Photorealistic Rendering for Augmented Reality

A Research and Development Project Report
by

Rishabh Shah
150050006

under the guidance of
Prof. Parag Chaudhuri

Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Mumbai 400076 (India)
2018

Abstract

Photorealistic Rendering for Augmented Reality (AR) is concerned with seamlessly blending the virtual world and real world in real time. In this project not only do we create a self-suficient application for realtime AR rendering, but we also explore ways of making these augmentations more photorealistic.

The aim of this project is to build an application which can use any monocular camera output and augment synthetic meshes on our feed in realtime. Our focus is to make the application user friendly and encapsulate all the fussy details. This report starts with building a basic video processing application, designing user interface for various interactions followed by rendering an augmented version of the input camera feed. Finally, we dig into various ways of making the augmentations photorealistic.

Acknowledgements

I wish to sincerely thank my guide Prof. Parag Chaudhuri for his guidance and unwavering support, without whom this endeavor would have remained fruitless.

Rishabh Shah
IIT Bombay
6 May 2018

Table of Contents

Chapter 1

Introduction

1.1 Augmented Reality

With the advent of GPUs and other improvements in computation speeds, Augmented Reality has become feasible in realtime. Augmented reality holds the promise of creating direct, automatic, and actionable links between the physical world and electronic information. With so many applications ranging from the crucial military and medical fields to everyday use in shopping, games and education, AR has developed into a topic of broad and current interest in computer graphics. The main distinguishing feature aspect of AR over film is that these renderings are generated in realtime and are interactive.

1.2 Photorealistic rendering

Many of the applications require AR to be convincing enough for a spectator to not be able to differentiate between real and virtual. This requirement demands both the real and virtual worlds to be lit with same light models. This is what is also known as visual coherence.

The photorealistic part of the problem proves to be more difficult. Just from an image, we cannot be sure which combination of light color and surface material caused a particular color observation in an image. Thus recreating light model from a set of images proves to be a challenging task.

1.3 Objective and Scope

The problem is aggravated when the input is constrained to be just RGB images and output rendering is expected to calculated in realtime without any constraints in the environment. We look into ways to tackle these issues under the input constraints to create an application that can render synthetic meshes in realtime.

We allow the user to scan the environment (preprocessing phase), and add augmentations at appropriate places. Then as the user starts viewing the same environment from a different camera feed, the application recognizes the scene and augments the synthetic mesh (that the user had saved in preprocessing phase) at its appropriate position and orientation and can be viewed from any camera pose. Later, we explore methods to capture the light models from the RGB input images to coherently render the synthetic objects in the scene.

1.4 Report Outline

The rest of this report is organized as follows. In Chapter 2 we outline the existing methods related to our work and their limitations. Chapter 3 describes the details of implementation and user interface. We show the usage of our application, and results on test data in Chapter 4. Chapter 5 talks about various approaches to tackle the problem of coherent rendering for photorealistic AR. The report is concluded in Chapter 6.

Chapter 2

Related Work

A large body of work on both photorealistic and non-photorealistic rendering for AR exists. Augmenting synthetic objects involves retrieving user's camera pose for every moment. Also, we need some amount of geometry of the environment to place mesh objects relative to some plane surface. Coherence requires estimation of light models. Thus, three major challenges to be addressed for photorealistic AR are reconstruction of 3D features from group of images, retrieving camera pose for every frame of the video and realtime estimation of the current lighting in user's environment. Many other issues need to be seen for photorealistic AR like occlusion which are not discussed as part of this report. We address only the issue of coherent rendering issue for photorealistic AR here.

2.1 3D Reconstruction and Camera tracking

The goal of multi view 3D reconstruction is to infer geometrical structure of a scene captured by a collection of images. ORB-SLAM-2 by Mur-Artal *et al.*[6] had shown that using monocular cameras one can construct 3D sparse point cloud data in realtime. It is a complete SLAM system for monocular cameras, including map reuse, loop closing and relocalization capabilities. It also allowed for simultaneous tracking of camera pose in a sequential RGB image data. In this work, we use its efficient capabilities of 3D reconstruction and camera tracking for our preprocessing phase.

2.2 Visual Coherence

Common approaches rely on light probes, e.g., fisheye cameras, chrome spheres or other reference objects with known shape and reflectivity being present in the scene. An obvious downside of using a light probe is that it interferes with user's experience. As an alternative Gruber *et al.*[2] demonstrated a probe less method that estimates incident light purely from reflections scanned in real time using a RGB-D sensor. This approach forgoes artificial light probes, but suffers from high computational demands that currently cannot be run of mobile devices.Profound work on

this subject was presented by Richter-Trummer *et al.*[7] which delved upon a novel approach to recover both incident lighting and surrounding surface materials from casually scanned geometry via RGB-D sensor. But since our model is restricted to use of RGB data, one could not construct the lighting using just RGB input using this method.

With the advent of neural networks, many research papers were published trying to incorporate this approach for detecting natural scene light. One such approach was explored by Mandl *et al.*[5] It described an innovative approach to use CNNs to approximate the scene lighting in form of spherical harmonics (SH). The idea was that of using any known object in scene as a light probe and then creating synthetic data for the neural net using that probe and predict spherical harmonic coordinates to light the scene. The only problem was that using a known object adds constraints on the environment. And, constructing known object from a scene via Richter-Trummer *et al.*[7]'s method and training it again would interfere with the realtime constraints we have to abide by.

Another similar approach was done by Hold-Geoffroy *et al.*[3] wherein they described use of natural available data to train a CNN to predict lighting model in outdoor environment. The idea was that to use parameters from sky luminance distribution model as the output to be learned from images via a CNN. So the neural network served as a complex mapping between input image pixels and output illumination parameters. But this approach was constrained on outdoor environments. Such sky models won't be very much useful for indoor illumination.

None of these approaches meet all the requirements and constraints we need to fulfill in our attempt to create a photorealistic rendering for AR. However, all these approaches inspired us to come up with a novel idea for going about this problem that could be both realtime as well as use only RGB data. We discuss this idea in Chapter 5.

Chapter 3

Method

3.1 Overview

We divide our work into three phases. Preprocessing, User Interaction, and Augmentation Playback. In preprocessing stage, user feeds in a video (or group of images) and we reconstruct the entire scene from those images in form of sparse point clouds. In user interaction stage, user can play with the generated point cloud, add planes, upload meshes, position them, and perform various other actions. In the final playback phase, we replay the video to user with augmentations that were saved after interaction phase.

We also provide a live mode wherein the user once again gives video input but this time our method recognises the environment (the preprocessing phase had seen) to superimpose the scenes using the previously saved augmentations.

3.2 Preprocessing Phase

We use ORB-SLAM-2 library for the purposes of 3D reconstruction of the scene in the form of sparse point cloud. This point cloud is then used in the next phase for user to interact with the environment. User can see the points being tracked in the scene as the video progresses. This can be seen from small black dots in the Figure 3.1. User should try to avoid sudden movements which can lead to loss of track in the middle of scanning. In such cases one has to take the camera back to the last tracked position and wait for the system to reconfigure and start retracking. This phase is timer based i.e we capture the feed from camera for a fixed time interval.

3.3 User interaction Phase

In AR users want to add augmentations at some specific position in the environment. This stage takes care of these demands by providing a user-friendly interface to perform all these tasks. We provide interface for actions to add, remove and merge various planes to form a geometry. For

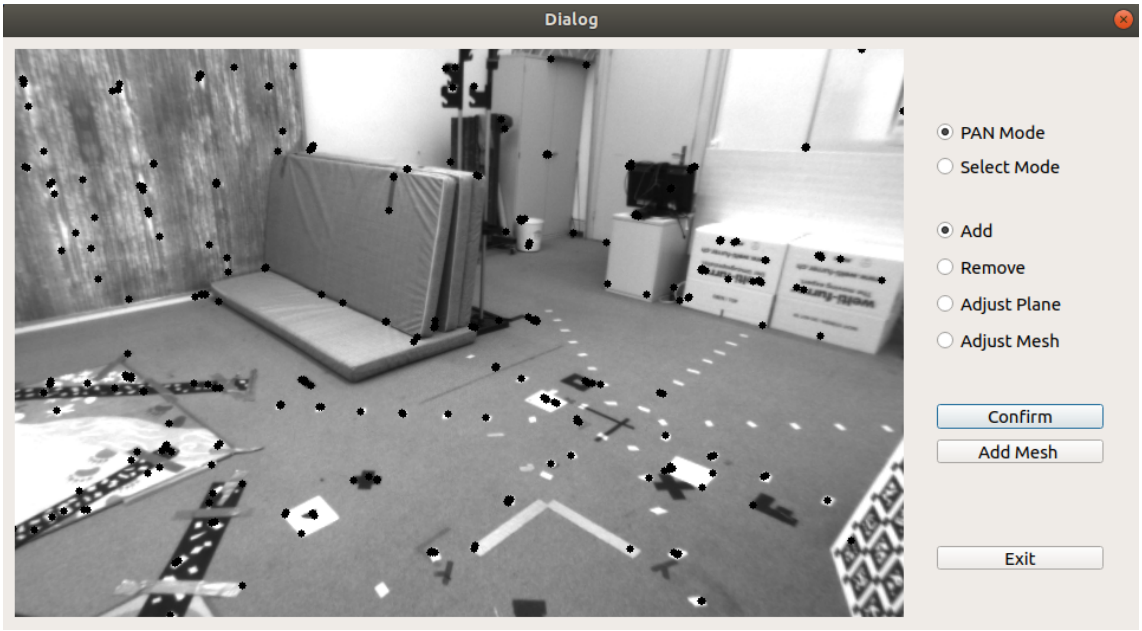


Figure 3.1: The Preprocessing Phase.

user to be able to access all parts of environment scanned, a traversal mechanism along some specific keyframes is available wherein the user sees the point cloud from various keyframes detected in the preprocessing stage.

Then when the user is happy with the geometry, a mesh can be uploaded and can be placed in the created geometry via our interface. After all the adjustments, user can save the augmentations.

3.4 Playback Phase

After the user is satisfied with the augmentations, playback phase kicks in. Here, we display the AR version of same input video where the user can see the saved augmentations in the real video as if these synthetic mesh augmentations are immersed inside that scene.

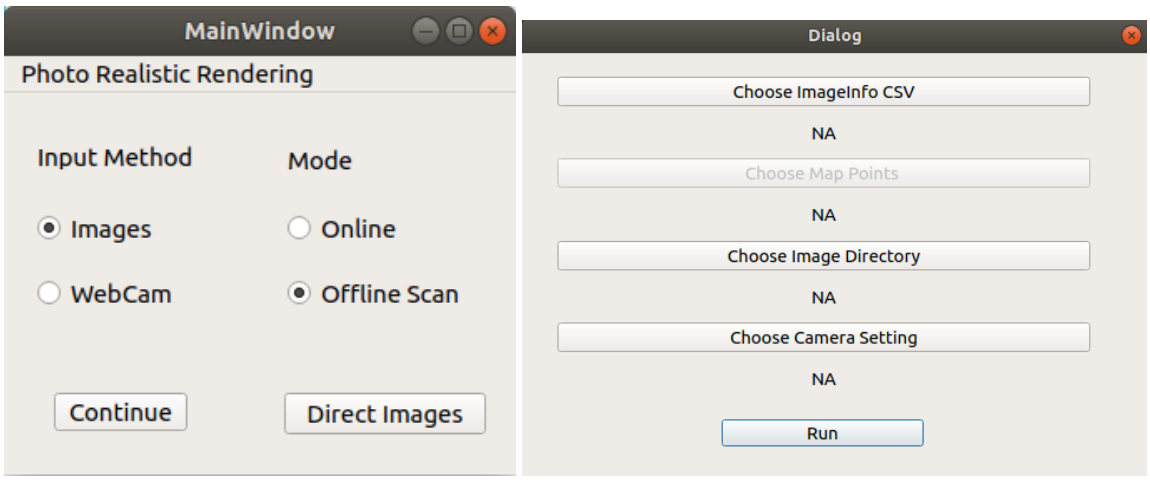
Chapter 4

Implementation and Results

4.1 Implementation Details

4.1.1 Libraries Used

We have designed our application for windows currently but simple translation can make it work on other devices. We have used Qt's open and modular framework to design out UI. All the coding has been done in C++. For preprocessing phase we have used API calls from Qt code to ORB-SLAM-2 library for tracking and reconstruction. We use many of OPENCV functions for working with images and webcam feed, and OPENGL methods for rendering the images over the UI widget window



(a) User application modes

(b) User Inputs Required

Figure 4.1: The UI for user input before the preprocessing phase.

4.1.2 User Interface Details

Our user interface offers two modes to user for working with the application. These are online and offline modes. Both the modes have two possible input options of either using specific

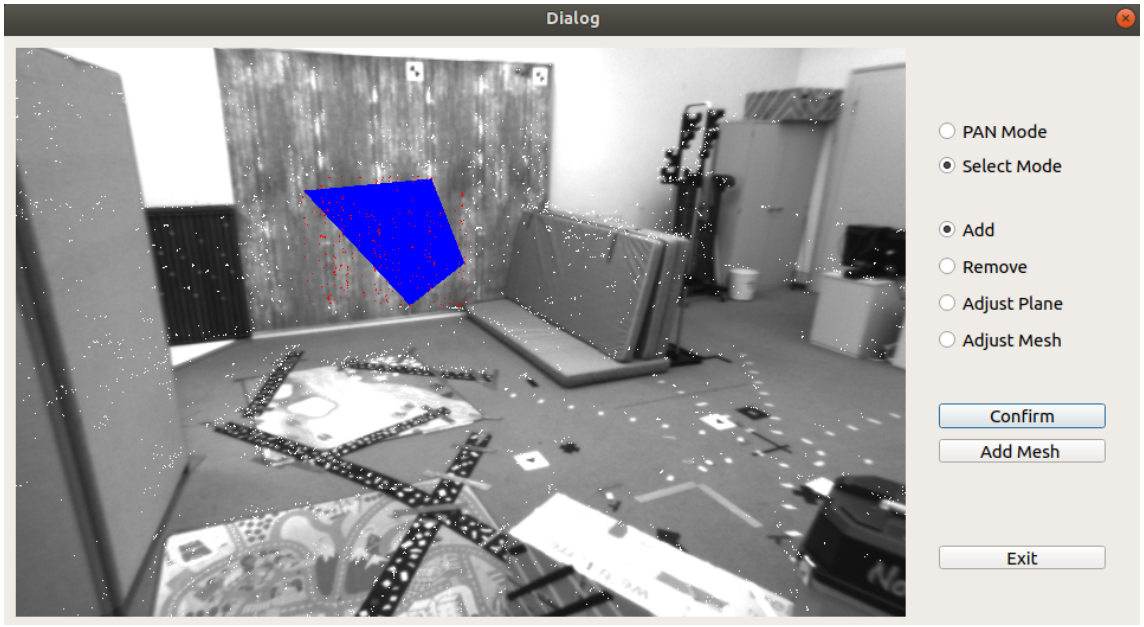


Figure 4.2: Adding and modifying Planes in the scene.

webcam feed or a bunch of images as input. Figure 4.1a shows the entry UI window wherein the user can select both the input type and the mode of usage. When a user presses "Continue" button, a dialog box for selection of input files phase appears as shown in Figure 4.1b.

Offline mode

This mode uses sequence of images as input. The application then performs the preprocessing phase, takes user to the interaction mode. In this mode, we have buttons for viewing scene either in select mode or pan mode. These and all other UI interaction elements can be seen on the right side of UI Window in Figure 4.2.

In select mode, user can select points/planes/meshes by making a rectangle by click and drag motion of mouse. For planes and points the color of selected entity changes to indicate a successful selection. This can be seen in Figure 4.2 where the selected points for adding plane are highlighted in red color.

In pan mode, users can press keys "O" or "P" to traverse along keyframes to view the scene in various positions. Also, adjustment keys for translation, rotation of point cloud are provided for minor calibration. These are "WASDZX" for translation, and mouse click-drag motion for rotation. Users can also change the precision of movement along translation using "Shift"(for fine translation movements), and "Alt"(for faster translation movements).

The typical actions user can take are select points to create a best fit plane(Add option) using RANSAC method, merge two planes (Adjust Plane option), remove planes(Remove option).

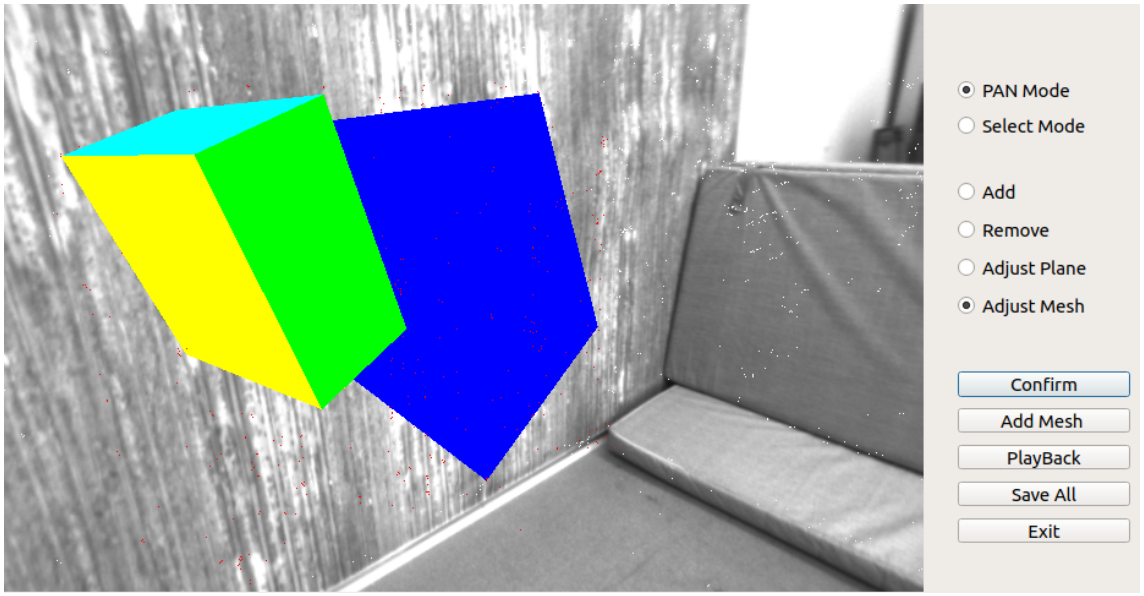


Figure 4.3: Adjusting mesh.

The pressing "Confirm" applies the transformation selected. (either add/remove/adjust)

Then users after being satisfied with the geometry can add a mesh clicking the "Add mesh" button. One can then adjust mesh and position it along one of created planes (using Adjust Mesh option). In this step, user selects a corner of plane, then selects and corner of mesh and clicks on confirm to join those two points. This join can be seen as in Figure 4.3

After placing the mesh in right pose, user can save the augmentations using "Save" button. This marks the end of interaction phase. Now user can click "Playback" button to see those augmentations in the input video.

Online mode

This is typically done after completing offline mode. In this mode, user provides saved point cloud and RGB data as inputs to the application. Then we run the slam and matching algorithm simultaneously, to detect whether the current image of the feed has been seen in the preprocessing phase in previous offline mode. If the application finds a match, it augments the previously saved meshes at appropriate locations in the current feed. Now the user can view that mesh in realtime from all possible camera poses. Figure 4.4 shows the result of online mode where the mesh is viewed from a different camera pose than shown in Figure 4.3.

4.2 Results

We have tested our application on the recent EUROC dataset by Burri *et al.*[1]. The video sequence used had 2912 images, capturing a well textured environment for easy tracking

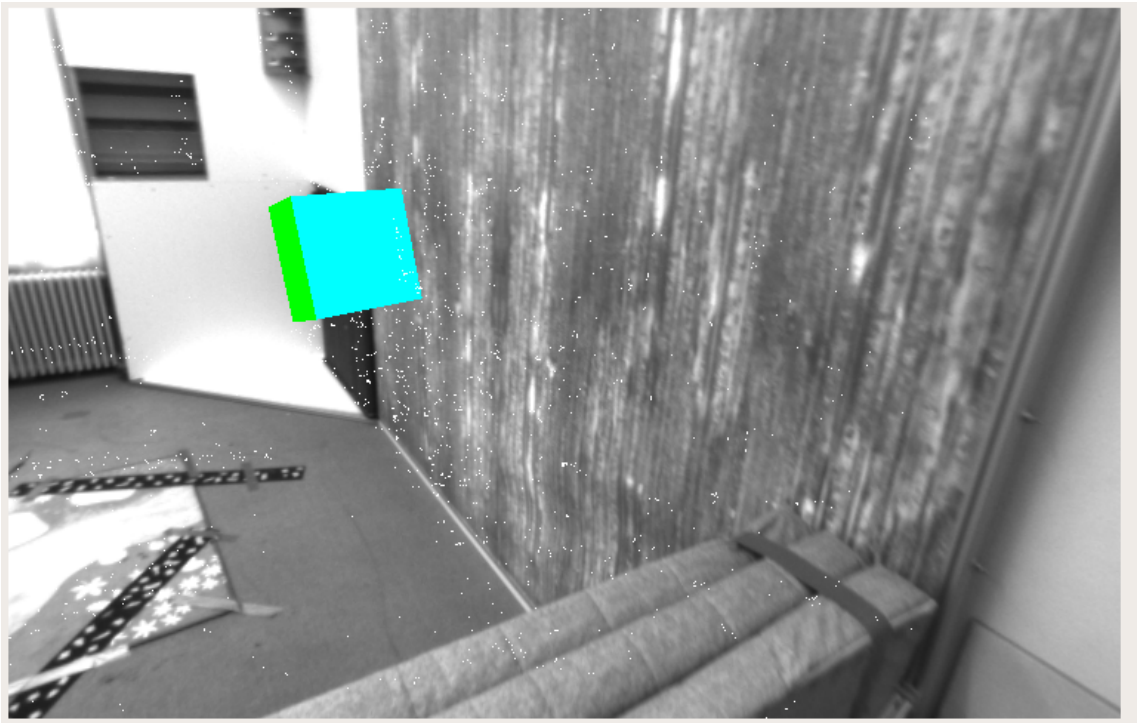


Figure 4.4: Online Mode After Successful matching.

and detection for SLAM. The augmentations in the scene were stable enough to blend in the environment. The initial calibration of position of augmentation was a bit off, which could be fixed by translation commands at time of playback mode.

For testing the webcam mode, we captured a few indoor environment videos of time interval 20-30 seconds. SLAM was not working well in this environment so it took many tries to register the scene. This was due to the poor texture and lighting in this environment. In the part that was detected well in the scene, augmentations were stable.

For online mode the matching algorithm was working efficiently to produce augmentations in the required place for the EUROC dataset.

4.3 Source Code and User Manual

For viewing the source code and detailed explanation of our user interface check out our Github page.

Chapter 5

Photorealistic Approach

While studying various methods to go about performing photorealistic rendering for AR, one common thing we found was that these methods used RGB-D sensors and not RGB sensors. Since our requirement demands the use of RGB data, we began to study what exactly in the depth data is required to tackle the issue of visual coherence.

From Mandl *et al.*[5]'s work, we learnt that we could have used multiple light probes as long as their combined distribution of surface normals covered most of the hemisphere (like a standard light probe). Studying the work of Gruber *et al.*[2] carefully revealed that it required the use of depth maps only so that better geomery and surface normals can be estimated from the reconstruction of scene. This led to the idea of using point cloud to obtain better surface normal estimates. If this would be possible in realtime, then one could actually use the surface normals obtained in Gruber *et al.*[2]'s method to get incident illumination. Here, they assume that the light source is white colored, albedo of all materials in the scene is constant and all surfaces are lambertian in nature. These assumptions are not very restrictive for our photorealistic rendering and thus we can work with this method.'

Following this idea we came across a method for converting point clouds to mesh by LadickÃ¡ *et al.*[4]. They designed a suitable feature vector and efficient oct-tree based GPU evaluation, capable of predicting surface of high resolution 3D models in milliseconds. Their method learns and predicts surfaces from an observed point cloud in a small-time. We could incorporate this in our preprocessing phase and use modified version of Gruber *et al.*[2]'s method to get radiance transfer estimates and solve inverse rendering equation to get spherical harmonics coefficients and thus render the scene using the obtained light model.

Chapter 6

Conclusion and Future Work

At present, the application can render high quality non-photorealistic AR. The UI could be improved upon to make the application more user-friendly. The clicks and drags for selecting points is not very easy and requires more work from users in a non-intuitive manner.

We plan to implement the approach presented in Chapter 5, to tackle the issues of visual coherence in realtime using only RGB data. Also, the results on a non-slam-friendly scene were not very good due to bounds on the slam API used. We can try to improve upon that aspect to get more robust AR application.

In conclusion, We provide an application for AR rendering that is both realtime and interactive. We also propose a novel approach to tackle some of the issues of photorealistic AR by solving inverse rendering equation smartly. Unlike earlier approaches proposed, our approach is guided by constraints of RGB input and does not make any assumptions about the environment.

References

[1] Burri, M., Nikolic, J., Gohl, P., Schneider, T., Rehder, J., Omari, S., Achtelik, M. W., and Siegwart, R., 2016, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research* doi:\bibinfo{doi}{10.1177/0278364915620033}

[2] Gruber, L., Richter-Trummer, T., and Schmalstieg, D., 2012 Nov, "Real-time photometric registration from arbitrary geometry," in *2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 119–128.

[3] Hold-Geoffroy, Y., Sunkavalli, K., Hadap, S., Gambaretto, E., and Lalonde, J.-F., 2017, "Deep outdoor illumination estimation," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2373–2382.

[4] LadickÃ¡, L., Saurer, O., Jeong, S., Maninchedda, F., and Pollefeys, M., 2017 Oct, "From point clouds to mesh using regression," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 3913–3922.

[5] Mandl, D., Yi, K. M., Mohr, P., Roth, P. M., Fua, P., Lepetit, V., Schmalstieg, D., and Kalkofen, D., 2017 Oct, "Learning lightprobes for mixed reality illumination," in *2017 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 82–89.

[6] Mur-Artal, R., Montiel, M., J. M., Tardós, and D., J., 2015, "ORB-SLAM: a versatile and accurate monocular SLAM system," *IEEE Transactions on Robotics* **31**, 1147–1163.

[7] Richter-Trummer, T., Kalkofen, D., Park, J., and Schmalstieg, D., 2016 Sept, "Instant mixed reality lighting from casual scanning," in *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 27–36.