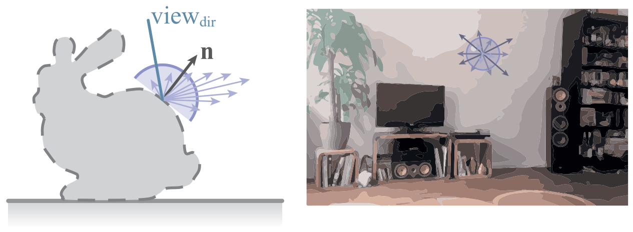

Fig. 6. Integrating the incoming radiance on a virtual object for rendering *(left)* and on a real surface for material estimation *(right)*.

once after completing the environment acquisition and contains the entire scene. Depending on the size and the resolution of the volume, this data structure is usually limited due to the memory requirement.

Distance Impostors To be able to deal with larger scenes we suggest using a distance impostor which is an environment map that stores radiance and distance into all directions from a specified center position. It can be created by splatting – just like the G-Buffer – while using a cube map as target. The position of the DI should be close to the virtual object – ideally at the center of it. Thus, it needs to be updated as the object is moved in the scene, but depending on the size of the object a threshold can be used to limit the updates to trigger only after larger changes in position. The orientation of the map can be aligned with the world coordinate system for easier access. To enable ray or cone casting from positions different from the center position we use *Impostor Tracing* presented by Szirmay-Kalos et al. [42]. Here, the direction for the lookup into the map is refined iteratively based on the stored distance to approximate the correct intersection point of the ray with the environment.

To favor speed over accuracy we also investigate standard environment mapping which simply uses the color channels of the DI but also depends on the depth for occlusion tests.

6.2 Virtual Object Representation

To compute the shading of virtual objects there are no special requirements and any triangle meshes can be used.

While the virtual object itself is rendered using the rasterization pipeline, additional ray or cone casts are required for indirections to enable self-shadows and reflections for instance (see Figure 5). Independent of the scene representation, we decided to use VCT for this task as testing the whole triangle mesh for intersections is too expensive. Therefore, we use a solid volume with mipmaps that is small, tightly fit to the object, and is defined in the object's local coordinate system. The solid voxelization is done on the GPU [8] and does not change over time as long as the object is not deformed, e.g., by vertex skinning [25]. Rigged transformations can be applied to the ray or cone instead of rebuilding the volume. However, the meshes need to be closed (watertight), which is a limitation of the solid voxelization algorithm [8]. The volume also stores materials and local normals at the boundary voxels.

6.3 Rendering the Virtual Object

To compute a physically correct shading of the virtual object surface we aim for solving Kajiya's rendering equation [21]:

$$L(\mathbf{x},\omega_o) = \int_{\Omega} L(\mathbf{x},\omega_i)\ f_r(\mathbf{x},\omega_i,\omega_o)\ \cos\theta\ \partial\omega_i \tag{3}$$

where L is the radiance from or into a direction ω, f_r is the *Bidirectional Reflectance Distribution Function (BRDF)* which describes how much of the (incident) irradinace is reflected as radiance into the outgoing direction ω_o at surface position $\mathbf{x}$, θ is the angle between the surface normal and the incoming direction ω_i. The integration domain Ω is the hemisphere over the surface, as visualized in Figure 6 (left). The figure also shows outgoing sample directions that are concentrated around the reflected view direction (ω_o). This distribution depends on the BRDF, in our case a Blinn-Phong BRDF for glossy materials, but any other BDRF that can be sampled properly can be used instead. To evaluate this equation, we use ray or cone tracing into these reflected directions (see Sec. 7).

To include self-shadowing of the virtual object, each ray is not only traced in the environment but also in the object's volume. In case the ray hits opaque voxels, the ray is stopped and the radiance, $L(\omega_i)$, is set to zero or to a predefined ambient value to compensate the missing indirect light. Optionally, one or more secondary rays are cast recursively, as usually do in offline rendering. As the voxelization stores normals and material coefficients this is supported and used in Figure 1 *(left)*. Note the reflected wheels on the body of the toy car.

6.4 Differential Background Rendering

For a coherent rendering, we also need to estimate the influence of the virtual object on the environment. Therefore, we use *Differential Rendering* [6] to compute a correction image from two global illumination simulations: $\Delta LS = LS_{Obj} - LS_{Env}$. Both simulations require to solve the same integral we used before (see Equation (3)) but at this point we are missing the material coefficients for the BRDF. Thus, the reflectance parameters for visible surfaces must be estimated before computing the integrals. Therefore, we assume that the environment material is Lambertian. Hence, the BRDF is independent of the observer direction ω_o and specified by the constant reflectance coefficient ρ. Equation (3) is then solved for ρ [6]. This involves an integral over the hemisphere above a real surface point $\mathbf{x}$ as shown in Figure 6 (right), where $L(\mathbf{x})$ is equal to the color visible in the corrected input image:

$$L(\mathbf{x}) = L(\mathbf{x},\omega_o) = \int_{\Omega} L(\mathbf{x},\omega_i)\,\frac{\rho}{\pi}\ \cos\theta\ \partial\omega_i \tag{4}$$

$$\rho = \pi \frac{L(\mathbf{x})}{\int_{\Omega} L(\mathbf{x},\omega_i)\ \cos\theta\ \partial\omega_i}. \tag{5}$$

Now, the influence of the virtual object, $\Delta L(\mathbf{x})$, can be estimated by solving an integral similar to Equation (4) where $\Delta L(\mathbf{x},\omega_i)$ is queried for each incoming direction instead of $L(\mathbf{x},\omega_i)$. The three different cases that can occur are illustrated in Figure 7.

Ray 1 does not hit the virtual object and ends in the environment. In this case, the influence in incoming radiance $\Delta L(\mathbf{x},\omega_i)$ at the surface point is zero.

Ray 2 hits the virtual object before it hits the environment, ($d_{\text{Obj}_2} < d_{\text{Env}_2}$). Here we gather the radiance from the environment as well as the radiance of the object and compute $\Delta L(\mathbf{x},\omega_i) = L_{\text{Obj}} - L_{\text{Env}}$. Note that the object radiance, L_{Obj}, is generally not known for rays from arbitrary directions, so we again could stop and return zero or an ambient term. But instead we trace a reflected ray into the environment and compute an indirect light bounce. For a correct result, this bounce needs to be computed like in Sec. 6.3. By assuming a diffuse virtual object (only for this bounce), we can achieve color bleeding (e.g., on the wall in Figure 9), but we neglect specular effects like caustics that appear when bright light is reflected on highly specular objects like in a metal ring on a table.

In the last case, represented by ray 3, the ray hits the environment before the virtual object ($d_{\text{Obj}_3} > d_{\text{Env}_3}$) so $\Delta L(\mathbf{x},\omega_i)$ is also zero. Note that these kinds of early occlusions are the main source of artifacts when using impostor tracing. False positive and false negative visibility tests are likely, as the correct ray directions might not be found or not available.

Depending on the technique and data structure, this ray casting is implemented differently, but the concept is the same in all cases.

6.5 Extension to Cone Casts

To speed up the ray tests and to reduce the noise, multiple rays can be handled in bundles called *cones*. Compared to a ray in Figure 7 a cone can cover multiple cases at once. Assume ray 1 and ray 2 are the boundaries of such a cone. Then the environment is visible to a degree $\alpha \in [0,1]$ and the object's influence is weighted with $1-\alpha$. With

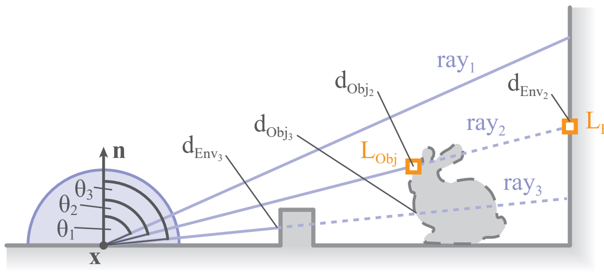

Fig. 7. Estimating the difference in incoming radiance ΔL for different directions based on the ray distance d. Three cases can be identified.

$\overline{L}_{\text{Obj}}$ and $\overline{L}_{\text{Env}}$ representing the average over the cone, the differential radiance becomes:

$$\Delta L(\mathbf{x},\boldsymbol{\omega}_i) = (1-\alpha)(\overline{L}_{\text{Obj}} - \overline{L}_{\text{Env}}). \quad (6)$$

To obtain $\overline{L}_{\text{Env}}$ we use the mipmap levels of the environment representations. For VCT and EM we are following the known implementations in [3, 11] and [2] respectively. In case of DIT, the search for the corrected central ray of the cone is performed on the most detailed level of the DI. Only the final texture fetch to gather the radiance is done in a coarser level, which depends on the size of the solid angle and the distance to the surface.

For $\overline{L}_{\text{Obj}}$ we start and end cone marching at the boundaries of the object's bounding box and use a cone radius dependent step size as well as radius dependent mipmap layers from the voxelization. Each voxel contains the probability τ for a surface intersection. If the probability τ is greater than zero, the object radiance is estimated using one of the approaches explained in Sec. 6.4 (ray 2 case). The result is weighted with the probability and the cone marching is proceeded. The following steps must be weighted with the probability of continuation α which is the product $\prod_s(1-\tau_s)$ of all previous steps. These previous steps can contain the occlusion between cone origin and object (ray 3 case). Finally, when the cone left the object volume, α is the weight for the environment as given in Equation (6).

6.6 Composition

During composition, the differential image and the virtual object are added to the camera image in reference color space. Then, the inverse color correction c_t^{-1} (see Sec. 4) is applied to the augmented image to transform it back into the input color space.

Figure 8 shows results and the capability of the compensation. While the image in the top center shows the augmented original image, we altered the input image by changing the brightness and introducing a strong color shift. Applying the inverse operation to the augmented image also transforms the colors of the virtual object which results in a coherent integration of the new content.

7 Illumination Estimation using Sampling

The integral of a function $f(x)$ can be estimated by a stochastic process called *Monte Carlo* Integration [21]:

$$\int_a^b f(x)\,\partial x \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(x_i)}{p(x_i)}, \quad (7)$$

where $p(x_i)$ is the probability to produce sample x_i. Any probability density function (PDF) p can be used, but having a function which scales with f yields the fastest convergence. Using such an improved distribution function is called importance sampling. For rendering the virtual object, Equation (3) is integrated on the GPU using the BRDF as PDF, as described in [2].

For the material estimation, we need to solve Equation (5). Since $L(\boldsymbol{\omega})$ is unknown we use the material-based cosine distribution $p(\boldsymbol{\omega}) = {}^1/_\pi \cos\theta$. Here, θ is the angle between the surface normal of the

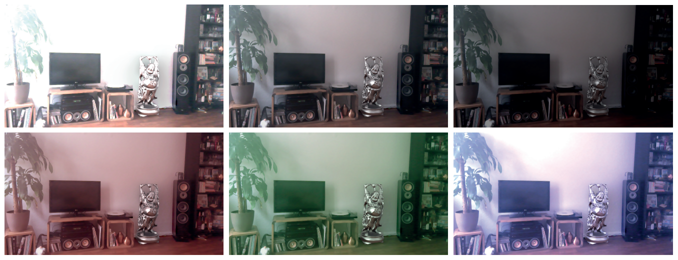

Fig. 8. Transforming back into input color space. *Top center*: original input image augmented by a Buddha statue. *Others*: artificially altered input images that have been transferred automatically into the reference color space and eventually, after the augmentation, back into the input space.

G-Buffer and the produced direction of $\boldsymbol{\omega}$. Applying Monte Carlo integration gives:

$$\rho \approx \pi \frac{L(\mathbf{x})}{\frac{1}{N}\sum_{i=1}^{N}\frac{L(\mathbf{x},\boldsymbol{\omega}_i)\cos\theta}{^1/_\pi\cos\theta}} = \frac{L(\mathbf{x})}{\frac{1}{N}\sum_{i=1}^{N}L(\mathbf{x},\boldsymbol{\omega}_i)}.$$

Applying the same scheme to Equation (4) while sampling the differential values $\Delta L(\mathbf{x},\boldsymbol{\omega}_i)$ directly (see Sec. 6.4) yields the estimation for the differential rendering:

$$\Delta L(\mathbf{x},\boldsymbol{\omega}_o) \approx \frac{1}{N}\sum_{i=1}^{N}\frac{\Delta L(\mathbf{x},\boldsymbol{\omega}_i)\frac{\rho}{\pi}\cos\theta}{\frac{1}{\pi}\cos\theta} = \frac{\rho}{N}\sum_{i=1}^{N}\Delta L(\mathbf{x},\boldsymbol{\omega}_i). \quad (8)$$

Further, we can improve the sampling by incorporating the knowledge that $\Delta L(\mathbf{x},\boldsymbol{\omega}_i)$ is zero when the ray misses the object (ray 1 case in Figure 7), which is related to observations in [39]. Instead of just skipping these rays, we try to keep to GPU utilization high by tracing other more promising rays. This influences the probability which must be corrected, as explained in the following.

First, we split the set of traced cones into two sets: a set $\mathcal{F}$ containing all cones that miss the object (ray 1 case) and a set $\mathcal{H}$ holding the remaining cones. Then, we divide $\mathcal{H}$ into disjoint sets $\mathcal{H}_a$ and $\mathcal{H}_b$ dependent on ray cases 2 and 3. Accordingly, Equation (8) can be split into three parts (omitting $\mathbf{x}$ for compactness):

$$\Delta L(\boldsymbol{\omega}_o) \approx \frac{\rho}{N}\left(\underbrace{\sum_{\boldsymbol{\omega}_i\in\mathcal{H}_a}\Delta L(\boldsymbol{\omega}_i)}_{L_{\mathcal{H}_a}} + \underbrace{\sum_{\boldsymbol{\omega}_i\in\mathcal{H}_b}\Delta L(\boldsymbol{\omega}_i)}_{L_{\mathcal{H}_b}} + \underbrace{\sum_{\boldsymbol{\omega}_i\in\mathcal{F}}\Delta L(\boldsymbol{\omega}_i)}_{L_{\mathcal{F}}=0}\right). \quad (9)$$

By using early intersection tests with the virtual object's bounding box, we can reject all samples in $\mathcal{F}$ before starting expensive tracing. Consequently, the number of samples in $\mathcal{H}$ is decreasing with the distance to the virtual object. To keep the GPU utilized, we increase N and create a large set of N directions, test them against the bounding box and record them for later tracing in a local array of size $|\mathcal{H}_a| < N$. In the experiments $|\mathcal{H}_a|$ is usually 16 or 32 and is referred to as the cache size. Now, it is possible that more than $|\mathcal{H}_a|$ valid samples are found for certain pixels. In that case, $|\mathcal{H}_a|$ directions are recorded randomly and the remaining $|\mathcal{H}| - |\mathcal{H}_a|$ are added to set $\mathcal{H}_b$.

$L_{\mathcal{H}_a}$ and $L_{\mathcal{H}_b}$ from Equation (9) are estimates of the same integral. Hence, we can discard all samples in $\mathcal{H}_b$ if $L_{\mathcal{H}_a}$ is scaled accordingly. I.e., we assume that on average the samples in $\mathcal{H}_a$ and $\mathcal{H}_b$ fetch the same values. The final equation then becomes:

$$\Delta L(\mathbf{x},\boldsymbol{\omega}_o) \approx \left(1+\frac{|\mathcal{H}_b|}{|\mathcal{H}_a|}\right)\frac{\rho}{N}\sum_{\boldsymbol{\omega}_i\in\mathcal{H}_a}\Delta L(\mathbf{x},\boldsymbol{\omega}_i).$$

Consequently, we maximize the use of a limited number of traced samples with a joint importance sampling of the material (the cosine distribution) and the object. Still, our sampling can produce weak

Table 1. Performance of the Acquisition Phase. Timings measured using a Google Tango Developer Kit. As in all examples, the point cloud size is limited to 4M environment samples. Depth information is only provided every 5th frame causing the steps marked by $(\cdot)^*$.

Computation Step	Time in ms
Get Color Image	6.3
Update c_t + Apply c_t	6.0 + 2.5
G-Buffer Splatting + Push-Pull	28.0 + 2.5
Get Sparse Depth Image	4.0*
Find nearest Neighbors	25.8*
Normal Estimation	18.8*
Add Samples to Point Cloud	8.9*
Remove Samples from Point Cloud	5.3*
Apply c_t^{-1}	2.5
Others (Visualization, ...)	13.8
(Averaged) Total Frame	**(73.4) 123.6***

Table 2. Performance of the Rendering Phase. Timings are measured in ms for the *toy car in office* scene. All three quality settings are shown in Figure 1. *Med* is also used in the video.

Technique	Tango EM	Tango EM	Tango DIT	PC DIT	PC VCT	PC DIT
Quality Settings	low	med	med	med	med	high
Get Color Image	5.8	7.2	6.2			
Update c_t	7.2	10.3	7.8	1.1	1.1	1.2
Apply c_t	3.0	2.5	2.7	0.1	0.1	0.1
G-Buffer Splatting	34.7	30.5	30.5	8.1	8.1	8.3
G-Buffer Push-Pull	2.6	1.8	1.8	0.4	0.4	0.4
Render Virtual Object	18.7	149.6	195.0	5.1	8.6	133.2
Material Estimation	45.7	45.5	45.4	1.1	1.2	18.6
Diff. Background	22.5	32.5	61.7	0.8	2.0	16.9
Apply c_t^{-1}	3.0	2.5	2.7	0.1	0.1	0.1
Others (Compose, ...)	4.4	4.5	4.1	0.2	0.4	0.5
Total Frame	**147.6**	**288.5**	**357.9**	**16.9**	**21.9**	**179.0**
Frame Rate [Hz]	**6.7**	**3.4**	**2.8**	**59.1**	**45.6**	**5.6**

results if $\Delta L(\mathbf{x},\omega_i)$ varies a lot. This happens if only a few samples hit very bright regions in the environment map behind the object or if existing bright regions are missed completely.

8 Results and Discussion

Here, we discuss the results of the different stages in the order of the previous sections. To evaluate the presented approach, we investigated all stages of the pipeline in real world scenarios and in synthetic scenes acquired by a simulated depth sensing device. Hence, even the synthetic experiments perform all steps of the pipeline. To provide insights on the performance of the system, we show results created on a desktop PC with an Intel i7-4790S and a NVIDIA GeForce GTX 980. The mobile device used is a Google Tango Developer Kit powered by an NVIDIA Tegra K1.

Environment Acquisition. As the synthetic scenes showed no issues, we are focusing only on real data. Therefore, we mainly refer to the accompanying video. Visual results can also be found in Figures 2 and 3. The depth sensor provides between 3000 and 14000 new depth samples every 5th frame in which the complete acquisition pipeline is executed. During the other frames, we also provide feedback to the user by showing the point cloud from the current camera position. The timings provided in Table 1 show that the average frame time, including frames with and without depth sensing, is 73.4 ms which equals about 13.5 fps.

While larger errors in the position of the samples are mainly caused by the given tracking, we concentrate on issues with the measured radiance. Using our compensation estimation, we are able to measure the radiance in the scene relative to the first frame. The compensation provides a stable color adjustment in consecutive frames but as known from related algorithms like SLAM, there is a drift caused by error accumulation over time. This can be observed when closing the loop of 360°. We observed increasing errors in scenes where the camera gets over-saturated by bright light sources or under-saturated in very dark areas causing noisy images. Another issue we identified is vignetting, a radial reduction of brightness in the input. Our compensation is not able to deal with local effects.

However, the strongest discrepancies are due to the Lambertian material assumption as real surfaces usually show reflections and highlights. As the effects did not appear in the synthetic experiments, we can assume that there is no systematic error even though we are not able to provide a real-world ground truth comparison.

G-Buffer Generation. Splatting the entire point cloud every frame is one of the bottlenecks not addressed so far. Besides the computation time (see Tables 1 and 2), there are also qualitative issues. Splatting densely sampled areas can lead to z-fighting, which is acceptable for colors but leads to temporal incoherence in the depth and normal channels. Applying bilateral Gaussian filters or down-sampling solved the problem in our experiments. We used 320×180 as G-Buffer resolution in all experiments. Sparsely sampled areas, e.g., because of glossy surfaces, are smoothly filled by the push-pull steps. Large gaps however, especially at the viewport boundaries can lead to serious artifacts. Even in moderate cases the surface is bumpy. Figure 1 (left) shows bright spots close to the papers that are not shadowed. This is an important point for further research. Note that the proposed normal estimation could also improve the G-Buffer but at a high cost.

Scene Representation. Splatting the point cloud into a distance impostor follows the generation of the G-Buffer. The timings are similar and scale linearly with the number of pixels. However, DIs are only updated when the virtual object is moved and computations can be distributed over multiple frames. Hence, temporal coherence can only become a problem if the object is moved. If not mentioned otherwise, we use a resolution of 6×256^2 for DI or (scene) voxel volumes of size 256^3.

Rendering of the Virtual Object. For performance evaluation, we refer to Table 2. The scene with the toy car, visible in the video and Figure 1, is rendered using different quality settings. *Low* and *med* use 16 samples for material estimation, differential rendering (with cache size 8) and rendering of the virtual object. In *high* settings, the sample counts are increased to 64 for material estimation as well as differential rendering (with cache size 32) and to 128 for rendering the virtual object. The toy car consists of 230k triangles and is voxelized into a volume of size 32^3 for *low* and *med* but $128 \times 96 \times 96$ for *high*. The *low*-quality setting additionally omits self-shadow tests. With *med* settings, we are checking for visibility only and for *high* we trace a secondary bounce. Compared to visibility test only, the secondary bounce takes approximately twice as long for rendering the virtual object. Note that for the material estimation DIT is also used in EM cases, as EM cannot be applied here. All measured timings are reasonable but show that the mobile device is barely able to reach interactive frame rates even with low quality.

Figure 9 shows the Buddha model in a synthetic test scene which is used for qualitative comparison of the alternative techniques. We evaluate a diffuse material as they reflect light from all directions and tend to show more noise in the sampling. The results show, that self-shadows appear significantly more plausible in all three approaches, but also increase timings notably (see Table 3). Secondly, the simple EM technique, which assumes distant light, illuminates the object uniformly from top to bottom. In both other techniques, the origin of the ray or cone is considered which leads to a more correct variation in brightness. Table 3 contains the timings for different settings, where gray cells correspond to the images in Figure 9.

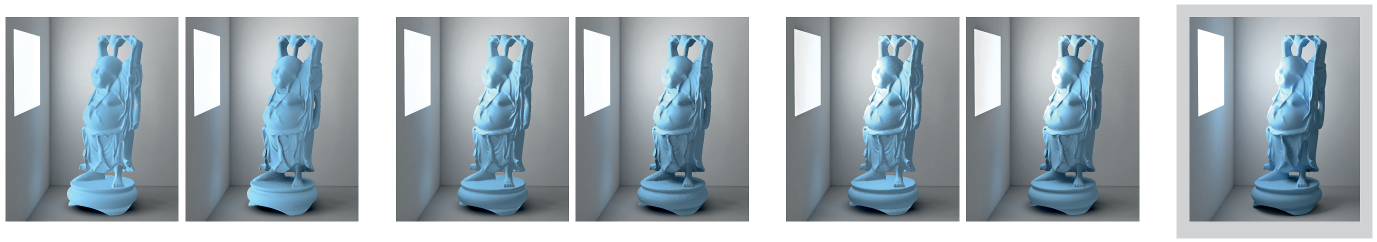

Fig. 9. Illumination Variants for the Virtual Object. *Groups from left to right:* Environment Mapping, Impostor Tracing, VCT and the reference. Each group shows the results without and with self-occlusion test for the object illumination. Timings can be found in Table 3.

Table 3. Environment Query Timings. Accumulated time in ms for Rendering Virtual Object, Material Estimation and Differential Background (with cache size 32) on PC (also see Figure 9).

Sample Count (N)	16	32	64
EM (no visibility test)	0.27	0.48	0.99
EM	9.88	20.49	42.12
DIT (no visibility test)	3.57	6.91	13.82
DIT	12.81	25.81	51.86
VCT (no visibility test)	6.43	12.14	27.01
VCT	18.97	37.72	79.24

Table 4. Computing Virtual Influence. Timings in ms for different number of samples and cache sizes to estimate the differential rendering integral using EM, DIT and VCT on PC (also see Figure 11).

Sample Count (N)		8	16	32	64	128
EM	(Cache 8)	2.15	2.62	3.33	3.85	4.46
EM	(Cache 16)	2.16	3.10	4.90	6.44	7.36
EM	(Cache 32)	2.19	3.11	5.88	9.51	12.48
DIT	(Cache 8)	2.25	3.27	4.16	4.67	5.51
DIT	(Cache 16)	2.74	3.86	6.05	7.87	9.86
DIT	(Cache 32)	2.76	3.85	7.14	11.56	15.50
VCT	(Cache 8)	8.37	3.98	5.10	5.73	6.49
VCT	(Cache 16)	8.34	4.71	7.56	9.92	11.10
VCT	(Cache 32)	8.45	4.71	8.77	14.57	19.23

Material Estimation. We assumed a Lambertian environment which allows to integrate the incident light over the upper hemisphere of the surface. Since this assumption is not valid in real world scenarios we are not able to achieve perfect results. Another problem is caused by an incomplete acquisition of the environment. Windows or bright sources of light cannot be measured by the Tango device. Therefore, we added a workaround to add guessed environment samples. Triggered by a manual action of the user (see the accompanying video), the current camera image is projected along the view direction and samples are created at the distance estimated by the hole-filled depth buffer. These samples are excluded from the compensation estimation but are used for rendering and material estimation. Since these samples are often created in saturated areas of the camera image, the real – often much brighter – radiance is underestimated. Hence, the material estimation will be overestimated and shadows computed by differential rendering appear different.

Figure 10 shows the entire pipeline, including the estimated material parameters as well as the resulting difference image. Besides the color shift on the right side of image, the material still contains some shadows but removed the shading to a large extent. Ideally, the material image should not show any lighting or shading.

Differential Background Rendering. To evaluate the relighting of the background, we measured different timings with varying parameters in the synthetic scene (see Table 4 and Figure 11). The images show an equal time comparison of EM, DIT and VCT. The resolution for the differential image is reduced by a factor 2 to 640×360. One bounce of indirect light is computed in case the virtual object was hit.

Obviously, none of the techniques can reproduce the reference solution. VTC and DIT at least provide contact shadows, whereas EM yields a uniform directional shadow because of the distant light assumption. Even though this is incorrect, the resulting shadow is smooth and could be used as fast approximation. Increasing the number of cones and reducing their radius generally improves the visual appearance.

Because of the diffuse assumption and the low number of cones, the probability to hit a light source is low and the cone radius must be large to properly sample the entire hemisphere. Hence, we are not able to reproduce high frequency shadow details. Compared to the ground truth, the direction and the coarse shape are correct though. In other rendering engines, VCT is used for indirect light only. Therefore, 5 to 8 samples are often considered as sufficient. We also handle the direct illumination. In the future, hybrid solutions with extracted light sources can be desirable. However, extracting them from the point cloud is not straightforward.

Table 4 also shows the impact of the presented caching approach that allows to select sample directions not only based on the BRDF but also on the direction of the virtual object (see Sec. 7). The measurements confirm an increase of the utilization and thereby a reduction of cost per sample. The costs increase as long as the cache contains less than $|\mathcal{H}_a|$ samples (see Equation (9)). Afterwards, the costs per sample equal the cost of the box test. For temporal coherence, we refer to the accompanying video showing the moving Buddha and DIT with $N = 64$ and a cache size of 32.

Interreflections between the real world and inserted virtual objects are another important feature of coherent AR rendering. By computing a secondary bounce with any of the three techniques, we are able to produce color-bleeding, e.g., on the wall in Figure 9.

Composition. For visual results and an impression of the visual behavior we refer the reader to the accompanying videos. During the acquisition of the environment the colors are matched frame by frame. In the sequence containing the toy car, a visually plausible, automatic adaptation to the camera exposure can be observed (also see Figure 1 *tc* and *bc*). Figure 8 shows that the approach is also robust against different color transformations on the input. Therefore, our compensation is of great use in many AR scenarios.

However, if the input image is already saturated, problems will occur. In this case the transformation leads to undefined colors in the corrected image, but the material estimation relies on them. Figure 10 shows the effect on the resulting reflectance coefficients. Since the resulting values are stored in HDR, this is not visible to the user as long as these areas are not in the shadow of an object.

Timings for updating c_t can vary strongly (see Table 1 and 2). This is because the timing contains a read-back from the GPU to disable acquisition in case of failures (shared code for rendering). Setting up and solving the system on the GPU itself requires about 2.6 ms on the Tango and only depends on the G-Buffer resolution.

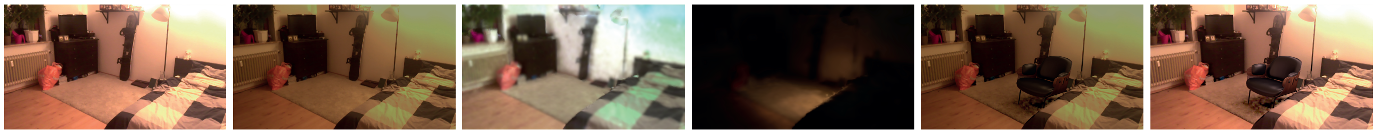

Fig. 10. Compensated Differential Rendering. *left to right*: Camera image in unknown input color space, camera image transformed in HDR reference space, estimated reflectance coefficients (contain errors because of saturation in the input image), differential image containing the influence of the virtual object (absolute value), augmented image in HDR reference space, augmented image in unknown input color space (model courtesy of Van Quang Ho).

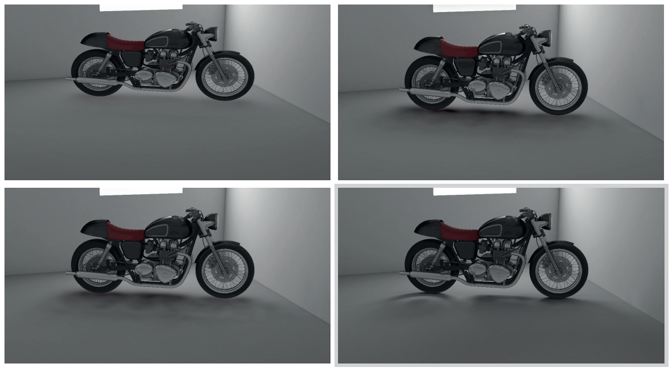

Fig. 11. Differential Shadows achieved within 10 ms of differential background computation: EM *(top left)*, DIT *(bottom left)*, VCT *(top right)* and ground truth *(bottom right)*. Table 4 contains the corresponding sample counts, cache sizes and timings.

9 Limitations and Future Work

To be able to demonstrate a coherent AR rendering starting from the acquisition using a single mobile device, we need to make a set of assumptions that are not necessarily true in real world scenarios.

The compensation estimation relies on a linear model to map input images into a given reference color space. In our experiments, this approach was sufficiently robust even in difficult situations, e.g., when turning the camera directly to the light. However, for other devices or different applications, a linear compensation is not adequate, e.g., if the transformation is not constant over the entire image. Further, we are limited by the dynamic range of the Tango. When pointed to a bright light source, the exposure is reduced to not over-saturate. When this turns the rest of image too dark or even black, the matching must fail.

The most restrictive assumption is the Lambertian environment BRDF. This affects the acquisition process, where the captured environment is used as input to the color estimation in consecutive steps. Reflections on non-diffuse surfaces introduce errors that cannot be handled by the system at the moment. Also, the material estimation required for differential rendering depends on this assumption. Including additional information about the direction of measurement could help to overcome this problem by measuring more complex material properties. Further, the sampling quality will benefit from smaller cones due to more focused reflections.

Although we are able to remove samples from the point cloud, the environment is assumed to be static after completing the acquisition. During the augmentation, the acquisition pipeline could update the environment model, but this only works for the camera's small field of view. By simply turning a light source on or off, the entire reconstruction becomes outdated. Measuring the illumination with additional hardware or inverse rendering techniques could be used here.

As noticed earlier, the reconstructed surface can be very bumpy. This can be addressed by taking the previous acquired scene into account while processing new samples, e.g., by applying ICP.

The presented rendering techniques allow different levels of quality. However, the tablet used in the experiments achieves barely interactive frames rate, even with low settings. There is a lot of potential in optimizing the performance, as we aimed for flexibility instead of polishing a single technique. However, different bottlenecks can be addressed to achieve higher frame rates. Processing and optimizing the acquired scene, by reducing samples in dense regions, creating hierarchical structures that allow culling and fast traversal, estimating reflectance coefficients offline or creating a low poly triangle mesh for instance.

At this point, only one virtual object can be inserted into the image. Adding multiple objects with multiple voxel volumes also multiplies the computation effort. Using a hierarchy of boxes that supports importance sampling is a challenging task for the future.

10 Conclusion

We demonstrated that photorealistic AR is made possible by a single mobile device. First, we presented a linear estimation of the unknown color adjustments applied by the mobile camera and the driver. This allows to fuse the LDR color information of consecutive frames into HDR samples. These are stored in a point cloud along with fitted surface normals.

We applied several-sampling based techniques for high quality renderings based on the acquired data. While aiming for interactive augmentations on mobile devices, we presented a framework that scales with the performance of the available platform up to high quality renderings, as shown in Figure 1. Note that progressive rendering to produce high quality screenshots, e.g., for sharing, is a straight forward extension.

Eventually, the composed image is transformed back into input color space. The result is an augmented image containing a virtual object, coherently illuminated by the environment. The virtual and real objects are integrated in terms of perspective, occlusion, shading and the original color transformation of the camera, including exposure and white balance.

Further, it is also possible to use this technique with different devices involved. So, the color compensation is not limited to be used with depth sensing devices. As an example, the background of Figure 1 (left) was captured with a DSRL camera.

Thinking of a high-quality scan of an area of interest – maybe a historic sight – a common smartphone can display augmentations based on that scan. Therefore, extrinsic camera parameters can be estimated from visual features and the presented estimation method is used to map the camera image into the color space of the high-quality scan. After applying the augmentations, the inverse compensation is used to transform back into the unknown color space that was optimized for displaying the real content.

Acknowledgments

We would like to thank the Stanford 3D Scanning Repository for the Stanford Bunny and the Buddha model. We also would like to thank VILAC for the Toy Car, Maciek Ptaszynski for the Cafe Racer and Van Quang Ho for the Low Lounger model. Additionally, we would like to thank the anonymous reviewers for valuable comments and suggestions. This work is partially supported by the German Research Foundation (DFG), Grant No. GR 3833/3-1.

References

[1] E. H. Adelson and J. R. Bergen. The Plenoptic Function and the Elements of Early Vision. In *Computational Models of Visual Processing*, pp. 3–20. Massachusetts Institute of Technology, 1991.

[2] M. Colbert and J. Křivánek. GPU-based importance sampling. *GPU Gems 3*, pp. 459–476, 2007.

[3] C. Crassin, F. Neyret, M. Sainz, S. Green, and E. Eisemann. Interactive Indirect Illumination Using Voxel Cone Tracing. *Computer Graphics Forum (Proc. Pacific Graphics)*, 30(7):1921–1930, 2011.

[4] C. Dachsbacher and M. Stamminger. Reflective Shadow Maps. In *Proc. Symposium on Interactive 3D Graphics and Games (I3D)*, pp. 203–231, 2005.

[5] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt. BundleFusion: Real-Time Globally Consistent 3D Reconstruction Using On-the-Fly Surface Reintegration. *ACM Transactions on Graphics (TOG)*, 36(3):24:1–24:18, 2017.

[6] 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 SIGGRAPH '98*, pp. 189–198, 1998.

[7] P. Debevec and J. Malik. Recovering High Dynamic Range Radiance Maps from Photographs. In *Proceedings of SIGGRAPH '97*, pp. 369–378, 1997.

[8] E. Eisemann and X. Décoret. Single-pass GPU Solid Voxelization for Real-time Applications. In *Proc. Graphics Interface (GI)*, pp. 73–80, 2008.

[9] A. Fournier, A. S. Gunavan, and C. Romanzin. Common Illumination between Real and Computer Generated Scenes. In *Proc. Graphics Interface (GI)*, pp. 254–262, 1993.

[10] T. A. Franke. Delta Light Propagation Volumes for Mixed Reality. In *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 125–132, 2013.

[11] T. A. Franke. Delta Voxel Cone Tracing. In *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 39–44, 2014.

[12] S. Gibson, J. Cook, T. Howard, and R. Hubbold. Rapid Shadow Generation in Real-world Lighting Environments. In *Proc. Eurographics Symposium on Rendering (EGSR)*, pp. 219–229, 2003.

[13] D. B. Goldman. Vignette and Exposure Calibration and Compensation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 32(12):2276–2288, 2010.

[14] T. Grosch. PanoAR: Interactive Augmentation of Omni-directional Images with Consistent Lighting. In *Proc. Computer Vision / Computer Graphics Collaboration Techniques and Applications (Mirage)*, pp. 25–34, 2005.

[15] T. Grosch, T. Eble, and S. Müller. Consistent Interactive Augmentation of Live Camera Images with Correct Near-field Illumination. In *Proc. ACM Symposium on Virtual Reality Software and Technology (VRST)*, pp. 125–132, 2007.

[16] L. Gruber, T. Richter-Trummer, and D. Schmalstieg. Real-time Photometric Registration from Arbitrary Geometry. In *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 119–128, 2012.

[17] L. Gruber, J. Ventura, and D. Schmalstieg. Image-space illumination for augmented reality in dynamic environments. In *Proc. IEEE Virtual Reality (VR)*, pp. 127–134, 2015.

[18] M. Grundmann, C. McClanahan, S. B. Kang, and I. Essa. Post-processing Approach for Radiometric Self-Calibration of Video. In *IEEE International Conference on Computational Photography (ICCP)*, pp. 1–9, 2013.

[19] T. Hachisuka and H. W. Jensen. Parallel Progressive Photon Mapping on GPUs. In *ACM SIGGRAPH Asia Sketches*, p. 54, 2010.

[20] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera. In *Proc. ACM Symposium on User Interface Software and Technology*, pp. 559–568, 2011.

[21] J. T. Kajiya. The Rendering Equation. *Computer Graphics (Proc. SIGGRAPH)*, 20(4):143–150, 1986.

[22] P. Kán. Interactive HDR Environment Map Capturing on Mobile Devices. In *Proc. Eurographics Short Papers*, pp. 29–32, 2015.

[23] P. Kán and H. Kaufmann. Differential Irradiance Caching for fast high-quality light transport between virtual and real worlds. In *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 133–141, 2013.

[24] P. Kán and H. Kaufmann. Differential Progressive Path Tracing for High-Quality Previsualization and Relighting in Augmented Reality. In *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 133–141, 2013.

[25] L. Kavan, P.-P. Sloan, and C. O'Sullivan. Fast and Efficient Skinning of Animated Meshes. *Computer Graphics Forum (CGF)*, 29(2):327–336, 2010.

[26] S. J. Kim and M. Pollefeys. Robust Radiometric Calibration and Vignetting Correction. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 30(4):562–576, 2008.

[27] M. Knecht, C. Traxler, O. Mattausch, W. Purgathofer, and M. Wimmer. Differential Instant Radiosity for Mixed Reality. In *Proc. IEEE International Conference on Computer Vision (ICCV)*, pp. 99–107, 2010.

[28] M. Knecht, C. Traxler, W. Purgathofer, and M. Wimmer. Adaptive Camera-Based Color Mapping For Mixed-Reality Applications. In *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 165–168, 2011.

[29] J. Kronander, F. Banterle, A. Gardner, E. Miandji, and J. Unger. Photorealistic rendering of mixed reality scenes. *Computer Graphics Forum (Proc. Eurographics)*, 34(2):643–665, 2015.

[30] P. Lensing and W. Broll. Instant indirect illumination for dynamic mixed reality scenes. In *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 109–118, 2012.

[31] R. Marroquim, M. Kraus, and P. R. Cavalcanti. Efficient Point-Based Rendering Using Image Reconstruction. In *Proc. Symposium on Point-Based Graphics*, pp. 101–108, 2007.

[32] M. Meilland, C. Barat, and A. Comport. 3D High Dynamic Range dense visual SLAM and its application to real-time object re-lighting. In *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 143–152, 2013.

[33] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3D Reconstruction at Scale using Voxel Hashing. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, 32(6):169:1–169:11, 2013.

[34] D. Nowrouzezahrai, S. Geiger, K. Mitchell, R. Sumner, W. Jarosz, and M. Gross. Light Factorization for Mixed-Frequency Shadows in Augmented Reality. In *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 173–179, 2011.

[35] G. Patow and X. Pueyo. A Survey of Inverse Rendering Problems. *Computer Graphics Forum (CGF)*, 22(4):663–687, 2003.

[36] T. Richter-Trummer, D. Kalkofen, J. Park, and D. Schmalstieg. Instant Mixed Reality Lighting from Casual Scanning. In *IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 27–36, 2016.

[37] K. Rohmer, W. Büschel, R. Dachselt, and T. Grosch. Interactive Near-Field Illumination for Photorealistic Augmented Reality on Mobile Devices. In *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 29–38, 2014.

[38] K. Rohmer, W. Büschel, R. Dachselt, and T. Grosch. Interactive Near-Field Illumination for Photorealistic Augmented Reality with Varying Materials on Mobile Devices. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 21(12):1349–1362, 2015.

[39] K. Rohmer and T. Grosch. Tiled Frustum Culling for Differential Rendering on Mobile Devices. In *Proc. IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 37–42, 2015.

[40] T. Saito and T. Takahashi. Comprehensible Rendering of 3-D Shapes. *Computer Graphics (Proc. SIGGRAPH)*, 24(4):197–206, 1990.

[41] D. Schmalstieg and T. Höllerer. *Augmented Reality: Principles and Practice*. Addison-Wesley Professional, 1st ed., 2016.

[42] L. Szirmay-Kalos, B. Aszódi, I. Lazányi, and M. Premecz. Approximate Ray-Tracing on the GPU with Distance Impostors. *Computer Graphics Forum (CGF)*, 24(3):695–704, 2005.

[43] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. J. Leonard, and J. McDonald. Real-time Large Scale Dense RGB-D SLAM with Volumetric Fusion. *International Journal of Robotics Research (IJRR)*, 34(4-5):598–626, 2015.

[44] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger. ElasticFusion: Real-Time Dense SLAM and Light Source Estimation. *International Journal of Robotics Research (IJRR)*, 35(14):1697–1716, 2016.

[45] E. Zhang, M. F. Cohen, and B. Curless. Emptying, Refurnishing, and Relighting Indoor Spaces. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, 35(6):1–14, 2016.