END-TO-END REPRESENTATION LEARNING FOR 3D RECONSTRUCTION

Soroush Saryazdi

A thesis
in
The Department
of
Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Science (Computer Science) at
Concordia University
Montréal, Québec, Canada

April 2021

Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: Soroush Saryazdi

Entitled: **End-to-end Representation Learning for 3D Reconstruction**

and submitted in partial fulfillment of the requirements for the degree of

Master of Science (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Dr. Andrew Delong — Chair

Dr. Thomas Fevens — Examiner

Dr. Andrew Delong — Examiner

Dr. Kaustubha Mendhurwar — Co-supervisor

Dr. Sudhir P. Mudur — Supervisor

Approved by

Dr. Hovhannes Harutyunyan, Graduate Program Director
Department of Computer Science and Software Engineering

20

Dr. Mourad Debbabi, Interim Dean
Faculty of Engineering and Computer Science

Abstract

End-to-end Representation Learning for 3D Reconstruction

Soroush Saryazdi

Physically based rendering requires the digital representation of a scene to include both 3D geometry and material appearance properties of objects in the scene. Reconstructing such 3D representations from images of real-world environments has been a long-standing goal in the fields of computer vision, computer graphics, robotics, augmented and virtual reality, etc. Recently, representation learning based approaches have transformed the landscape of several domains such as image recognition and semantic segmentation. However, despite many encouraging advances in other domains, how these learning-based approaches can be leveraged in the realm of 3D reconstruction is still an open question. In this thesis, we propose approaches for using neural networks in conjunction with the 3D reconstruction pipeline such that they can be trained end-to-end based on a single end objective (e.g., to reconstruct an accurate 3D representation). Our main contributions include the following:

- A fully differentiable dense visual SLAM framework for reconstructing the 3D geometry of a scene from a sequence of RGB-D images, called `gradslam`. This work, carried out in collaboration with the Robotics and Embodied AI Lab (REAL) at MILA, resulted in the release of the first open-source library for differentiable SLAM.

- We propose the disentangled rendering loss for training neural networks to estimate material appearance parameters from image(s) of a near-flat surface. The disentangled rendering loss allows the network to weigh the importance of each material appearance parameter based on its effect on the final appearance of the material, while also having desirable mathematical properties for gradient-based training.

- We describe work towards an end-to-end trainable model that can simultaneously reconstruct the 3D geometry and predict the material appearance properties of a scene. A publicly available dataset for training such a model is

not currently available. Thus, we have created a dataset of material appearance properties for complex scenes which we intend to release publicly.

Our approach enjoys many of the benefits of classical 3D reconstruction approaches such as interpretability (due to the modular nature) and the ability to use well-understood components from the reconstruction pipeline. Further, this approach also enjoys representation learning benefits such as the capability of solving challenging tasks which have been difficult to solve by designing explicit algorithms (e.g., material appearance property estimation for complex scenes), and their strong performance on end-to-end training tasks.

Acknowledgments

My two wonderful years as a Master's student have been an amazing learning experience for me, and there are many amazing people that have contributed to this incredible journey.

First, I must thank my supervisor Sudhir Mudur who gave me the freedom to follow my research interests while guiding me to become a more competent researcher. Under his supervision, I was inspired to explore many exciting topics such as deep learning, computer vision, robotics and I was introduced to the world of computer graphics. He was always open for having thoughtful discussions about my work and going over my paper drafts. When I presented an idea, he would break it down to its core elements and uncover the fundamental essence of it. I am very grateful to him for showing me the guidelines of doing great research and more importantly for shaping my thoughts. I would also like to thank my co-supervisor Kaustubha Mendhurwar for all his support during my studies and for proof-reading this thesis.

I am very thankful to Liam Paull for giving me the opportunity to partake one of the limited seats in one of the most enjoyable courses ever, autonomous vehicles (Duckietown), but also continuing to open doors for me beyond this course and welcoming me into collaborating with his group, Robotics and Embodied AI Lab (REAL) at Mila, which I have learned so much from.

I must thank my close collaborators who I have had the pleasure of working with and learning from. I would like to especially thank Krishna Murthy Jatavallabhula who has inspired me with his unique perspectives and his unrivaled ability in communicating technical ideas, Ganesh Iyer who is a seemingly infinite resource of unique and interesting ideas, and Christian Murphy who has an uncanny ability in recognizing the missing piece in a research area for it to become a valuable practical tool.

I am deeply thankful to Derek Nowrouzezahrai and Mike Roberts for being extremely generous with their support and insightful comments. I would like to thank Nader Asadi, Farzad Salajegheh, Iman Haji Abolhassani, and Amir Sarfi for

the thoughtful interactions that contributed to my thinking and research philosophy. I want to extend thanks to Hossein Nezamabadi-pour, Kambiz Afrouz, Saeid Seydnejad, Ali Mahani, Mahdi Eftekhari and Devansh Arpit for putting their trust in me and taking the time to submit recommendation letters on my behalf.

Lastly, I would like to thank all my close friends and family who have believed in me for a very long time now. I want to thank my aunts and their families for always supporting me during my stay in Canada. A big thank you to my brother who always believed in me way more than I did. And lastly, a big thank you to my parents for supporting and educating me all throughout my life, including their amazing support of me studying abroad. Wanting to make you proud has perhaps been the strongest motivation for me during my studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

> *"Some facts might be made available if the programs could be taught to read and understand books, but comprehending even simple words would require detailed knowledge of the physical world. Such knowledge is assumed to preexist in the minds of book readers—no book attempts a comprehensive definition of a rock, a tree, the sky, or a human being. Possibly some of this world knowledge, as it has come to be called, could be obtained by the machine itself if it could directly observe its surroundings through camera eyes, microphone ears, and feeling robot hands."*
>
> —Hans Moravec, *Mind Children (1988)*

Understanding the 3D appearance of a scene based on one or more captured images is something humans are inherently capable of. As an example, in Fig. 1, we can look at the image on the left and immediately tell that the table surface is very reflective, we can point out the location of light sources, we can imagine the shape of the chairs, and we can even get an approximate estimate of the room dimensions. Moreover, we can easily connect the contents of the image on the left with the 3D scene on the right. For instance, we can tell how the 3D reconstruction on the right would differ if there was a vase on the table in the left image. We can also estimate the camera location with respect to the 3D scene when taking that image. While this ability feels very natural to a human, recovering this 3D representation from images is a very difficult task for a computer.

Challenges. Many of the physical laws that dictate how the 3D world appears from a viewpoint have been thoroughly investigated, well-understood and are commonly used in graphics (projective geometry, light flow computations, etc.).

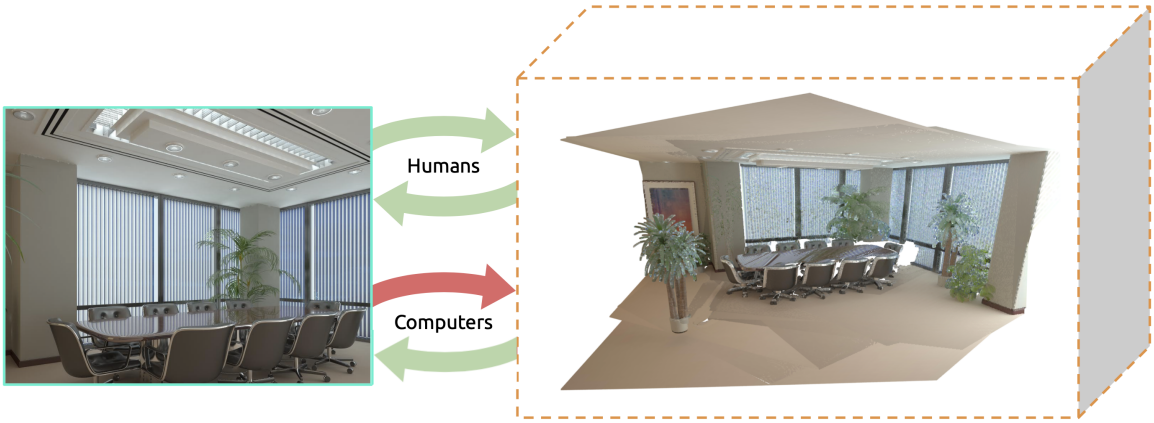

Figure 1: Comparing human and computer perception. **Left:** image of a synthetic indoor scene from the Hypersim dataset [1]. **Right:** partial 3D reconstruction of the scene.

However, going back from the captured image(s) to the 3D representation continues to be a very challenging task. To begin with, there has been a lack of consensus on the 3D representation (pointclouds, meshes, surfels, etc.) for recovering the 3D geometry (shape). This fundamental choice of the 3D representation dramatically impacts the processing blocks and the types of challenges that we face. Moreover, recovering the physical material appearance properties from images of a single object is an inherently ill-posed problem, as a material can appear quite different under various lighting and viewing conditions. Additionally, recovering the material appearance properties *of an entire scene* has extra challenges caused by inter-reflections of the different objects in the scene, shadows, etc.

Current state. Reconstructing the 3D geometry of a scene based on a captured sequence of RGB or RGB-D images has been an exciting and active area of research [2, 13–18]. Dense visual Simultaneous Localization and Mapping (SLAM) approaches focus extensively on building and updating the 3D map of an environment while simultaneously recovering the camera location in that map [2, 16–18]. Recently, gradient-based learning approaches have transformed the outlook of several domains (e.g., image recognition [19], language modeling [20], speech recognition [21]). These learning-based approaches have also shown exciting results in both realms of recovering the material appearance parameters [22–26] and improving SLAM pipelines [13–15, 27–29].

Remaining challenges. However, how to blend representation learning approaches with SLAM systems is still an open question. This is primarily because modern *dense* SLAM systems are quite sophisticated, with several non-differentiable

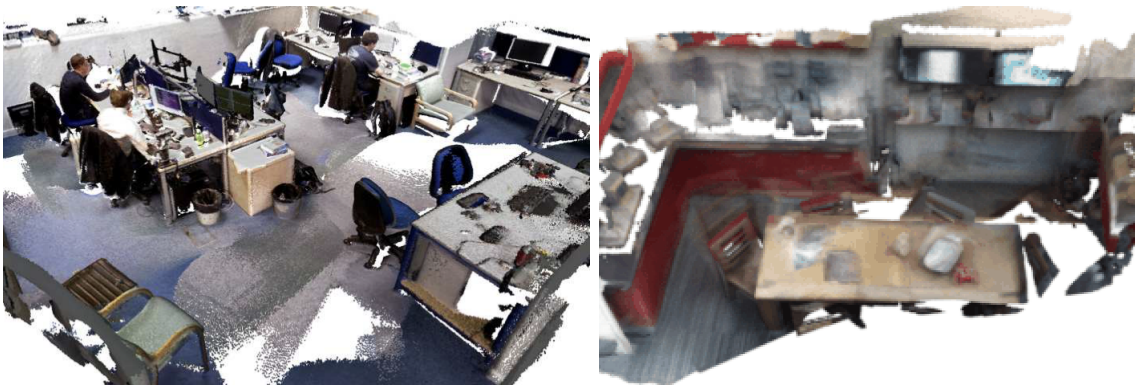

Figure 2: Dense visual SLAM examples. **Left:** ElasticFusion results [2]. Figure courtesy of [2]. **Right:** Volumetric TSDF fusion results. Figure courtesy of [3].

subsystems (optimizers, raycasting, surface mapping), that make such a construct challenging. This has limited learning-based works in this domain to adopt the following approach - first train the model for a specific task, and then plug the trained model into the SLAM pipeline [14, 30]. Additionally, SLAM approaches very commonly make an implicit or explicit Lambertian assumption on the materials captured in the scene, and there has been very little attempt on recovering finer material appearance properties such as shininess (specular reflections) or refractive behavior. To the best of our knowledge, a model which simultaneously recovers and updates a 3D map along with the complete material appearance properties has yet to be developed. In the context of learning-based material appearance recovery, several approaches leverage physical reflection models in the objective function for optimizing their data-driven models. However, the computational impacts of using reflection models in the optimization objectives has yet to be investigated.

Outline of Contributions. In this thesis, we propose end-to-end trainable 3D reconstruction models to enable simultaneous recovery of geometry and materials, as opposed to training separate components individually and piecing them together. This allows all components of our model to be trained simultaneously for a single end objective. The key insight in developing these models is to have the learning-based models leverage well-observed knowledge from classical computer vision, graphics, and robotics approaches in their training phase (e.g., via a loss function or a differentiable pipeline). Having this combination of learning-based and classical approaches can make our models easier to interpret (due to the modularity) as opposed to replacing the entire classical pipeline with a learned model. In summary, our goal is to blend end-to-end representation learning with classical 3D reconstruction knowledge.

Long-term motivations. The long-term goal of these techniques is to build

towards models which can accurately reconstruct a detailed 3D map from casually captured input images. Imagine turning on your camera, casually walking around with it, and getting an accurate 3D representation of the environment appearance, such as the shape of objects, the appearance of the materials, and the lighting conditions. This will open a huge realm of possibilities. First, these detailed 3D maps can directly be used in many tasks such as robotic navigation, augmented reality, virtual reality, games and many other visualization and graphics design tasks. As a concrete example, this can enable a robot to robustly navigate and operate in an environment containing non-Lambertian materials such as metals, etc. Second, this would enable massive amounts of detailed real-world 3D datasets to be created. The value of these datasets is not only that they accurately capture the appearance of the real world, but also that they contain valuable content from the real world long tail cases which could not have been created in a synthetic dataset.

Short-term motivations. These long-term aspirations can be motivated by shorter term valuable practical applications. First, a fully differentiable 3D reconstruction system would enable task-driven representation learning since the error signals indicating task performance could be back-propagated all the way through the reconstruction system, to the raw sensor observations. Second, accurate material appearance parameters can be used in several down-stream tasks such as scene re-lighting, material type classification [31], and virtual object insertion in augmented reality environments. Third, simultaneous recovery of the 3D geometry and material appearance properties would enable estimating the material appearance parameters of a complex environment without needing to capture each material in isolation. Moreover, this could open new realms of possibilities for self-supervised learning on images once combined with differentiable renderers.

Challenges of this approach. However, this approach also comes with some challenges. One common challenge is the choice of the 3D representation for the 3D geometry of the scene (pointcloud, voxel grid, mesh, surfel, etc.). This fundamental choice dramatically impacts the choice of processing blocks in the 3D reconstruction pipeline, as well as all other downstream tasks that depend on the outputs of the 3D reconstruction system. We started by considering the three most common types of representation for dense SLAM systems: voxel grids, pointclouds, and surfels. However, in our more recent work we mostly focus on the pointcloud representation due to the lower memory cost (memory can easily become a bottleneck on current GPUs). Another challenge that arises is in gathering a large dataset of labeled data for recovering material appearance properties. Acquiring large scale and accurate ground

truth material appearance parameters for real world data is close to impossible (either annotators would have to roughly estimate the parameters, or one would need to use expensive dedicated hardware for capturing one object at a time). For example, the OpenSurfaces dataset [32] crowdsources internet images for material, texture and semantic annotations. However, their material annotations are approximate estimates and assume uniform appearance across a surface, and their texture annotations are combined with lighting effects. Thus, a much more feasible alternative is to generate synthetic data. However, the distribution of the generated synthetic data should be as close to the real world data as possible to get better generalization (to real world data) out of our models. To lessen this gap between synthetic data and the real world, we can use accurate physically based rendering engines with artist designed 3D models. However, acquiring this data can be financially expensive and rendering it can be very computationally expensive. Lastly, while there is considerable work in recovering exact lighting conditions in a scene, it should be mentioned that in our work we have not addressed this explicitly. However, our network needs to implicitly reason about lighting conditions to predict material and shape accurately.

1.2 Related Work

Below, we will briefly review related work primarily to set the stage for the main contributions of this work as presented in the next section. More comprehensive reviews of related work for each of the problems are separately provided in the individual chapters, which discuss the corresponding problems and solutions.

Geometry recovery. As previously mentioned, dense visual SLAM approaches focus on building and updating the 3D map of an environment from a sequence of camera captures while simultaneously recovering the camera's location in that map [2, 16–18]. There is a large body of work in deep learning-based SLAM systems. For example, systems such as CodeSLAM [33], SceneCode [34], and DeepFactors [35] represent scenes using compact *codes* that can be decoded into 2.5D depth maps. DeepTAM [13] trains a tracking network and a mapping network, which learns to reconstruct a voxel representation from a pair of images. CNN-SLAM [14] an extension of LSD-SLAM [15], a popular monocular SLAM system, to use single-image depth predictions from a convolutional neural network. Another recent trend has been to formulate the SLAM problem over higher level features such as objects, which may be detected with learned detectors [27–29]. However, a large fraction of the aforementioned approaches *replace* SLAM subsystems with learning based models.

In contrast, there is another significant line of work that leverages differentiability to *complement* and accelerate learning mechanisms.

The Lucas-Kanade iterative matching algorithm [36] is one example of a strong demonstration of the benefits of differentiable SLAM subsystems. Kerl *et al.* [37] applied this technique to real-time dense visual odometry. Their system is differentiable, and has been extensively used for self-supervised depth and motion estimation [38–40]. Coupled with the success of Spatial Transformer Netowrks (STNs) [41], several libraries such as gvnn [42] and kornia [43] have implemented these techniques as differentiable *layers*, for use in neural networks.

However, extending differentiability beyond the two-view case (*frame-frame alignment*) is not straightforward. Global consistency necessitates fusing measurements from live frames into a global model (*model-frame alignment*), an operation which is not trivially differentiable. In summary, while all these approaches try to leverage differentiability in submodules of SLAM systems (eg. odometry, optimization, etc.), there is no single framework that models an entire SLAM pipeline as a differentiable graph.

Material appearance parameters recovery. The problem of recovering the material appearance properties can be simplified into recovering the BRDF (bi-directional reflectance distribution function) or SVBRDF (spatially-varying BRDF) parameters of every material present in the scene. Classical BRDF measurement approaches rely on capturing a large number of images under different calibrated viewing and lighting conditions using dedicated acquisition setups [44]. For this thesis, we focus on methods of SVBRDF recovery which use casual image captures of materials in the wild.

Recently, deep learning models have shown a lot of promise in reflectance modeling from images in the wild [22–25]. Li *et al.* [25] used a convolutional neural network which takes an image of a near-planar surface and estimates its per-pixel BRDF parameter values. They train this network by using the traditional L2 loss over the predicted BRDF parameters. Deschaintre *et al.* [22] showed that the predictions of a network trained with this loss function do not yield accurate re-renderings compared to the ground truth images. Instead, in their work on recovering SVBRDF from a single flash-lit image of a near-planar surface, both Deschaintre *et al.* [22] and Li *et al.* [24] found the *rendering loss* to be a better alternative for training their networks. The rendering loss is computed by using the L1 loss between the renderings of the predicted and the ground truth BRDF maps under the same lighting and viewing conditions. Intuitively, this will incorporate some of the information about

how much each BRDF parameter is useful in the final rendering output and how the different estimated BRDF parameters interact with another through the differentiable rendering process into the neural network. However, in their approach, the recovered specular and diffuse albedo maps have errors when compared to ground truth maps, despite the fact that the final rendered image(s) looks similar to the input image(s).

Currently, the best results are obtained using multi-image deep networks [23, 45]. These are networks that use multiple images of the same material under different light and view conditions as their input. In the multiple image setup, the different views can now provide the network with more cues on what the BRDF should be, and ideally, we would like the network to rely less on the learned priors about the material properties and more on the visual cues in the different images as the number of views increases. The more recent work by Deschaintre *et al.* [23] can handle an arbitrary number of input images. Similar to previous work by Gao *et al.* [45], Deschaintre *et al.* [23] found that using a combination of L1 loss on the predicted maps and rendering loss during training helped stabilize the training procedure. However, the individual recovered SVBRDF maps still have inaccuracies, and there are often instances where the network predicts incorrect maps that render to a a very similar looking image, for example, by incorrectly assigning the color from the diffuse albedo map to the specular albedo map.

Simultaneous geometry and BRDF recovery. Recent works have been looking into using RGB-D sensors to simultaneously recover geometry and SVBRDF [46–50]. These approaches commonly either require extra equipment at capture time [46, 48] or simplify the domain of recovered SVBRDFs by clustering materials and associating a per-segment BRDF [47, 50]. However, doing per-segment BRDF associations reduces result quality when the scene contains a large number of different BRDFs. Moreover, with the exception of the work of Zhan *et al.* [50], all above approaches focus on recovering SVBRDF for a single object, and not an entire scene. However, Zhan *et al.* [50] rely on manual adjustments and restrict BRDF parameter recovery to only the floor, ceiling, and walls. Another limitation of the aforementioned approaches is that they require an iterative refinement optimization process, which can be time consuming to apply for each RGB-D sequence.

An interesting recently emerging line of work is approaches that take inspiration from Neural Radiance Fields (NeRF) [51–54]. NeRF based approaches for recovering the SVBRDF rely on fitting a separate neural network for every captured scene to recover the per-pixel BRDF parameters [52–54]. However, this optimization process can be long and needs to be done separately for every new captured scene. This has

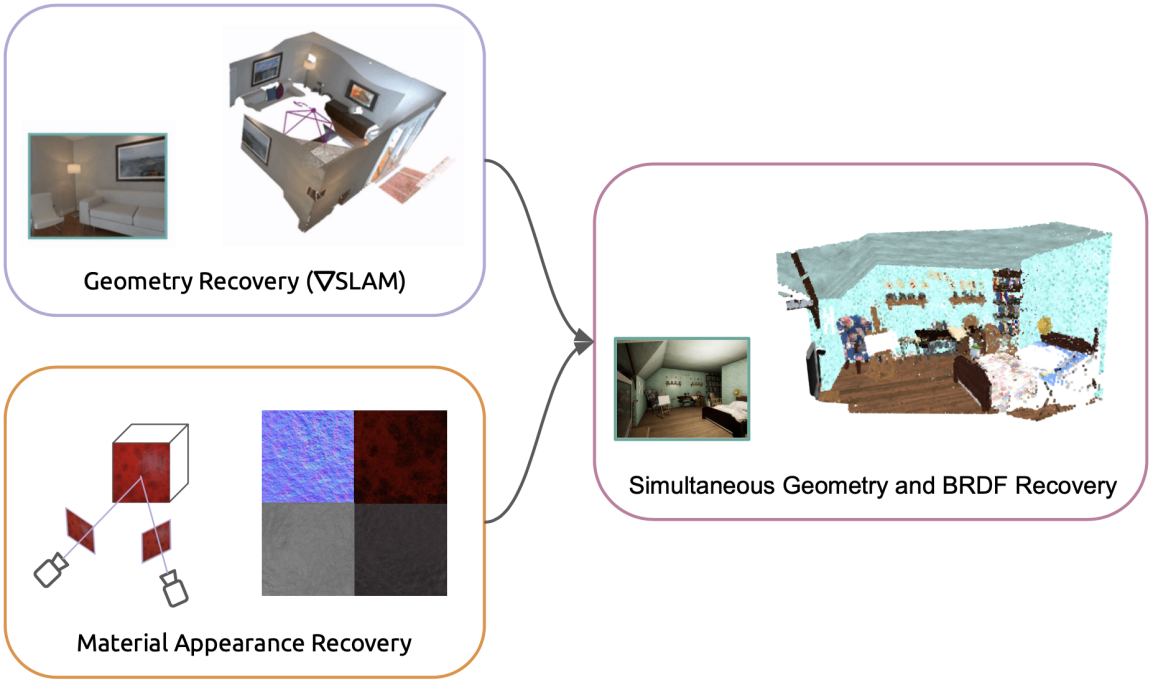

Figure 3: Overview of our contributions. **Top left:** differentiable geometry recovery using `gradslam` library. **Bottom left:** material appearance parameters recovery. **Right:** end-to-end learning for simultaneous material appearance parameters and geometry recovery.

led us to believe that there are currently no generalized approaches for simultaneously recovering the geometry and SVBRDF of a scene based on captured RGB or RGB-D images.

1.3 Contributions

In this work, we propose models for blending representation learning approaches with classical 3D reconstruction knowledge. We begin by developing an open-source library containing fully differentiable methods of reconstructing the 3D geometry. Then, we look at how we can better leverage our knowledge of appearance modeling in training neural networks for material appearance parameter estimation. Once we have a solution to both aforementioned problems, we propose a model for simultaneously recovering the 3D geometry and the material appearance properties. The overview of our contributions is depicted in Fig. 3. Our contribution in each of these sections can be broken up as follows:

1. **Differentiable dense SLAM library:** We develop and release `gradslam`,

the first open-source differentiable SLAM library that supports multiple differentiable dense SLAM systems out-of-the-box. The work on ∇SLAM was conducted in collaboration with the Robotics and Embodied AI Lab (REAL) at Mila led by Dr. Liam Paull. Our work on ∇SLAM has led to papers at *CVPR 2020 Deep Declarative Networks* workshop and *RSS 2020 SARL* workshop [9].

2. **Disentangled rendering loss:** Earlier works in recovering material appearance parameters from one or more images of a near-flat surface have found the *r*endering loss to work well if one's goal is to recover accurate re-renderings from the estimated parameter maps [22–24]. We identify that neural networks trained with the rendering loss have trouble with accurate recovery of individual parameter maps. We begin by defining the problem and analyzing why these inaccuracies arise. We then propose a new loss function which addresses this problem, named the *disentangled rendering loss*. We show that using our disentangled rendering loss to train the current state of the art network leads to a noticeable increase in the accuracy of recovered material appearance property maps. The analysis of the rendering loss led to a paper at *Eurographics MAM 2020* workshop [55], and proposing the disentangled rendering loss led to a paper at *GRAPP 2021* conference [56], where it won the *best student paper award*.

3. **Simultaneous geometry and BRDF recovery:** We describe a model for simultaneously recovering the 3D geometry and the material appearance parameters from casually captured RGB-D images of a scene, yielding a generalized approach for simultaneous BRDF and geometry recovery of complex casually captured scenes. Moreover, we propose a novel approach for leveraging 2D multiview information when making predictions for each point's BRDF by leveraging ∇SLAM, and we extend our idea of the disentangled rendering loss from near-flat surfaces into being used for pointclouds. Finally, we generate a new large-scale dataset based on the Hypersim [1] rendered images which contains appearance parameters. We intend to make this dataset public in the near future. While the experiments in this section are still work in progress, this section ties the whole thesis together into our broader vision for 3D reconstruction with accurate material appearance capture. The works in this section are in collaboration with Farzad Salajegheh, Krishna Murthy Jatavallabhula, Mike Roberts, Derek Nowrouzezahrai and Liam Paull.

1.4 Thesis Outline

The rest of this thesis is as follows:

In **Chapter 2** we provide the relevant background in deep learning, physically based rendering, material appearance modeling, and dense visual SLAM.

In **Chapter 3** we describe the components of a fully differentiable dense SLAM pipeline, and describe our differentiable dense SLAM library called `gradslam`. The content of this chapter is primarily based on the work of Jatavallabhula, Saryazdi, Iyer and Paull [9] where Jatavallabhula and Saryazdi are equal co-authors.

In **Chapter 4** we analyze the rendering loss used in material appearance parameter estimation research. We then propose the disentangled rendering loss to address some of the rendering loss limitations. The content of this chapter is primarily based on the work of Saryazdi, Murphy and Mudur [55, 56].

In **Chapter 5** we describe a method for simultaneously recovering geometry and material appearance parameters of complex scenes. We also introduce our dataset which is an extension of the Hypersim [1] dataset with additional per-pixel material appearance parameters.

In **Chapter 6**, we conclude this thesis by summarizing our findings and discussing the path forward.

Chapter 2

Background

2.1 Deep Learning

We assume an understanding of deep learning fundamentals from the reader. We refer the reader to the excellent Stanford CS231n online course material[1] and the Deep Learning book [57] for a comprehensive background in deep learning. In this chapter, we just briefly touch upon the background materials which are specifically relevant to this thesis.

2.1.1 Architectures

U-Net. The U-Net architecture has become one of the most common architectures for image-to-image modeling. U-Net is a fully convolutional encoder-decoder type architecture. The architecture of U-Net is shown in Fig. 4 for an input image of size 572×572.

In particular, the first half of U-Net is the encoder (*contracting path*) which encodes the context of the image by reducing the spatial dimensions. This is done by repeatedly applying a downsampling step, which uses two 3×3 (unpadded) convolutions with ReLU non-linearity followed by a 2×2 max pooling layer with a stride of 2. The number of feature channels is doubled at each downsampling step. The second half of the network is a decoder (*expansive path*) which consists of repeated upsampling steps to increase the spatial dimensions. Each upsampling step consists of a 2×2 transposed convolutional layer (*deconvolution*) which halves the number of channels, followed by cropping and concatenating the corresponding feature map from the encoding step, and finally two 3×3 convolutional layers with

[1] http://cs231n.stanford.edu

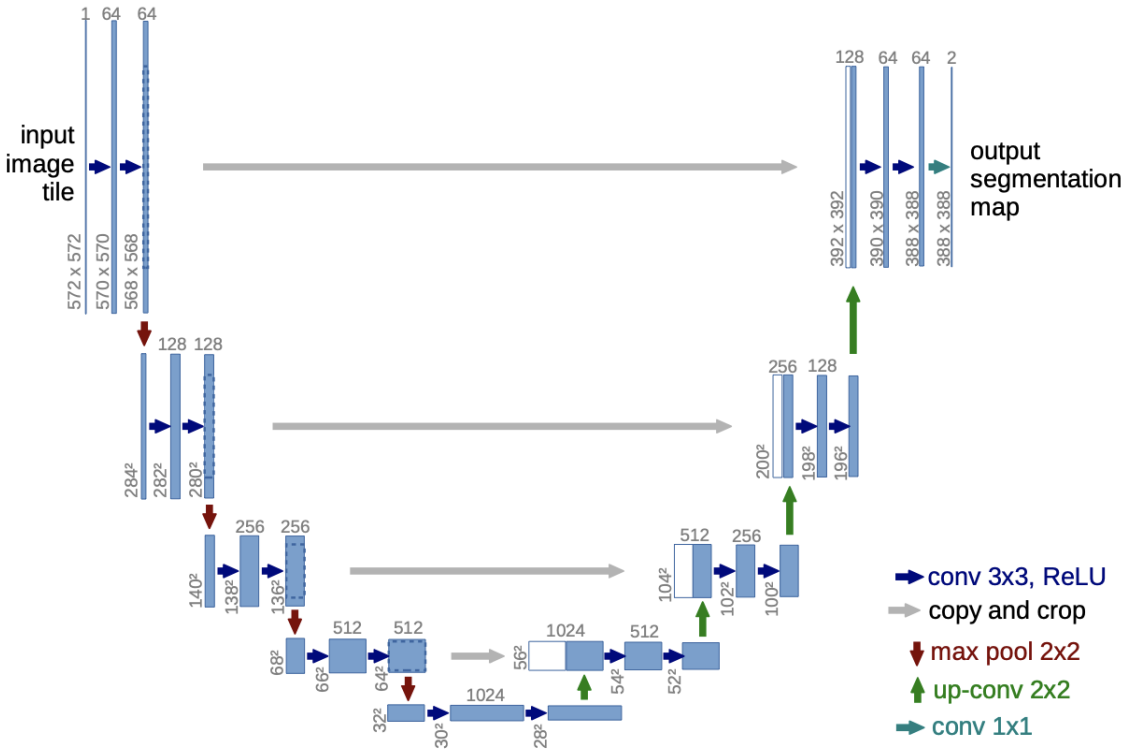

Figure 4: U-Net architecture. Blue boxes correspond to feature maps, and white boxes correspond to copied feature maps. The number at the top of each box denotes the number of channels. The spatial dimensions are denoted at the bottom left of each box. Figure courtesy of [4].

ReLU activation functions. At the final layer, 1×1 kernels are used to convert the 64 feature channels into the desired number of output classes. In total, this network has 23 convolution layers.

Point-Voxel CNN. Point-Voxel CNN [5] is a fast and efficient point-based neural network architecture. Due to the irregular format of a pointcloud, conventionally researchers transformed the pointcloud into a voxel grid and used 3D convolutional operations for processing this data. However, the rasterization of points into voxel grids causes information to be lost (i.e., when multiple points fall in the same voxel). The solution to this is to increase the voxel grid resolution. However, both computation and memory cost increase cubically as the resolution increases. Point-Voxel CNN [5] addresses this issue by keeping the 3D representation of data as a pointcloud to keep a small memory footprint, while performing the convolution operation over low-resolution voxels to leverage locality.

The main building block for the Point-Voxel CNN [5] is the Point-Voxel

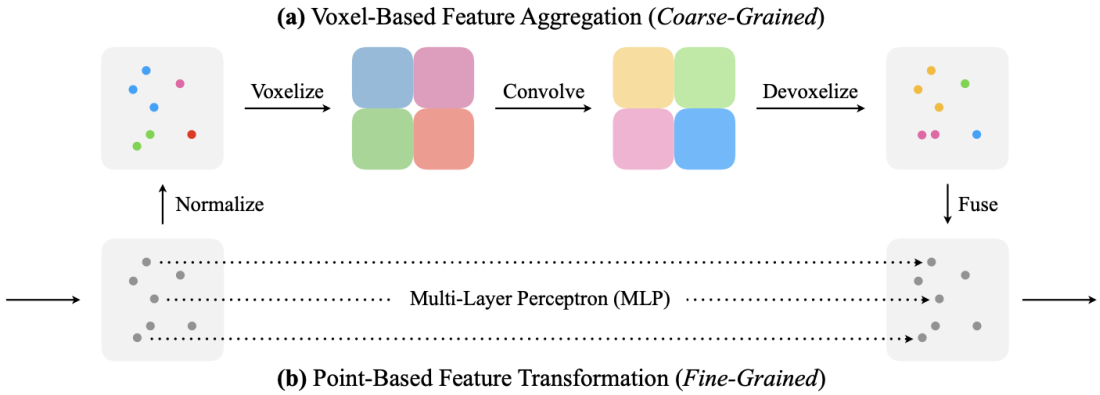

Figure 5: Point-Voxel convolution. Figure courtesy of [5].

convolution (PVConv). In particular, Point-Voxel convolution has a voxel-based branch and a point-based branch, as shown in Fig. 5. Given a pointcloud of $\{(\mathbf{v}_k, \mathbf{f}_k)\}$ where $\mathbf{v}_k$ is the vertex position of the k^{th} input point and $\mathbf{f}_k$ is the associated features with v_k, the voxel-based branch starts by normalizing the point vertices v_k into a sphere of unit diameter and centered at $(0.5, 0.5, 0.5)$. Then, the normalized pointcloud is transformed into a low-resolution voxel-grid where the associated features of all points that fall in the same voxel are averaged. After voxelizing the pointcloud, a stack of 3D convolutional layers is applied to the pointcloud. Each convolution layer also uses batch normalization [58] and a non-linear activation function [59]. The final stage of the voxel-based branch is to devoxelize the voxel-grid using trilinear interpolation back into the pointcloud domain. Trilinear interpolation ensures that points which were inside the same voxel can have distinct features.

The point-based branch directly applies a Multi-Layer Perceptron (MLP) on top of each point feature $\mathbf{f}_k$. This simple mechanism allows for high-resolution discriminative features for each point. Finally, the features from the voxel-based and point-based branch are aggregated together in the pointcloud domain by using addition.

2.1.2 Loss Functions

Chamfer Distance. One of the important aspects when working with 3D deep learning is the choice of the loss function. The Chamfer Distance (CD) is an error metric between two sets of points (pointclouds) S_1 and S_2. For every point in either sets, CD finds the nearest point from the other set, computes the euclidean distance between the two, and sums up all these distances for all points (see Fig. 6):

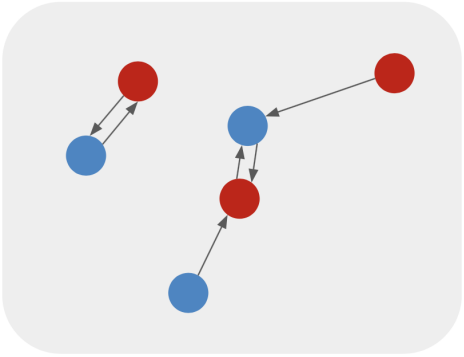

Figure 6: Chamfer distance illustration. Blue points belong to the set S_1, and red points belong to the set S_2. Black vectors start at each point from either set and end at its nearest neighbor from the other set.

$$\text{CD}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} ||x - y||_2^2 + \sum_{y \in S_2} \min_{x \in S_1} ||x - y||_2^2.$$

Formally, CD is not a distance function as it does not satisfy the triangle inequality axiom. However, here the term "distance" is used to imply a non-negative metric function. CD has several nice properties. First, it is differentiable with respect to the point positions. Second, CD is efficient to compute and simple to parallelize as the nearest neighbor for each point can be found independently. Moreover, the nearest neighbor search can be further accelerated by using efficient spatial data structures (e.g., KD tree). Third, CD is robust to having a small number of outlier points in the pointcloud.

2.2 Physically Based Rendering

Physically based rendering (PBR) is the process of generating a 2D image from a 3D representation, aiming to create images that look like real world photos (photorealism) by modeling light behavior (see Fig. 7). As a simplistic high level overview, light gets emitted by light sources, bounces around the environment, and whatever light rays that get reflected into the camera determine the pixel values of the generated image. Trying to recreate the appearance of the real world is difficult, and light computations can take a very long time. In this section, we discuss a high-level overview to some of the components of the PBR pipeline which are most relevant to this thesis.

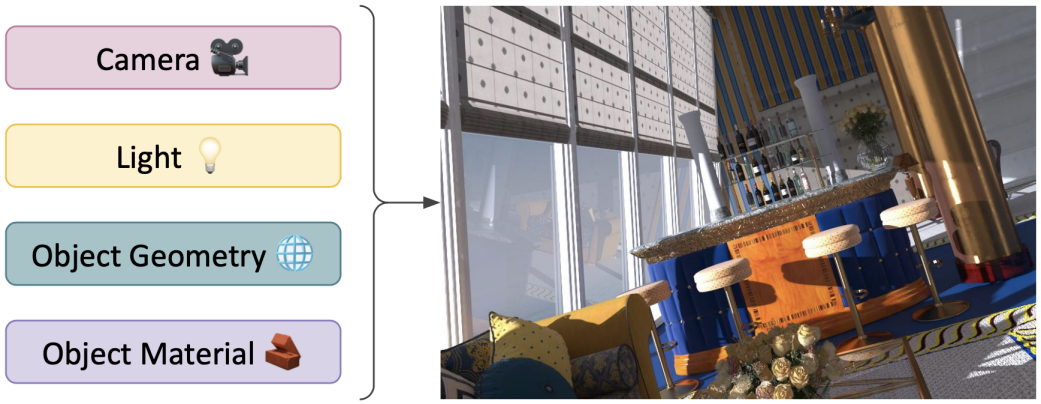

Figure 7: Camera, light conditions, object geometries and object materials are used by the PBR to generate a photorealistic image. Rendered image courtesy of Hypersim dataset [1].

2.2.1 Lights

The only reason objects are visible is due to light. In the real world, light is emitted by light sources, and all light sources have a physical body. In graphics, however, light sources can either have a physical body (such as *area lights*) or not (such as *point lights* and *directional lights*). Various light source types can illuminate the graphics scene in different ways (shown in Fig. 8):

- **Point:** Point lights emit light from a single point in all directions. The intensity of the light decreases with distance. The point light can be thought of as a very rough approximate for a bare light bulb hanging from a cord in the real world.
- **Directional:** Directional lights are produced by light sources which are at an infinite distance from the scene, thus all their light rays reaching the scene will be parallel to one another. Unlike point lights, the intensity of directional lights does not change with distance. An example directional light source in the real world is sunlight.
- **Area:** Area lights are more realistic and are important in physically based rendering as real world light sources have physical bodies. Area lights allow for creation of soft shadows, and can have different shapes such as rectangle, sphere, disk, or tube. However, using area lights has a higher computation cost.

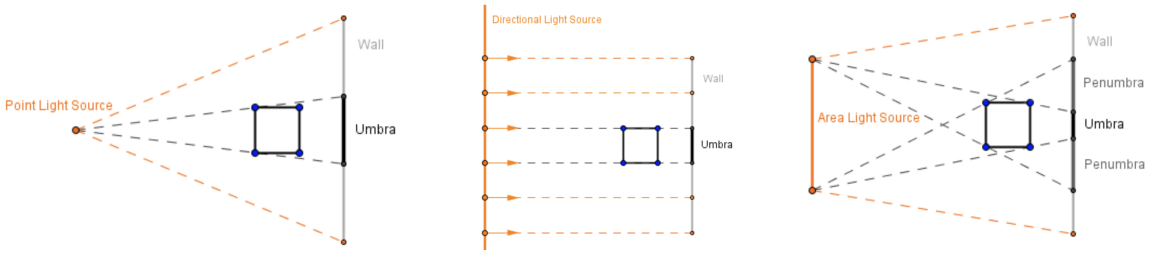

Figure 8: Light source types. **Left:** Point light. **Middle:** Directional light. **Right:** Area light. All 3 images courtesy of [6].

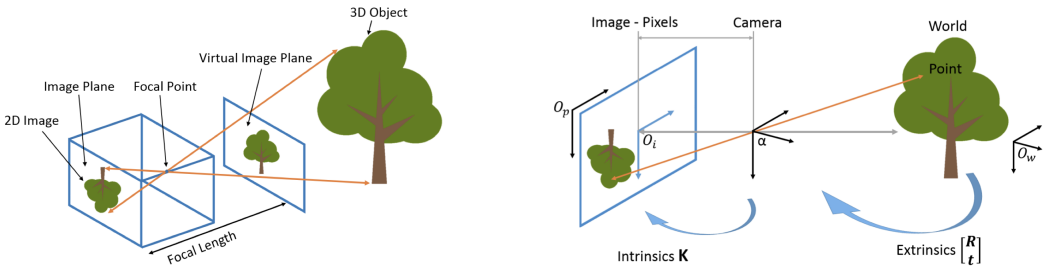

Figure 9: Pinhole camera model. Both images courtesy of [7]. **Left:** Image formation process. **Right:** Using extrinsics and intrinsics for transforming coordinate systems.

2.2.2 Camera

For the purpose of this thesis, we discuss one of the simplest and most commonly used camera models in computer graphics and computer vision: the pinhole camera. A pinhole camera has a tiny aperture and no lens. It is essentially an enclosed box with a tiny hole on one side (see Fig. 9). A pinhole camera can be modeled by its intrinsics matrix and its extrinsics matrix:

Camera intrinsics. This intrinsics matrix denotes *the coordinate system transformation from the 3D camera coordinates into 2D image coordinates.* For a pinhole camera, the intrinsics matrix **K** can be formulated as:

$$\mathbf{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

where f_x and f_y are the focal length, c_x and c_y are the principal point offset, and s is the axis skew. The camera intrinsics are independent of the camera position and orientation in space.

Camera extrinsics. The extrinsics matrix describes how *the world is transformed relative to the camera.* The extrinsics matrix is made up of a 3×1

Figure 10: Artist designed scene. **(a)** Final rendered image from the Hypersim [1] dataset. **(b)** Diffuse albedo map. **(c)** Specular albedo map. **(d)** Specular roughness map.

translation vector $\mathbf{t}$ and a 3×3 rotation matrix $\mathbf{R}$. We can concatenate these two into a single 4×4 homogeneous matrix:

$$\begin{bmatrix} R & \mathbf{t} \\ \vec{0}_{1\times 3} & 1 \end{bmatrix}$$

The inverse of the extrinsics matrix, called the *camera pose*, gives us *the position and the orientation of the camera in world coordinates.*

Lastly, the camera projection matrix $\mathbf{P}$ for an ideal pinhole camera is composed of the camera intrinsics matrix and the camera extrinsics matrix:

$$\mathbf{P} = \mathbf{K} \times [\mathbf{R}|\mathbf{t}]$$

2.2.3 Materials and BRDFs

Assuming we have the light sources, the camera model, and the shapes of the object in the scene, we now need to determine how light will interact with the surface of the objects. In a simple case, if we have a *Lambertian* (ideal matte) material, the reflected light will be the same regardless of the viewing angle. However, in the more general case, the BRDF (Bi-directional Reflectance Distribution Function) can model

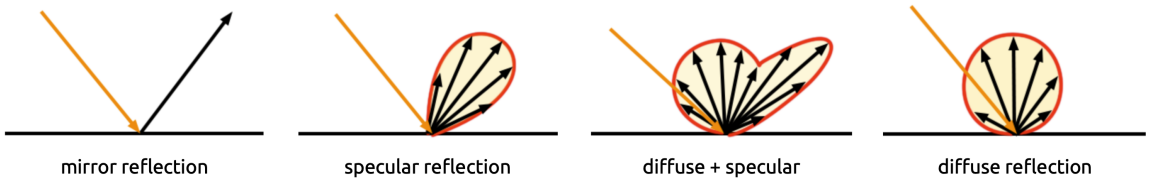

Figure 11: Illustration of different reflection types. All 4 images courtesy of [8].

how light is reflected (*radiance*) in any direction when incident light (*irradiance*) hits the surface of an object:

$$f_r(\omega_\mathrm{i}, \omega_\mathrm{o}, \lambda)$$

where f_r is the BRDF, ω_i and ω_o are the incoming and outgoing light directions respectively in a coordinate system where their z-axis is along the object's surface normal direction $\mathbf{n}$, and λ is a particular wavelength of light. The Spatially Varying BRDF (SVBRDF) $f_r(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o}, \lambda)$ takes an additional 2D variable $\mathbf{x}$ as input which denotes the spatial position over an object's surface and allows for spatial variability of the BRDF.

Over the years, a lot of works have focused on modeling the BRDF f_r as a generalized mathematical formula with a set of physically meaningful parameters that depend purely on an object's material [60–62]. In this thesis, we use the Cook-Torrance model [62] which is a *microfacets surface model*, where each surface is assumed to be composed of many tiny facets with different normal directions about the general surface normal. We use the GGX microfacet distribution [63] for the normal direction of these microfacets.

The GGX model has the following parameters: *diffuse albedo* which controls the intensity and the color of the light that gets scattered in many directions, *specular albedo* which controls the intensity and the color of the mirror-like reflection of the light, and *specular roughness* which controls how smooth the surface appears (see Fig. 11). If we assume the RGB color model, we would need 3 parameters for each of the diffuse and specular albedos, and 1 parameter for the specular roughness. These SVBRDF parameter maps are typically stored as images, and a *UV map* is used to project these 2D images onto the 3D model's surface. These SVBRDF maps are typically either artist designed or captured with expensive and dedicated hardware. An example of the rendering of some of these artist designed BRDFs is shown in Fig. 10. For a more comprehensive discussion on different material appearance representations we refer the reader to the excellent survey by Guarnera *et al.* [64].

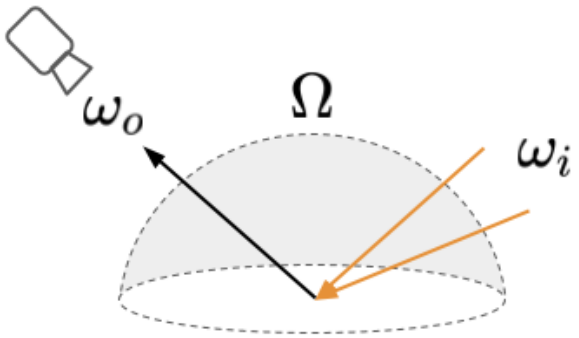

Figure 12: Rendering equation illustration.

2.2.4 Rendering Equation

The explanation above for materials and BRDFs was assuming a single incident light ray. In reality, we have many light rays coming onto a surface point simultaneously, and the surface point might be emitting light as well. Therefore, to compute the outgoing light in any direction, we need to sum up the contribution of each of these components (see Fig. 12). This will give us the following integral over the hemisphere of the surface point:

$$L_{\mathrm{o}}(\mathbf{x}, \omega_{\mathrm{o}}, \lambda) = L_{\mathrm{e}}(\mathbf{x}, \omega_{\mathrm{o}}, \lambda) \; + \int_{\Omega} f_{\mathrm{r}}(\mathbf{x}, \omega_{\mathrm{i}}, \omega_{\mathrm{o}}, \lambda) L_{\mathrm{i}}(\mathbf{x}, \omega_{\mathrm{i}}, \lambda)(\omega_{\mathrm{i}} \cdot \mathbf{n}) \, \mathrm{d}\, \omega_{\mathrm{i}}$$

where L_{o} is the total outgoing light of wavelength λ along the direction ω_{o} at a surface point at location $\mathbf{x}$, L_{e} is the emitted light of wavelength λ by the surface point at $\mathbf{x}$, Ω is the unit hemisphere about the surface normal $\mathbf{n}$ covering all possible incoming light directions, and L_{i} is the incoming light onto surface point $\mathbf{x}$ of wavelength λ and from direction ω_{i}. The above equation is called the rendering equation. Solving the rendering equation is the most challenging and compute intensive part of PBR. For any non-trivial scene, this integral is intractable and thus current approaches rely on numerical integration methods for estimating this integral.

2.3 Dense Visual SLAM

The major goal for dense visual SLAM pipelines is to accumulate temporal sensor data into a single consistent 3D model (the *global map*), while simultaneously recovering the location of the agent within that map (see Fig. 13). In this chapter we give a very broad overview of dense visual SLAM methods that operate on a sequence of RGB-D inputs. The SLAM components that are more relevant to our work are explained in

Figure 13: An example surfel-based map reconstruction using dense visual SLAM. Figure courtesy of [2].

more detail in Chapter 3.

The first step for dense visual SLAM is acquiring the sensor data. Most approaches require depth measurements as input. Depth measurements can come from various sources, such as Time-of-Flight (ToF) sensors or stereoscopic approaches. Second, a *data preprocessing* step will remove depth outliers and filter the raw depth maps for better final reconstructions. Next, we need to find the position and orientation of each RGB-D frame with respect to the built global map (*localization*), and finally we need to *map* all this data into our global map, which possibly involves a *fusion* step to reduce the final map size.

2.3.1 Localization

In order to transform the data from multiple RGB-D images into a single coordinate system, we need to find the position and orientation of the sensor at the time of capturing each RGB-D image (localization). One way of doing this localization step is to *register* the depth map of the current frame (*live frame*) with the global map using the Iterative Closest Point (ICP) algorithm. To do this, we start by converting the live frame's depth measurement into $\mathbb{R}^3$ point coordinates in the 3D camera coordinate system (*vertex map*):

$$V_t(\mathbf{u}) = D_t(\mathbf{u})\mathbf{K}^{-1}[\mathbf{u}^T, 1]^T \in \mathbb{R}^3$$

Figure 14: Iterative closest point algorithm. Minimizing the error function moves the source curve (pink) towards the target curve (blue).

where V_t and D_t are the vertex map and depth map for the live frame at time step t respectively, $\mathbf{u} = [x, y]^T \in \mathbb{R}^2$ is the 2D pixel position, and $\mathbf{K}$ is the camera calibration intrinsics matrix. We then need to align these 3D points with our current global map. One way of doing this is to align the live frame vertices with a subset of the global map points using ICP.

Iterative Closest Point. ICP has become a widely used approach for aligning 3D models based on their geometry. This algorithm works by iteratively optimizing the rigid transformation that aligns a *source pointcloud* with the *target pointcloud* in order to minimize an error metric (see Fig. 14). One example of an error metric could be using the Chamfer Distance (described in Sec. 2.1.2) between the source pointcloud and the target pointcloud. The error can then be minimized using a least squares optimizer.

In the context of localization, one simple approach is to use ICP for *pairwise registration* between the vertex map of the live frame and the previous frame to recover the relative rigid transformation between them. An important consideration is that ICP requires the initial rigid transformation for the optimization process to provide enough overlap between the two pointclouds.

2.3.2 Mapping

Once we have localized the live frame, we can proceed to build and update the global map by merging in the points from the live frame. The key insight here is that we want to add (*merge*) information from the live frame to the global map to fill in the global map gaps, and we want to *fuse* the information across different frames that correspond to the same point in space together. The exact method of this mapping step is dependent on the 3D representation of our global map (signed-distances, pointclouds, surfels, etc.).

Chapter 3

`gradslam`: Differentiable Dense SLAM Library

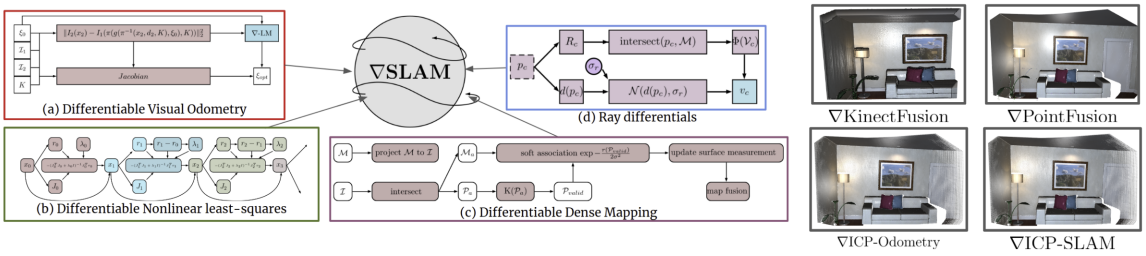

Figure 15: An overview of the differentiable dense SLAM components that ∇SLAM (gradSLAM) [9] proposes. Figure courtesy of ∇SLAM [9].

3.1 Introduction

Simultaneous Localization and Mapping (SLAM) is the problem of building and updating the map of an unknown space using input sensory data, while simultaneously recovering the location and orientation of the sensor in the map. Several SLAM systems have demonstrated an ability to reconstruct a 3D dense map of the environment from sequences of RGB-D data [16–18]. Due to a lack of consensus on the right 3D representation, each of the aforementioned approaches use a different type of map representation (e.g., pointclouds, surfels, voxels).

Over the last decade, deep neural networks have changed the landscape of several domains (e.g., object detection [19], natural language processing [65], audio modeling [66]). This has opened the following two questions: 1) How can SLAM be leveraged to improve the modeling tasks of deep neural networks?, and 2) How can deep neural networks be leveraged in improving SLAM systems? One of the main challenges in addressing both aforementioned questions is that current dense SLAM systems are

not differentiable. ∇SLAM (gradSLAM) [9] addresses this by proposing differentiable counterparts to several commonly used components in modern dense SLAM systems: mapping, raycasting, global measurement fusion, and trust region optimizer for nonlinear least squares problems. This enables multiple SLAM systems (KinectFusion [16], PointFusion [18], ICP-SLAM [9]) to be realized as fully differentiable systems by replacing their non-differentiable components with the differentiable counterparts of ∇SLAM [9].

The differentiability of ∇SLAM [9] enables gradients from the SLAM output (map and trajectory) to be backpropagated through the SLAM components, and into the input sensory data (RGB-D images). Intuitively, the first-order derivative of a reconstructed dense map $\mathcal{M}$ with respect to the sensory data s informs us how sensitive the reconstructed map is to each sensor measurement.

To foster further research on differentiable SLAM systems and their applications to spatially-grounded learning, we release `gradslam` as the first open-source library for differentiable SLAM. `gradslam` is a PyTorch [67] based library in Python which supports multiple dense SLAM systems out-of-the-box, in addition to a high-level interface that enables mixing and matching between different types of odometry and mapping algorithms. Our project page and code can be accessed at https://gradslam.github.io. To set the stage for what the `gradslam` library contains, we provide the required backgrounds on the components of ∇SLAM [9]. Our main contribution in this section is in developing the `gradslam` library and parts of the experiments on ∇SLAM.

3.2 Related Work

Machine learning advances have been applied to SLAM in several works through the reformulation of a subset of *components* of the full SLAM system in a differentiable manner. The contents of the following brief review of learning-based SLAM, differential visual odometry and differentiable optimization are largely borrowed from our paper [9].

Learning-based SLAM approaches. CodeSLAM [33] and SceneCode [34] are two examples of learning-based SLAM which attempt to represent scenes using compact *codes* that represent 2.5D depth map. DeepTAM [13] trains a tracking network and a mapping network, which learn to reconstruct a voxel representation from a pair of images. CNN-SLAM [14] extends LSD-SLAM [15], a popular monocular SLAM system, to use single-image depth predictions from a convolutional neural

network. Another recent trend has been to try to formulate the SLAM problem over higher level features such as objects, which may be detected with learned detectors [27] [28] [29]. DeBrandandere et al. [68] perform lane detection by backpropagating least squares residuals into a frontend module. Recent work has also formulated the passive [69] and active localization problems [70, 71] in an end-to-end differentiable manner. While all these approaches try to leverage differentiability in submodules of SLAM systems (eg. odometry, optimization, etc.), none of these provide a single framework that models an entire SLAM pipeline as a differentiable graph.

Differentiable visual odometry. The beginnings of differentiable visual odometry can be traced back to the seminal Lucas-Kanade iterative matching algorithm [36]. Kerl *et al.* [37][1] apply the Lucas-Kanade algorithm to perform real-time dense visual odometry. Their system is differentiable, and has been extensively used for self-supervised depth and motion estimation [38–40]. Coupled with the success of Spatial Transformer Netowrks (STNs) [41], several libraries (gvnn [42], kornia [43]) have since implemented these techniques as differentiable *layers*, for use in neural networks.

However, extending differentiability beyond the two-view case (*pairwise registration*) is not straightforward. Global consistency necessitates fusing measurements from live frames into a global model (*model-frame alignment*), which is not trivially differentiable.

Differentiable optimization. Optimization is a core requirement of any SLAM solution. Some approaches have recently proposed to learn the optimization of nonlinear least squares objective functions. This is motivated by the fact that similar cost functions have similar loss landscapes, and learning methods can help converge faster, or potentially to better minima.

In BA-Net [73], the authors learn to predict the damping coefficient of the Levenberg-Marquardt optimizer, while in LS-Net [74], the authors entirely replace the Levenberg-Marquard optimizer by an LSTM network [75] that predicts update steps. In GN-Net [76], a differentiable version of the Gauss-Newton loss is used to show better robustness to weather conditions. RegNet [77] employs a learning-based optimization approach based on photometric error for image-to-image pose registration. However, all the aforementioned approaches require the training of additional neural nets and this requirement imposes severe limitations on generalizability.

Concurrently, Grefenstette *et al.* [78] propose to unroll optimizers as

[1]The formulation first appeared in Steinbrüker *et al.* [72].

computational graphs, which allows for computation of arbitrarily higher order gradients. Our proposed differentiable Levenberg-Marquardt optimizer is similar in spirit, with the addition of gating functions to result in better gradient flows.

In summary, the goal of modeling the entire SLAM pipeline as a differentiable model is the motivation that underlies ∇SLAM [9].

3.3 Method

In this section we present the necessary background from ∇SLAM [9] to set the stage for the differentiable sub-components contained in our `gradslam` library.

Overview. As previously mentioned, several of the components in modern dense SLAM systems [16, 18] are not differentiable (i.e., the gradients will be unspecified). Examples of these non-differentiable components include non-linear least squares solvers, raycasting, and discretizations. Furthermore, some operations in current SLAM systems are differentiable, however, their gradients are zero *almost everywhere* (e.g., indexing, sampling) causing sparse gradients.

∇SLAM [9] makes every computation in subsystems of SLAM realised as a composition of differentiable functions. Broadly, the modules commonly used in a dense SLAM system can be termed as *odometry estimation* (frame-to-frame alignment), *map building* (model-to-frame alignment), and *global optimization*. An overview of the differentiable components of ∇SLAM [9] is shown in 3.

We describe the issues that cause the non-differentiability in each of the aforementioned modules, and the differentiable counterparts proposed by ∇SLAM [9]. Then, we describe how ∇SLAM [9] pieces the aforementioned differentiable components together to realise differentiable variants of several classic dense mapping systems.

3.3.1 ∇LM: A Differentiable Nonlinear Least Squares Solver

Most modern SLAM systems rely on optimizing a non-linear least squares objective at one or more stages of their pipeline. Example tasks that require solving a non-linear least squares objective include frame-to-frame alignment (e.g., depth measurement registration using ICP), and pose-graph optimization. Non-linear least squares solvers such as Gauss-Newton (GN) or Levenberg-Marquardt (LM) solvers can be used for optimizing such objective functions. LM solvers are the most common choice in modern SLAM systems as they are more robust than GN solvers.

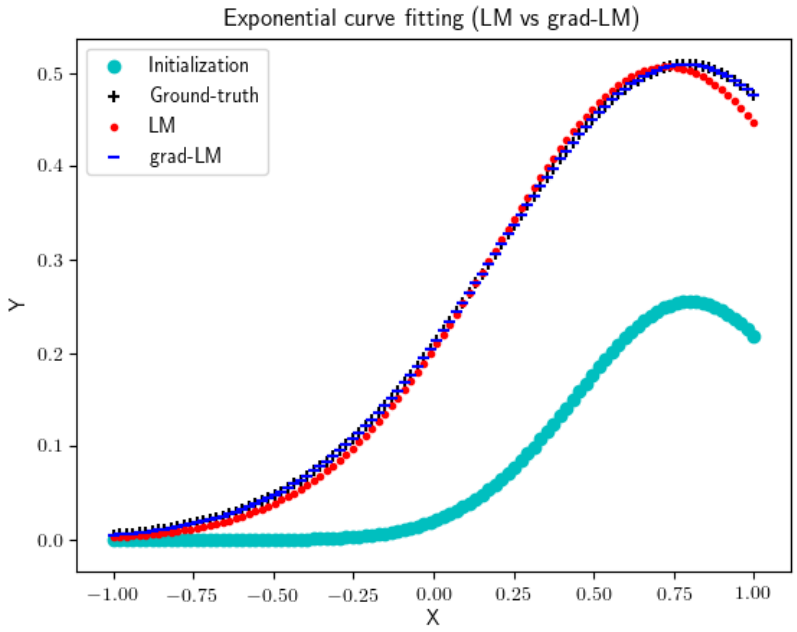

Figure 16: Example toy curve fitting problem. ∇LM which is a smooth reparameterization of the non-differentiable LM solver performs similarly to LM, while being fully differentiable. Figure courtesy of ∇SLAM [9].

However, the LM solver is not differentiable. This is because the LM solver makes a *discrete* switching decision between damping or undamping at each optimization step [79]. Moreover, the iterate is reverted to its previous value during undamping. ∇SLAM [9] proposes a *soft approximation* of the damping mechanism to realise differentiabile LM solvers. The key insight is that given the norm of the error at the current iterate $\mathbf{r_0}$ and at the *lookahead* iterate $\mathbf{r_1}$, $\mathbf{r_1} - \mathbf{r_0}$ determines whether to damp or undamp. Moreover, when we undamp, the iterate is reverted to its previous value. Thus, ∇SLAM [9] proposes the following smooth *gating* functions Q_x and Q_λ for updating the damping coefficient and the iterate:

$$\begin{aligned} \lambda_1 = Q_\lambda(r_0, r_1) &= \lambda_{min} + \frac{\lambda_{max} - \lambda_{min}}{1 + De^{-\sigma(r_1 - r_0)}} \\ Q_x(r_0, r_1) &= x_0 + \frac{\delta x_0}{1 + e^{-(r_1 - r_0)}} \end{aligned} \tag{1}$$

where λ_{min} and λ_{max} are the minimum and maximum possible values for the damping coefficient respectively, D and σ are parameters that adjust the falloff [80], and x_0 is the current iterate. This smooth approximation of the LM solver allows it to perform near-identical to the original non-differentiable LM solver while being fully

differentiable (see Fig. 16).

3.3.2 Differentiable Mapping

As the *global* map $\mathcal{M}$ is being constructed (in the reference coordinate system of the first frame I_0), the surface measurement of the live frame I_k at time k needs to be aligned with the global map $\mathcal{M}$. Generally, the *surface alignment* process consists of several differentiable but non-smooth operations (i.e., thresholding, indexing, clipping, active/inactive decision, etc.). The sparse nature of the gradients of the aforementioned operations causes the computation graph to have undefined gradients almost everywhere. ∇SLAM [9] alleviates this issue by approximating these functions with locally smooth alternatives. The following corrective measures are proposed by ∇SLAM [9]:

1. The surface measurement of every valid pixel p in the live frame I_k is computed via a bilinear interpolation *kernel* K, causing the output to be a function of that pixel p and its neighbours.

2. For associating the live frame I_k surface measurements with the global map $\mathcal{M}$ elements, a *soft* association to a subset of global map $\mathcal{M}$ elements is used. Specifically, for each point P in the live frame surface measurement, the subset of closes candidate points is found in a region $exp\left(-\frac{r(P)^2}{2\sigma^2}\right)$, where σ controls the falloff, and $r(P)$ is the distance from the viewing ray.[2]

3. By default, every surface measurement in the live frame I_k is assumed to represent a new global map element, which is then passed to the *differentiable map fusion* component (*cf.* Sec 3.3.3).

3.3.3 Differentiable Map Fusion

Using only the aforementioned mapping strategy for building a dense global map causes the number of elements in the map to grow in proportion to the exploration time. This is an undesirable property as ideally the map should only grow in proportion to the explored volume of occupied space. Classical dense mapping approaches solve this by *fusing* redundant observations of the same map element [16, 18]. This reduces the size of the recovered map, and also improves the reconstruction quality. Most fusion approaches are differentiable [16, 18], however,

[2]This is a well-known falloff function commonly used with Kinect-style depth sensors [18,81,82].

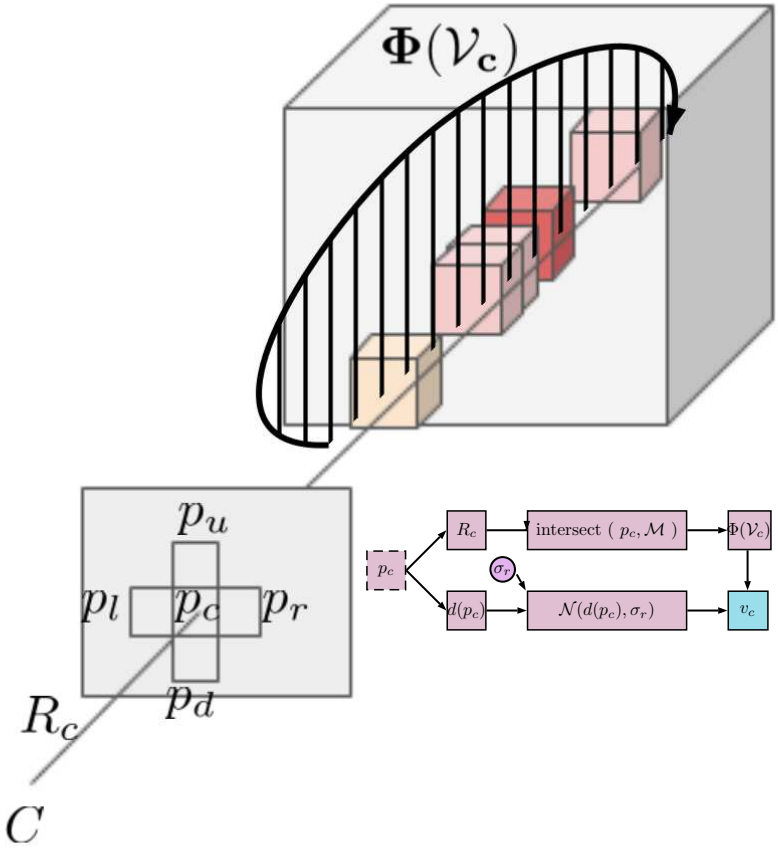

Figure 17: Finite differences over ray potentials. Figure courtesy of ∇SLAM [9].

they are not smooth as they use thresholds that cause an abrupt change in gradient flow at the truncation point. ∇SLAM [9] proposes to use a logistic falloff function, similar to Eq. (1), to ease gradient flow through these truncation points.

3.3.4 Differentiable Ray Backprojection

Several dense SLAM approaches raycast the global map $\mathcal{M}$ into the live frame coordinates for global pose estimation [16, 17]. This is done by backprojecting the viewing ray through each live frame pixel into the global map, and finding the intersect with the first map element by marching along the ray. Similar to DRC [83] and WS-GAN [84], ∇SLAM [9] uses a pooling operation over all voxels pierced by the ray to compute the ray *potential*. However, ∇SLAM [9] weighs this pooling mechanism by a Gaussian falloff about the depth measurement of the pierced live frame pixel.

Specifically, given a ray R_c which pierces the pixel p_c and a set of global map voxels $\mathcal{V}_c = \{v_c\}$, then the *aggregated value* of the ray is the normalized sum of every value $\psi(v_c)$ multiplied by the Gaussian fall-off value at v_c:

$$\Phi(\psi(v_c)\ \forall v_c \in \mathcal{V}_c) \tag{2}$$

where Φ denotes the aggregation function. Moreover, ∇SLAM [9] uses the finite difference based ray differentials proposed by Igehy *et al.* [85] to compute the derivative of the ray potential with respect to the pixel neighbourhood. In particular, the partial derivative $\frac{\partial v_c}{\partial c}$ can be approximated as:

$$\frac{\partial v_c}{\partial p_c} = \begin{pmatrix} (v_r - v_l)/2 \\ (v_u - v_d)/2 \end{pmatrix} \tag{3}$$

where v_r, v_l, v_u, and v_d are the *aggregated values* corresponding to the p_r, p_l, p_u and p_d pixels shown in Fig. 17 respectively.

3.4 Case Studies

To demonstrate the applicability of the ∇SLAM [9] framework, we leverage the differentiable computation graphs specified in Sec 3.3 and compose them to realise two practical SLAM solutions. In particular, we implement differentiable versions of the *PointFusion* [18] algorithm that constructs surfel maps, and a pointcloud-only SLAM framework called *ICP-SLAM*.

3.4.1 PointFusion

We implement PointFusion [18], which incrementally fuses surface measurements to obtain a global surfel map. Surfel maps compare favourably to volumetric maps due to their reduced memory usage. We closely follow our differentiable mapping formulation (*cf.*Sec 3.3.2) and use surfels as map elements. We adopt the fusion rules from [18] to perform map fusion.

3.4.2 *ICP-SLAM*

As a baseline example, we implement a simple pointcloud based SLAM technique which uses ICP to incrementally register pointclouds to a global pointcloud set. In particular, we implement two systems. The first one aligns every pair of consecutive

incoming frames to obtain an odometry estimate (also referred to as *pairwise registration* or ICP-Odometry). The second variant performs what we call *frame-to-model alignment* (ICP-SLAM). That is, each incoming frame is aligned (using ICP) with a pointcloud containing the entire set of points observed thus far.

3.5 Open-source Library

Figure 18: `gradslam` library teaser. An example reconstruction result using `gradslam` library.

To ensure reproducibility and to foster further research that leverages differentiability in SLAM systems, we make available `gradslam`[3], an open-source library for differentiable SLAM.

Overview. `gradslam` is an open-source Python library aimed at providing implementations of SLAM subsystems for deep learning practitioners. This is the first freely available dense SLAM implementations natively written in a deep learning framework such as TensorFlow [86] or PyTorch [67]. While a few existing packages provide some overlapping functionality (e.g. Kaolin [87], Kornia [43] and PyTorch3d [88] in PyTorch [67] and TensorFlow Graphics [89] in TensorFlow [86]), the scope for those libraries is broader. Importantly, full-fledged SLAM does not seem to be a near-term feature for these libraries.

Implemented in PyTorch [67], `gradslam` provides a high-level interface for designing SLAM systems by mixing and matching between mapping and odometry

[3]https://github.com/gradslam/gradslam

algorithms. `gradslam` also makes it easy to implement new SLAM systems by providing structures and boiler-plate code. Below is an example snippet of using `gradslam` to run PointFusion [18] on a sub-sequence from the ScanNet [11] dataset.

```
import gradslam as gs
from gradslam.datasets import Scannet
from torch.utils.data import DataLoader

# load rgbd sequence
dataset = Scannet("Scannet/scans", "Scannet/associations")
loader = DataLoader(dataset=dataset, batch_size=2)
colors, depths, intrinsics, poses, *_ = next(iter(loader))

# perform pointbased fusion
rgbdimages = gs.RGBDImages(colors, depths, intrinsics, poses)
slam = gs.slam.PointFusion(odom="icp")
recovered_map, recovered_poses = slam(rgbdimages)
```

Library structure. The `gradslam` library is composed of the following submodules:

gradslam.structures - A collection of high-level structures for storing and manipulating batched raw sensor data (e.g. RGB-D images) and reconstruction data (e.g. pointclouds). Each structure is a class which contains a set of PyTorch Tensors for efficient computations such as rigid-body transformations. Currently supported structure types are pointclouds and RGB-D images. For storing batches of variably sized pointclouds in a tensor, we adopt PyTorch3d's [88] padded representation by zero padding pointclouds as necessary to have them all contain the same number of points. We intend to expand the supported structures to TSDF volumes.

gradslam.datasets - A collection of PyTorch [67] `Dataset` subclasses for reading sensor data (images, calibration parameters, etc) from popular SLAM datasets. Each dataset handles the parsing, file handling and preprocessing of data. Currently supports ScanNet [11], TUM [10], and ICL-NUIM [90]. We intend to expand dataset support to include Rosbags and raw folder dataset format.

gradslam.geometry - A collection of functions for basic geometric operations, such as rigid-body transformations, projection and unprojection of points, homogenization and unhomogenization.

gradslam.odometry - A collection of classes for different types of odometry algorithms. An *odometry provider* is used within a SLAM module to

provide up-to-date odometry information. This module contains an abstract `OdometryProvider` class that users can inherit from to implement their own odometry algorithm. Currently implemented odometry providers are "ground-truth" odometry which provides the relative transformation between two frame poses, and ICP provider which uses point-to-plane ICP on the vertex-normal maps of consecutive frames to estimate the relative transformation.

gradslam.slam - A collection of subclasses of PyTorch `nn.Module` for different SLAM algorithms. Each module has two routines: initialize and forward. At initialization, each module takes an `OdometryProvider` object as well as the SLAM system parameters. At each `forward` call, the module takes sensor data (e.g., RGB(-D) images, and optional IMU poses) as inputs, and outputs maps and (optionally) camera poses. Currently supports PointFusion [18] and *ICP-SLAM*. We intend to expand support for KinectFusion [16].

3.6 Experiments

3.6.1 Qualitative Results

∇SLAM [9] works out of the box on multiple RGB-D datasets. Specifically, we present qualitative results of running the aforementioned differentiable dense SLAM systems on RGB-D sequences from the TUM RGB-D dataset [10], ScanNet [11], as well as on an in-house sequence captured from an Intel RealSense D435 camera.

Fig. 19- 21 show qualitative results obtained by running ∇SLAM on a variety of sequences from the TUM RGB-D benchmark (Fig. 19), ScanNet (Fig. 20), and an in-house sequence (Fig. 21). These differentiable SLAM systems all execute fully on the GPU, and are capable of computing gradients with respect to *any* intermediate variable (e.g., camera poses, pixel intensities/depths, optimization parameters, camera intrinsics, etc.)[4].

3.6.2 Pixel Contributions to Global Map

The differentiability of ∇SLAM allows us to back-trace every global map element to the frame pixels that generated it. Let F_i be an input RGB-D frame at time frame i, and $\mathcal{M}$ be the original global map that is constructed from all frames. We create the perturbed map $\tilde{\mathcal{M}}$ by perturbing a representation unit of $\mathcal{M}$. We then compute

[4]Qualitative results were obtained by Krishna Murthy Jatavallabhula and Ganesh Iyer.

Figure 19: Reconstructed pointcloud by running ∇ICP-SLAM on a sequence from the TUM RGB-D dataset [10]. Figure courtesy of ∇SLAM [9].

the error between $\mathcal{M}$ and $\tilde{\mathcal{M}}$ by use of some metric Φ: $e = \Phi(\mathcal{M}, \tilde{\mathcal{M}})$ (in this case, we used the Chamfer Distance). By computing $\frac{\partial e}{\partial F_i}$ we can measure the contribution of every pixel from any frame i to the perturbed element of the global map. An example of this for ∇PointFusion is shown in fig. 23, where an interactive tool is used for selecting global map points and visualizing the contribution of input pixels.

3.6.3 RGB-D Denoising

In the above setup (Fig. 23), only one pixel (the highlighted pixel) is perturbed at a particular instant. To extend the analysis to multiple perturbed pixels, we *jitter* the depth map and the RGB image by adding uniform noise to each pixel (in the range $[0, 0.2]$ for depth channels and $[0, 255]$ for RGB channels). This noise is added to the final frame of each sequence (we use a sequence length of 4 frames in all our experiments). This sequence of frames is reconstructed by ∇PointFusion and compared with the true map (here, we use a PointFusion [18] reconstruction on the clean images as the true map). Chamfer distance is used to compare the noisy and the true pointclouds for similarity. This error is then backpropagated through ∇SLAM, all the way to input RGB pixel intensity measurements and per-pixel depth measurements. We use the Adam [91] optimizer to update these pixel intensities and depths for 400 iterations, with a learning rate of 0.05 (and the PyTorch defaults for β_1 and β_2). Fig. 22 presents qualitative results for the RGB-D denoising experiments.

Figure 20: Qualitative results of running several differentiable SLAM systems (∇KinectFusion, ∇PointFusion, and ∇ICP-SLAM [9]) on the ScanNet [11] dataset sequences. Due to GPU memory constraints, only parts of the scene were reconstructed. Reconstruction results of BundleFusion [12] are also shown for reference. Figure courtesy of ∇SLAM [9].

The top half of Fig. 22 presents qualitative results with another initialization scheme, where the depth values at each pixel are replaced by the average depth of the entire image. As one would expect, this initial guess is farther away from the true solution, and given a fixed computation budget (400 iterations), additive noise initialization yields better convergence results.

3.6.4 RGB-D Image Completion

We also present results on an RGB-D image completion task. The overall setup is similar to that of the RGB-D denoising task, with the difference that in image completion, the entire image is replaced by uncorrelated uniform noise samples, as

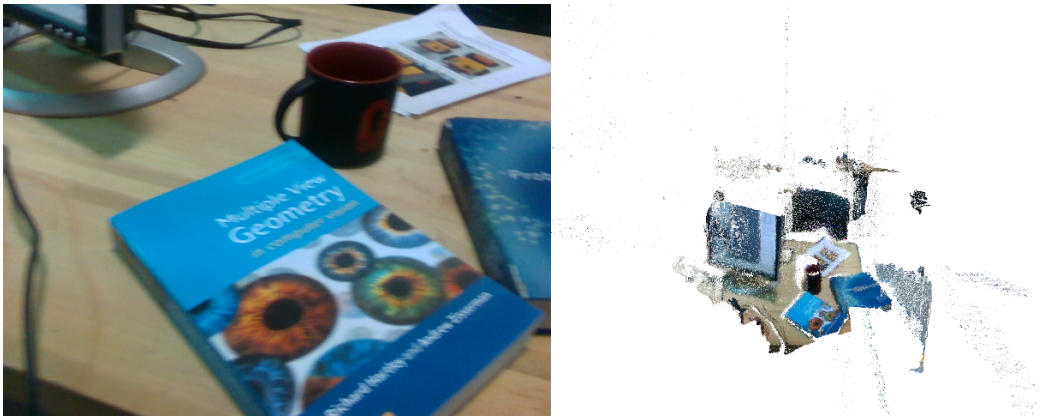

Figure 21: Reconstructed map by running ∇PointFusion [9] (right) on an in-house sequence captured using an Intel RealSense depth camera D435 (left). Note that the reconstruction was obtain without performing any noise removal. Figure courtesy of ∇SLAM [9].

opposed to an additive noise used for the denoising experiment. Fig. 25 presents qualitative results for the RGB-D completion task.

3.6.5 Occluder Gradients

In Fig. 24, occluders (top row) and pixel noise (bottom row) are introduced in one of the depth maps of a sequence and the scene is reconstructed using ∇PointFusion. The Chamfer Distance between the noisy and ground truth reconstructed maps is then calculated and backpropogated to each pixel. Minimizing the Chamfer Distance using Adam [91] optimizer recovers the depth information at the noisy and occluded regions[5].

3.7 Conclusion

We introduce `gradslam`, the first differentiable dense SLAM library that enables gradient-based learning for a large set of localization and mapping based tasks, by providing explicit gradients with respect to the input image and depth maps. We demonstrate a diverse set of case studies, and showcase how the gradients propogate throughout the tracking, mapping, and fusion stages. Future efforts will include adding differentiable counterparts to other common SLAM components into the `gradslam` library, and demonstrating usecases of the `gradslam` library for being plugged into downstream learning tasks.

[5] "Occluder Gradients" experiments done together with Krishna Murthy Jatavallabhula.

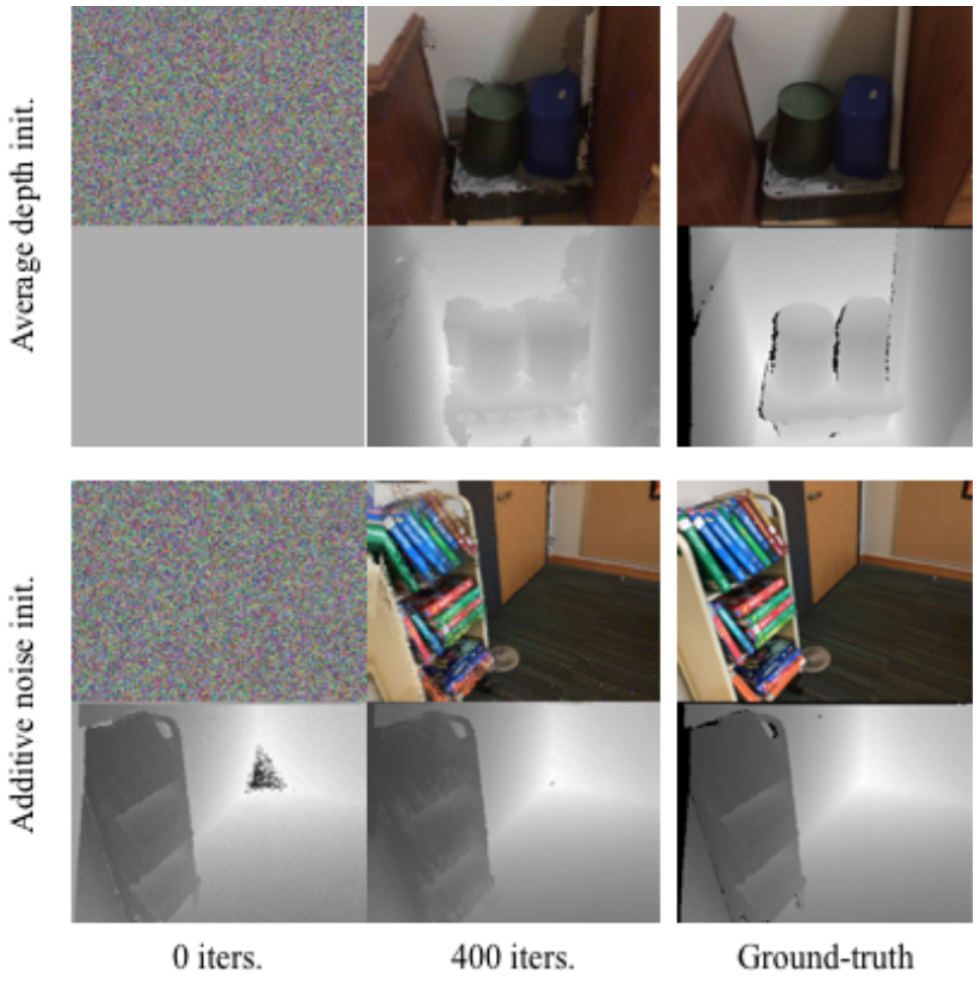

Figure 22: RGB-D denoising experiment. (Top/Bottom) The last RGB-D frame from a sequence is perturbed by adding white uniform noise to every RGB and depth pixel (left). The sequence is reconstructed using ∇PointFusion on this noisy frame, and compared to a clean reconstruction from the un-perturbed RGB-D image. The noisy and the clean maps are compared using Chamfer distance, and the gradients are backpropagated to the image. These gradients are used to update the input RGB-D image, and the optimized result is shown in the middle column. The ground-truth RGB-D image is shown on the right column for reference.

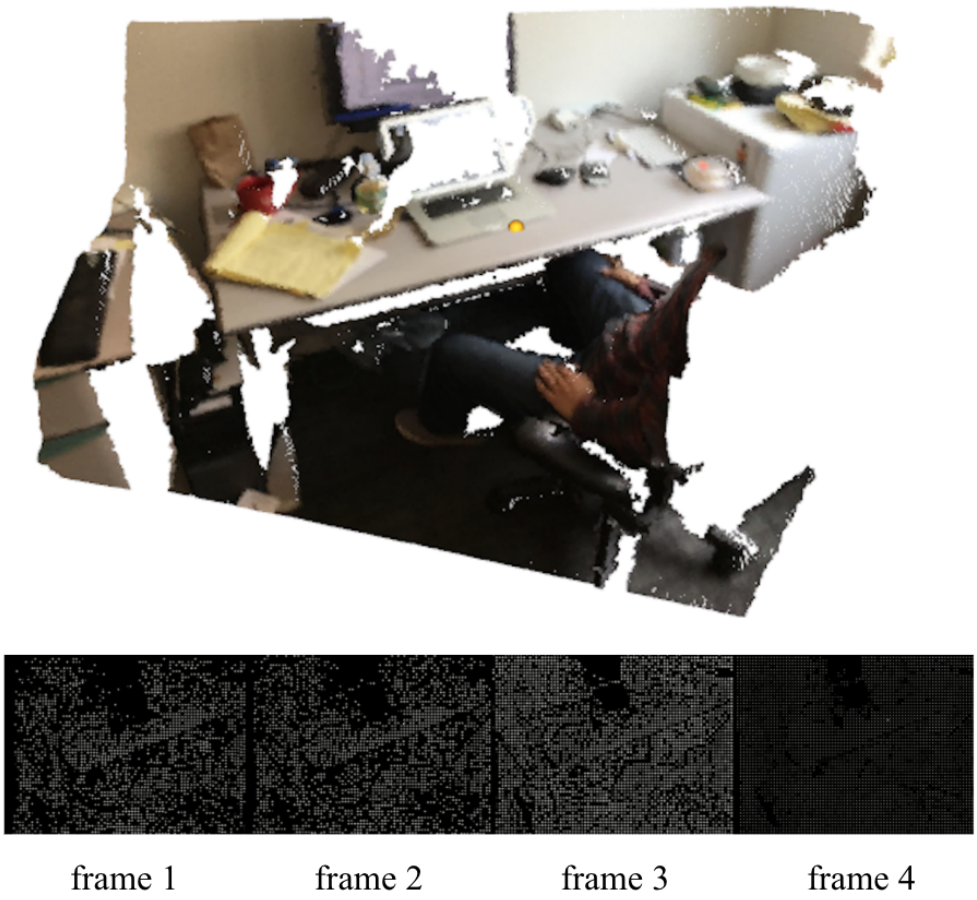

Figure 23: Interactive pixel contribution visualization. ∇SLAM allows us to "*backprop all the way from 3D maps to 2D pixels*" [9] and we use this to visualize the contribution of each pixel from an input frame to the eventual 3D map reconstructed (here, from 4 RGB-D images). The yellow sphere on the laptop's touchpad (top image) is selected from a pointcloud map generated by ∇PointFusion. In the bottom row, we show the gradient with respect to each frame when the selected point is perturbed slightly.

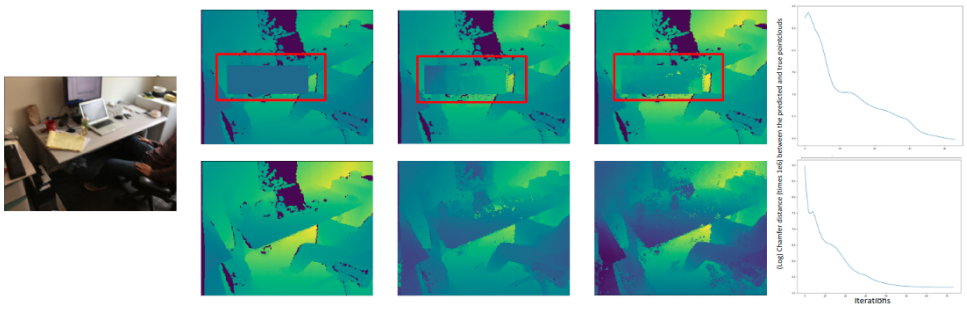

Figure 24: End-to-end gradient propagation. (*Top*): A chunk of a depth map is *chopped*. The resultant sequence is reconstructed using ∇PointFusion and the pointcloud is compared to a *clean* one reconstructed using the unmodified depth map. The Chamfer distance between these two pointclouds is used to define a reconstruction error between the two clouds, which is backpropagated through to the input depth map and updated by gradient descent. (*Bottom*): ∇SLAM [9] can *fill-in* holes in the depthmap by leveraging multi-view gradient information. Figure courtesy of ∇SLAM [9].

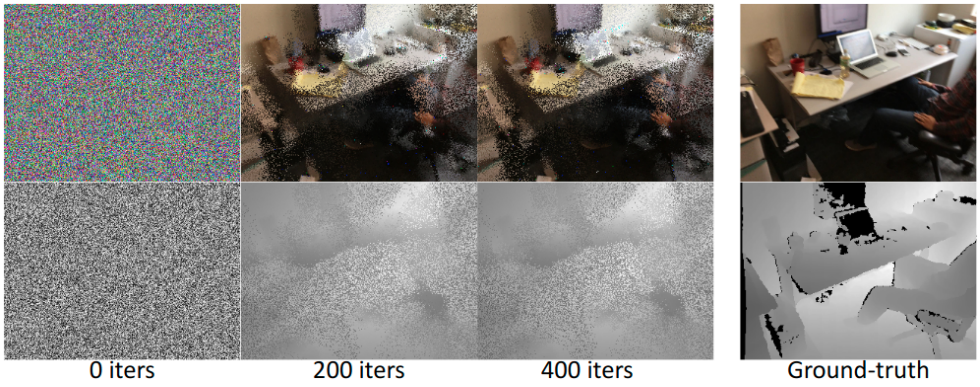

Figure 25: RGB-D completion using end-to-end gradient propagation. Three RGB-D images and a *noise image* are passed through ∇PointFusion, and compared to a clean reconstruction obtained from four RGB-D images. The reconstruction loss is used to optimize the *noise image* by gradient descent. Most of the original RGB and depth information is recovered through this optimization task. Note that finer features are hard to recover from a random initialization, as the overall *SLAM function* is only locally differentiable. Figure courtesy of ∇SLAM [9].

Chapter 4

Disentangled Rendering Loss

4.1 Introduction

In order to digitally replicate the appearance of real-world materials using computer graphics, accurate SVBRDF parameters are used in a multi-variable rendering function. Recent deep learning techniques for SVBRDF estimation of near-planar surfaces train their network by taking advantage of the rendering function and using the *rendering loss*. The rendering loss is defined as the error between rendered images of ground truth and predicted material property maps summed over several sampled light and view directions. While training with this rendering loss improves the quality of the rendered outputs of the predictions, it has the following drawbacks: 1) Since the rendering function is many-to-one, incorrect material property maps can generate similar renderings under limited light and view conditions (i.e., the same color could result from different combinations of SVBRDF parameter values). Thus, models trained with this loss often tend to predict incorrect individual maps. 2) When the rendering loss for a pixel is non-zero, gradients are backpropagated to all BRDF parameters of that pixel, effecting changes even to correct predictions. 3) The rendering loss needs a heuristic in the number of light and view conditions to sample, which if not chosen correctly can affect accuracy and training time.

We first analyze the aforementioned problems and why they arise. We then propose a new loss function named as *disentangled rendering loss* which addresses the above issues by making the following modifications: For 1) and 2) the disentangled rendering loss takes as input only one predicted parameter in the rendering function at a time, while using ground truth inputs for the other parameters, and for 3) it removes the dependence on the light and view sampling heuristic by integrating the L1 loss of the rendering function over a subset of light and view directions over the hemisphere,

making network training independent of light conditions and view directions.

We believe that this is the first work to present ways for overcoming the problem of entangled SVBRDF parameters. Our disentangled rendering loss yields map predictions that are individually more accurate while also yielding similar high-quality re-renderings. We show this by comparing recovered material properties qualitatively and quantitatively with those recovered using the standard rendering loss.

4.2 Related Work

Of late, deep learning models have shown a lot of promise in reflectance modeling from images in the wild [22–25]. For a detailed review of these approaches, we suggest the excellent recent survey [92]. Li *et al.* [25] propose a CNN architecture for predicting the reflectance properties of a single captured image under unknown natural illumination. They train a separate network for each material type (plastic, wood, and metal) and each output map (diffuse albedo, normal, specular albedo, and roughness) with the traditional L2 loss over the predicted maps. However, directly minimizing the error on the maps was later shown to not lead to predicting very accurate SVBRDFs nor ground truth render reproductions by Deschaintre *et al.* [22].

Deschaintre *et al.* [22] instead found that training their SVBRDF recovery network with *rendering loss* as a better solution for predicting maps which give sharper and more accurate renders. While renders are accurate, their approach fails to recover accurate specular and diffuse maps compared to ground truth due to entanglement of material properties.

Currently, the most accurate SVBRDF map recovery techniques use multi-image deep networks [23,45]. These networks use multiple images of the same material under different light and view conditions as their input to provide more cues on what the SVBRDF should be. Gao *et al.* [45] propose a deep inverse rendering approach which can handle an arbitrary number of inputs by getting an initial SVBRDF estimate and then train an auto-encoder to optimize the SVBRDF in latent space to minimize the rendering loss. Their method then uses a final refinement stage to optimize the SVBRDF map directly. However, their approach requires the light and camera position for every input image to be known and performs an optimization process for each of these.

The recent work by Deschaintre *et al.* [23] uses an encoder-decoder architecture to output a 64 channel feature map for each input image given to the network. Aggregating these feature maps using max pooling and following it with a CNN

decoder then outputs the SVBRDF prediction. Similar to previous work by Gao *et al.* [45], they find that using a combination of L1 loss on the predicted maps and rendering loss during training helps stabilize the training procedure. However, the individual recovered SVBRDF maps still have inaccuracies and entanglement in the diffuse albedo and specular albedo maps. In our experiments, we decided to use their expertly designed network architecture, but train the network using our new loss definition, so that any effect in network training time and accuracy of predicted maps can be directly attributed to the new loss function.

Various fields of research have shown that disentangling parameters in complex tasks helps to train the network to better understand the problem, which then leads to the network learning more accurate solutions for unseen data. Some examples of disentangled tasks include learning from videos [93, 94], sentence generation [95], face image editing [96], deblurring of images [97], and facial expression recognition [98].

4.3 The Entanglement Problem

It should be noted that a single or a few images by themselves may not contain enough cues for one to be able to infer material properties precisely. Thus, recovering material properties from a single image, or even a few images, is an ill-posed problem. Arbitrarily increasing the number of input images with different view and light directions leads to larger data collection requirements but not necessarily better-quality results. Hence, one of the major goals in new research would be to recover more accurate property maps by training with a few casually captured images. As per our analysis of current deep learning solutions, there are a few causes for these inaccuracies arising from the way the loss function is defined.

- Emphasis in training is on rendered image similarity rather than material properties.
- No effort at disentangled learning of properties.
- Dependence on a few views for render comparison.

As mentioned earlier, rendering loss was shown to be more effective and hence gets used in all recent work [22–24, 45, 99]. Using this loss as opposed to the traditional L1 or L2 loss on predicted maps lets the physical meanings of each map and the interplay between them to be relegated to the update steps. Formally, the rendering

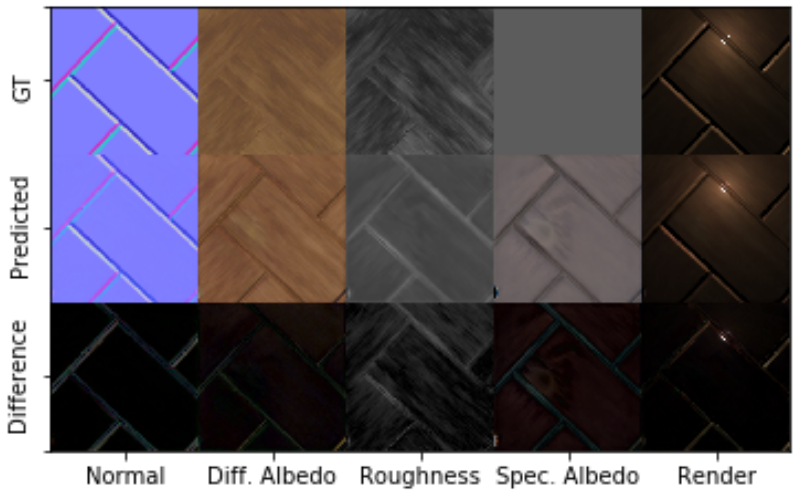

Figure 26: Using render loss leads to the recovery of maps which create similar renders to the ground truth, despite incorrect property maps.

loss is given by:

$$L_R(\vec{l}, \vec{v}) = |R_{N,D,R,S}(\vec{l}, \vec{v}) - R_{\hat{N},\hat{D},\hat{R},\hat{S}}(\vec{l}, \vec{v})| \tag{4}$$

Where $L_R(\vec{l}, \vec{v})$ is the rendering loss under some light direction $\vec{l}$ and view direction $\vec{v}$, $R_{N,D,R,S}(\vec{l}, \vec{v})$ is the rendering function parameterized by the 4 material maps N, D, R and S which are the predicted normal, diffuse albedo, specular roughness and specular albedo maps respectively, and $\hat{N}$, $\hat{D}$, $\hat{R}$ and $\hat{S}$ are the ground truths for those maps respecively. Since the rendering loss is light and view dependent, in practice the average of the rendering loss over multiple randomly sampled light and view directions is used for training. We note that this is the Monte Carlo method for approximating $\mathbb{E}_{\vec{l},\vec{v}}[L_R(\vec{l}, \vec{v})]$. This definition of the rendering loss has several major drawbacks.

Firstly, the rendering loss under limited light and view directions has multiple global minima. This is because two very different combinations of SVBRDF maps can generate the same rendering under limited light and view directions. As a direct implication of this, models trained with rendering loss tend to compensate for the incorrectness in one of the predicted maps by modifying another map in a way that would give a similar render. An example of this is shown in Fig. 26, where a model trained with rendering loss predicts a pinkish color as part of the specular map to compensate for the incorrect diffuse albedo and roughness map predictions.

Secondly, the many-to-one nature of the rendering function implies that the gradient is either zero or non-zero with respect to all 4 property maps. E.g., if the network correctly predicts three of the four parameter maps and has a mistake in one of them which causes the render to look different, the rendering loss will have

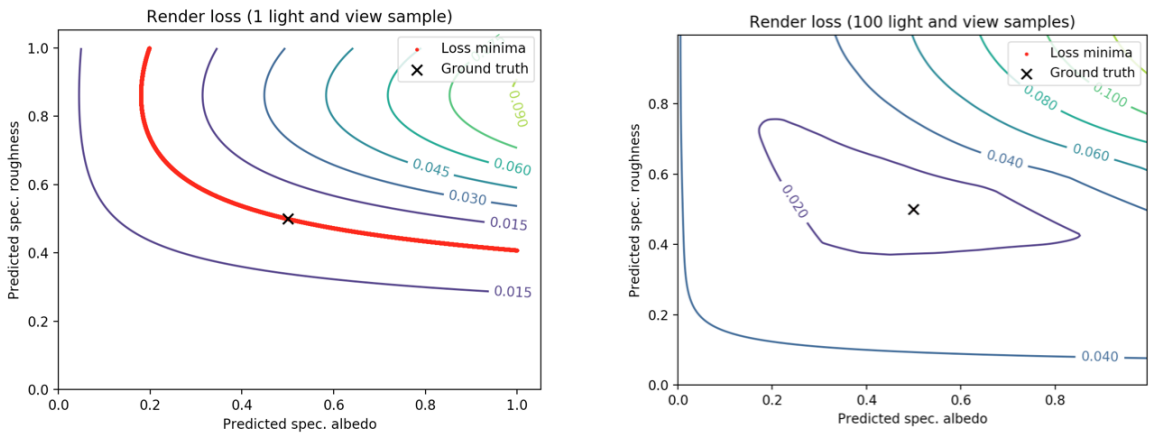

Figure 27: Contour plot of the rendering loss landscape with respect to the specular albedo and roughness. Multiple local minima exist when using one light and view sample (left). As we increase light and view samples, the problem of multiple local minima is reduced (right).

non-zero gradients with respect to all 4 maps, thus causing the network to *"forget"* about maps it has already learned to correctly predict.

Thirdly, the number of light and view directions is a heuristic that needs to be selected empirically. Sampling more light and view directions would make the approximation of $\mathbb{E}_{l,v}[L_R(l,v)]$ more accurate, albeit at the cost of more computation. Using a single render to compute loss with presents many loss minima possible (see Fig. 27). Therefore, most recent works heuristically use 9 renders to compute the loss with as they find it has a good computation to test render accuracy trade-off.

4.4 Method

We address the first two problems by simply parameterizing the rendering function with only one of the predicted maps at the time, while using ground truth maps for the rest of the maps. This change in rendering loss can be expressed as:

$$\begin{aligned} L_{DR} =& |R_{N,\hat{D},\hat{R},\hat{S}}(l,v) - R_{\hat{N},\hat{D},\hat{R},\hat{S}}(l,v)| \\ &+ |R_{\hat{N},D,\hat{R},\hat{S}} - R_{\hat{N},\hat{D},\hat{R},\hat{S}}| \\ &+ |R_{\hat{N},\hat{D},R,\hat{S}}(l,v) - R_{\hat{N},\hat{D},\hat{R},\hat{S}}(l,v)| \\ &+ |R_{\hat{N},\hat{D},\hat{R},S}(l,v) - R_{\hat{N},\hat{D},\hat{R},\hat{S}}(l,v)|. \end{aligned} \tag{5}$$

Note that the error on the diffuse map is not a function of light and view directions.

With this change, the error of each map can correctly be backtraced to that map while also considering the contribution of each map in the final rendering. We call this loss the disentangled rendering loss.

In order to avoid sampling multiple light and view conditions, we derive an analytical approximation for $\mathbb{E}_{l,v}[L_{DR}(l,v)]$. The complete derivation can be found in Sec. 4.6.3. The following simplifications were made to be able to derive a closed form solution for the integral:

1. The log of the specular term was used (as opposed to the specular term itself).
2. Light and view were assumed to have the same direction ($l = v$) with a uniform spread over the hemisphere.
3. $log(1 + x)$ was simplified to x in order to get a simple solution to the integral.
4. Since computing the expectation on the error of the normal map is not straight forward, we use an L1 loss on the normal map instead.
5. To make the implementation of the solution simpler, we use the upper bound of the error on the specular roughness map.

Using these simplifications, we obtain the following solution:

$$\begin{aligned} L_{IR} =& |N - \hat{N}| + \frac{|D - \hat{D}|}{\pi} + 2|\frac{1}{\hat{R}^2} - \frac{1}{R^2}| \\ &+ \frac{2}{3}|\hat{R}^4 - R^4| + |log(S) - log(\hat{S})| \end{aligned} \tag{6}$$

We denote this by L_{IR}, the integrated rendering loss. In addition to view independence, defining the loss this way gives us the following advantages:

- The major problem of not being able to correctly identify which map the error comes from is immediately fixed.
- The problem of the network predicting maps that have the same rendering but look different individually is also fixed.
- At the same time, the gradients for each map (except the normal map) continue to be computed through the rendering equation to express the role each map plays in the final rendered output, thus still providing us with nice sharp looking renders for the prediction map.

4.5 Experiments

4.5.1 Quality of Individual Predicted Maps

The primary goal of our experiments is to show that changing the loss function to our disentangled rendering loss enables us to recover more accurate material property maps. Hence, we adopt the same network architecture and training methodology as presented in the state-of-the-art multi-image SVBRDF recovery work [23]. After training the network for 300K iterations with each of the different rendering losses, we find that using our proposed rendering losses gives test set predictions with a higher SSIM due to better disentanglement of properties.

We test each trained network's ability to recover SVBRDF maps by inputting 10 renders using test set maps and then evaluating their predictions. Comparing the average SSIM error on the 200 sets of held-out property maps, presented in Table 1, shows that L_{IR} can recover better specular maps since the number of renders heuristic is not needed. In fact, L_{DR} also produces more accurate property map results on average than the original render loss even though it uses less FLOPs than the traditional rendering loss with 9 sampled light and view directions.

4.5.2 Overfitting Loss to One Sample

To better visualize and understand the implications of training with each of the losses, we trained the model to overfit to images rendered from a single SVBRDF map set while using the different loss functions. We then look at the predicted maps and their renderings for the same image that the model was overfitted to. This is shown in Fig. 28.

As can be seen, training the model on the rendering loss alone will cause the model to predict very inaccurate maps, although the renderings of these maps looks similar to the ground truth renderings. It can also be seen that much of the entanglement is between the predictions for the diffuse and specular albedo maps since these have the

	Property Maps				
	Normal	Diffuse	Roughness	Specular	Avg.
L_R	0.948	**0.861**	0.780	0.873	0.866
L_{DR}	**0.95**	0.839	**0.836**	0.887	**0.878**
L_{IR}	0.917	0.811	**0.836**	**0.908**	0.868

Table 1: Average SSIM on test set map predictions. Higher is better.

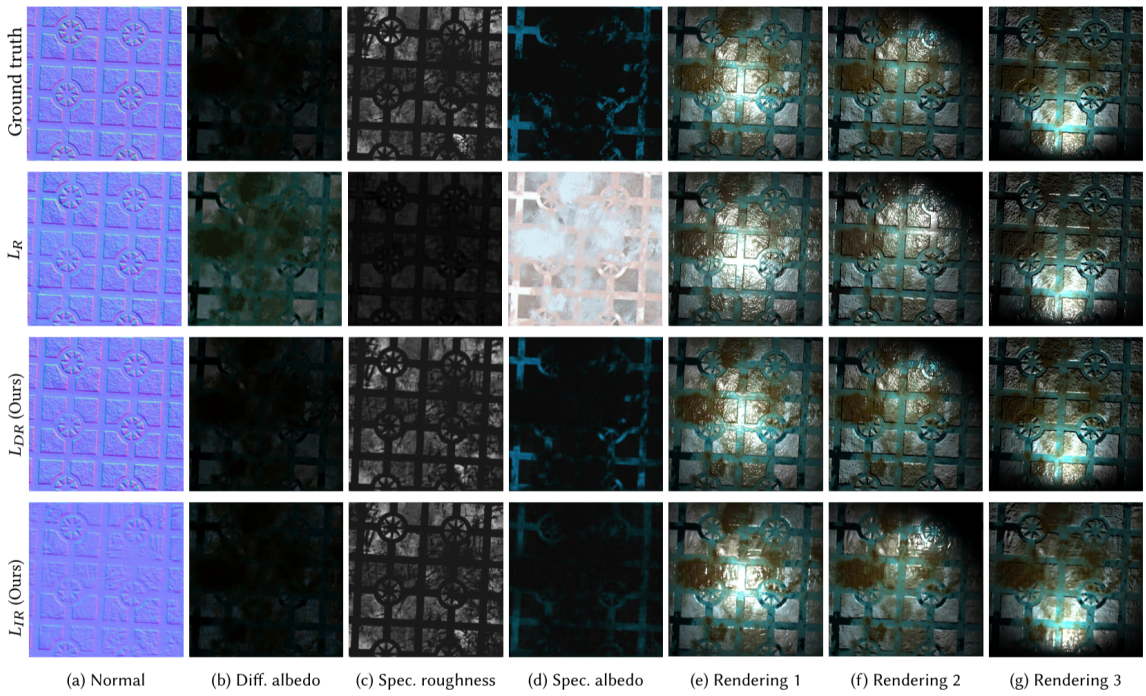

Figure 28: We overfit the model using the 3 different losses on renderings generated from the single property maps shown in the ground truth row. Both the disentangled and integrated rendering loss predict maps extremely close to ground truth, while traditional rendering loss predicts incorrect maps due to entanglement.

most error. The predictions from both L_{IR} and L_{DR} show far more accurate recovery of SVBRDF maps. This can be credited to the fact that when optimizing these new losses, the search would consistently move in a direction that would improve both the individual maps and their renderings.

4.5.3 Map Recovery

To reiterate, SVBRDF property maps recovered with the earlier defined rendering loss are very different from ground truth because the focus is on creating similar renders to the input images, without any regard to the accuracy of individual maps. Fig. 29 shows some examples wherein using rendering loss recovers inaccurate maps, whereas training with disentangled render loss or integrated loss recovers more accurate maps.

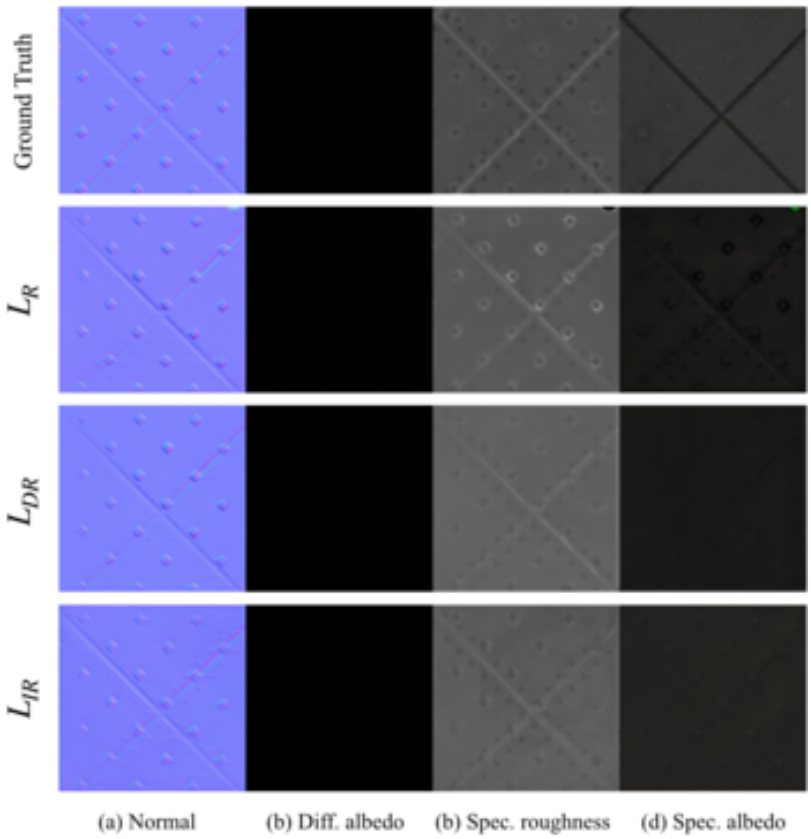

Figure 29: Example of material property map recovery of models trained with different losses.

4.6 Additional Notes

4.6.1 Network Architecture

The primary goal of our experiments is to show that changing the loss function to our disentangled rendering loss enables us to recover more accurate material property maps. Hence, we wish to emphasize that we have not deviated from state of the art work in terms of architecture, training/test data, and training cycles.

To evaluate our disentangled rendering loss, we adopt the state-of-the-art multi-image SVBRDF recovery network proposed by [23]. We use the popular U-Net encoder-decoder architecture [100] in parallel to a fully-connected track which transmits global information in the network, shown in Figure 30. This network then outputs 64 channels of feature maps for each input image view with the same spatial dimensions as the input. We then aggregate these feature maps by using max pooling so that we will have 64 channels of features of the same spatial dimensions as the input. As is the case in [23], we use the max-pooling operator which enables our

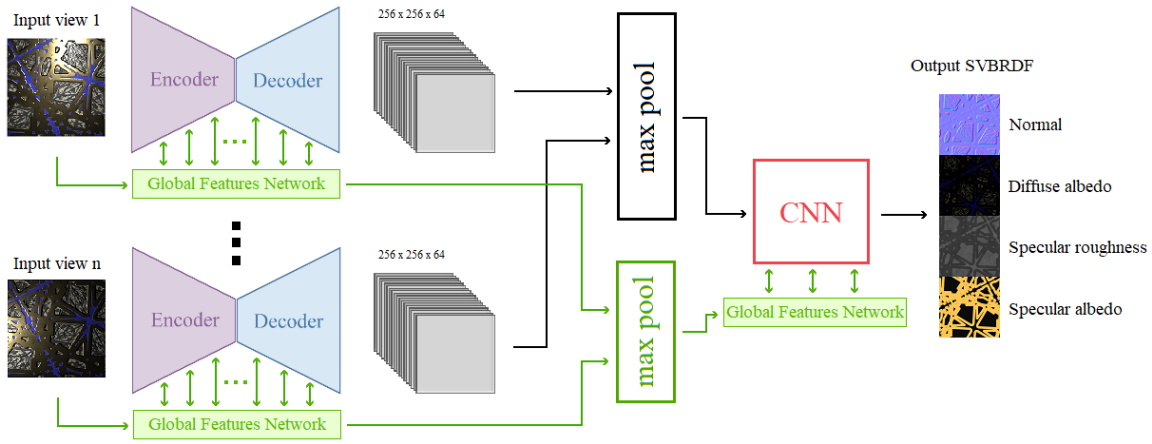

Figure 30: Network architecture.

model to handle any arbitrary number of views as inputs. Finally, the features are fed into 3 layers of convolutions with non-linearities to output the 4 material property maps.

4.6.2 Implementation Details

Training. We train our model for 300K iterations using the Adam optimizer [101] with a learning rate of 2×10^{-5}. We use a batch size of 2 and the number of views for each input sample during training is randomly chosen between 1 and 5. Training took 3 days on an Nvidia GTX 1080 Ti.

Dataset. We use the publicly available dataset[1] proposed by Deschaintre *et al.* [23]. This dataset contains 1,850 property maps of common material types such as wood, metal, leather, plastic, etc. During training, input property maps are rendered in Tensorflow [102] with a randomly chosen light and view direction, and then fed to the network.

Data augmentation. We use data augmentation to make our trained network more generalized. We use the same randomized linear interpolation of material property maps as done by Deschaintre *et al.* [23], which was shown to greatly improve accuracy.

[1] `https://repo-sam.inria.fr/fungraph/multi_image_materials/supplemental_multi_images/materialsData_multi_image.zip`

4.6.3 Integrated Loss

Rendering Equation. The rendering equation is composed of a specular term (f_r) and a diffuse term (f_d):

$$R_{N,D,R,S}(\vec{l},\vec{v}) = f_r(\vec{N},S,R,\vec{l},\vec{v}) + f_d(D) \tag{7}$$

Where $R_{N,D,R,S}(\vec{l},\vec{v})$ is the rendering function under some light direction $\vec{l}$ and view direction $\vec{v}$ parameterized by the 4 material maps N, D, R and S which are the normal, diffuse albedo, specular roughness and specular albedo maps respectively. The Cook-Torrance microfacet specular BRDF is expressed as:

$$f_r(\vec{N},S,R,\vec{l},\vec{v},\vec{h}) = \frac{F(S,\vec{v},\vec{h})G(\vec{N},R,\vec{v},\vec{l})D(\vec{N},R,\vec{h})}{4(\vec{N}\cdot\vec{l})(\vec{N}\cdot\vec{v})} \tag{8}$$

Where $\vec{h}$ is the half vector, $F(S,\vec{v},\vec{h})$ is the Fresnel function, $G(\vec{N},R,\vec{v},\vec{l})$ is the geometric shadowing term, and $D(\vec{N},R,\vec{h})$ is the Normal Distribution Function (NDF). For the Fresnel function F, we use an approximation by Schlick [103]:

$$F(S,\vec{v},\vec{h}) = S + (1-S)2^{-5.5(\vec{v}\cdot\vec{h})^2 - 6.98(\vec{v}\cdot\vec{h})} \tag{9}$$

For the geometric shadowing term G, we use Smith's method [104] which breaks G into light and view components, and uses the same G_l function for both:

$$G(\vec{l},\vec{v}) = G_l(\vec{l})G_l(\vec{v}) \tag{10}$$

We use the Schlick-Beckmann approximation for G_l [63, 103]:

$$\begin{aligned} G(\vec{N},R,\vec{l},\vec{v}) = &\frac{\vec{N}\cdot\vec{l}}{(\vec{N}\cdot\vec{l})(1-0.5R^2)+0.5R^2} \\ &\times \frac{\vec{N}\cdot\vec{v}}{(\vec{N}\cdot\vec{v})(1-0.5R^2)+0.5R^2} \end{aligned} \tag{11}$$

For the NDF term D, we use Trowbridge-Reitz GGX [63]:

$$D(\vec{N},R,\vec{h}) = \frac{R^4}{\pi\left[(\vec{N}\cdot\vec{h})^2(R^4-1)+1\right]^2} \tag{12}$$

For the diffuse term, we assume a uniform diffuse response over the microfacets hemisphere and use a simple Lambertian model:

$$f_d(D) = \frac{D}{\pi} \tag{13}$$

Putting the above formulations together, our final rendering equation is:

$$\begin{aligned} R_{N,D,R,S}(\vec{l},\vec{v}) &= f_d(D) + f_r(\vec{N},S,R,\vec{l},\vec{v},\vec{h}) \\ &= \frac{D}{\pi} \\ &+ 0.25\Bigg[\left(S + (1-S)2^{-5.5(\vec{v}\cdot\vec{h})^2 - 6.98(\vec{v}\cdot\vec{h})}\right) \\ &\quad \times \frac{1}{(\vec{N}\cdot\vec{l})(1-0.5R^2)+0.5R^2} \\ &\quad \times \frac{1}{(\vec{N}\cdot\vec{v})(1-0.5R^2)+0.5R^2} \\ &\quad \times \frac{R^4}{\pi\left[(\vec{N}\cdot\vec{h})^2(R^4-1)+1\right]^2}\Bigg] \end{aligned} \tag{14}$$

Solving the Integral. We start by making the simplifying assumption that our light and view direction are the same for our renderings ($\vec{l} = \vec{v} = \vec{h}$). By creating a new variable $t = \vec{N}\cdot\vec{v}$, the simplified rendering equation will be:

$$\begin{aligned} R_{N,D,R,S}(t) \approx \frac{D}{\pi} + \frac{0.25S}{\pi}\Bigg[&\frac{1}{\left(t(1-0.5R^2)+0.5R^2\right)^2} \\ &\times \frac{R^4}{\left(t^2(R^4-1)+1\right)^2}\Bigg] \end{aligned} \tag{15}$$

The optimization goal is to minimize the L1 error between ground truth renderings ($R_{N,D,R,S}(t)$) and prediction renderings ($R_{\hat{N},\hat{D},\hat{R},\hat{S}}(\hat{t})$) over a variety of light and view directions. If we sample infinite light and view directions, we are effectively looking

to compute $\mathbb{E}_{t,\hat{t}}[L_{DR}(t,\hat{t})]$:

$$\begin{aligned} L_{IR} =& \mathbb{E}_{t,\hat{t}}[L_{DR}(t,\hat{t})] \\ =& \iint |R_{N,\hat{D},\hat{R},\hat{S}}(t) - R_{\hat{N},\hat{D},\hat{R},\hat{S}}(\hat{t})| f(t,\hat{t}) dt d\hat{t} \\ &+ |R_{\hat{N},D,\hat{R},\hat{S}} - R_{\hat{N},\hat{D},\hat{R},\hat{S}}| \\ &+ \int |R_{\hat{N},\hat{D},R,\hat{S}}(\hat{t}) - R_{\hat{N},\hat{D},\hat{R},\hat{S}}(\hat{t})| f(\hat{t}) d\hat{t} \\ &+ \int |R_{\hat{N},\hat{D},\hat{R},S}(\hat{t}) - R_{\hat{N},\hat{D},\hat{R},\hat{S}}(\hat{t})| f(\hat{t}) d\hat{t}. \end{aligned} \tag{16}$$

We assume the distribution of the view direction such that we have the marginal probability density functions $\hat{t} \sim U(0,1)$. This assumes that our views are being sampled from directions which have a positive dot product with the ground truth normal. Since computing the expectation on the error of the normal map is not straight forward, we use an L1 loss on the normal map instead:

$$\begin{aligned} L_{IR} =& |N - \hat{N}| \\ &+ \frac{|D - \hat{D}|}{\pi} \\ &+ \int_0^1 |R_{\hat{N},\hat{D},R,\hat{S}}(\hat{t}) - R_{\hat{N},\hat{D},\hat{R},\hat{S}}(\hat{t})| d\hat{t} \\ &+ \int_0^1 |R_{\hat{N},\hat{D},\hat{R},S}(\hat{t}) - R_{\hat{N},\hat{D},\hat{R},\hat{S}}(\hat{t})| d\hat{t} \end{aligned} \tag{17}$$

To simplify the integration of the roughness and specular error terms in Eq. (17), we take the error on the log of each of these terms instead. This will not change the optimal solution that will minimize this loss:

$$\begin{aligned} &\int_0^1 \left| log(\frac{\hat{A}}{\left(Bt+1\right)^2 \left(Ct^2+1\right)^2}) \right. \\ &\qquad \left. - log(\frac{\hat{A}}{\left(\hat{B}t+1\right)^2 \left(\hat{C}t^2+1\right)^2}) \right| dt \\ &+ \int_0^1 \left| log(\frac{A}{\left(\hat{B}t+1\right)^2 \left(\hat{C}t^2+1\right)^2}) \right. \\ &\qquad \left. - log(\frac{\hat{A}}{\left(\hat{B}t+1\right)^2 \left(\hat{C}t^2+1\right)^2}) \right| dt \end{aligned} \tag{18}$$

where:

$$\begin{cases} A = \dfrac{0.25SR^4}{\pi(0.5R^2)^2} = \dfrac{S}{\pi}, & \hat{A} = \dfrac{\hat{S}}{\pi} \\ B = \dfrac{1-0.5R^2}{0.5R^2} = \dfrac{2}{R^2} - 1, & \hat{B} = \dfrac{2}{\hat{R}^2} - 1 \\ C = R^4 - 1, & \hat{C} = \hat{R}^4 - 1 \end{cases} \tag{19}$$

We simplify $log(1+x)$ to x in order to get a much simpler solution to the integral. Moreover, since $\lim_{x\to\infty} \frac{\partial log(x+1)}{\partial x} = 0$, for large values of x we will have a gradient vanishing problem, which would not be the case when simplifying $log(1+x)$ to x. Thus, Eq. (18) will be reduced to:

$$\begin{aligned} &2\int_0^1 \left|(\frac{1}{\hat{R}^2} - \frac{1}{R^2})t + (\hat{R}^4 - R^4)t^2\right| dt \\ &+ |log(S) - log(\hat{S})| \end{aligned} \tag{20}$$

To make the implementation of the solution simpler, we use the upper bound of the error on Eq. (20):

$$2|\frac{1}{\hat{R}^2} - \frac{1}{R^2}| + \frac{2}{3}|\hat{R}^4 - R^4| + |log(S) - log(\hat{S})| \tag{21}$$

Thus the upper bound on the integrated rendering loss L_{IR} would be:

$$\begin{aligned} L_{IR} =& |N - \hat{N}| + \frac{|D - \hat{D}|}{\pi} + 2|\frac{1}{\hat{R}^2} - \frac{1}{R^2}| \\ &+ \frac{2}{3}|\hat{R}^4 - R^4| + |log(S) - log(\hat{S})| \end{aligned} \tag{22}$$

4.7 Conclusions

In this work we have addressed the problem of recovering more accurate, disentangled material property maps from images. We define two versions of a new loss function or training SVBRDF estimation neural networks: The *disentangled rendering loss* and the *integrated rendering loss*. By separating out the rendering of maps and analytically integrating the specular albedo term of the rendering equation, we are able to recover more accurate SVBRDF maps than before. Our solutions are unique and require less computational resources while still producing better results than previous work without any network modifications.

Through intentional overfitting of the same model with each of the different losses,

we show property entanglement and inaccuracy in SVBRDF predictions when using traditional rendering loss, emphasizing the need for a change of loss formulations in SVBRDF recovery. However, more can be done to improve predictions further, such as exploring other network architectures, implementing the use of appropriate priors, and to increase generalization capability of the model through further data augmentation.

Broader impact. While the work presented is specific to material properties, such entanglement of component parameters would be present in other areas of deep learning research focused on recovering many parameters at once. Transferring our strategy of defining a disentangled loss function by selectively learning these parameters could potentially be transferred to these problems. Thus, the broader impact of this work can be stated as follows:

1. Potential for this methodology of defining a disentangled loss function to be applied to analogous problems.

2. Potential for this methodology of computing the expectation of a stochastic loss function with respect to some external parameters, as opposed to Monte Carlo sampling those parameters to be applied to analogous problems.

3. More accurate material property recovery will result in more correct results for downstream applications like material matching, SVBRDF editing, and AR/VR environments.

Chapter 5

Scan2Material

5.1 Introduction

The gold standard for 3D reconstruction is to be able to take a hand-held camera, casually capture the scene by walking around in an environment and recover the complete 3D geometry and material appearance properties of the environment. Recovering the 3D geometry and appearance parameters will allow for re-rendering of the scene with arbitrary light and view conditions. This has proven a difficult goal to aim for over the years. We aim to address this by describing a method for simultaneously recovering the geometry and the material appearance parameters of a scene from a casually captured sequence of RGB-D images.

The material appearance properties can be modeled through the Bidirectional Reflectance Distribution Function (BRDF). The BRDF takes as input the viewing direction and an incident light, and outputs the amount of light that gets emitted by the surface in the viewing direction. While the BRDF is a function of light and view directions, different materials are parameterized by values that depend purely on an object's material, independent of the lighting and view conditions. In this work we focus on recovering the Spatially Varying BRDF (SVBRDF) since real-world scenes consist of multiple different materials, sometimes even within a single object.

The big challenge of SVBRDF recovery from a casual capture is that there will be too many highly entangled unknowns to recover (i.e., geometry, diffuse albedo, specular albedo, etc.). One approach that has been successful for dealing with this inherent lack of information is to use learning-based methods for implicit reasoning about the unknowns in the capture [22, 23, 105]. However, this has mostly been attempted under highly restrictive assumptions on the geometry [22, 23, 105–109], lighting conditions [47, 106, 108, 110], viewing conditions [46, 48], and the BRDFs

[47, 50, 111–113]. Other works which do not have these strong restrictive assumptions typically rely on a non-generalized iterative optimization process for every capture [52–54]. Moreover, most of the previous works focus on recovering the SVBRDF of a single object since a lot of their assumptions do not hold for a complex indoor environment. Further in indoor environments, there are extra factors such as inter reflections which make the problem even more difficult.

From what we have seen in the literature, the proposed model in our work is the first to attempt to recover scene geometry and material appearance properties from casual RGB-D captures without an iterative optimization process. We do this by leveraging the power of deep neural networks and a differentiable geometry-based method of combining information from multiple views.

Our approach takes as input a sequence of casually captured RGB-D images, and extracts embedding maps from each frame individually by using a convolutional neural network (CNN). However, to observe the different reflective properties of each material in the scene, it is important to aggregate information across multiple frames. E.g., from a single viewpoint, one would not be able to observe whether the object is shiny or has a bright diffuse albedo color without prior knowledge of the material type. Our key insight is to aggregate the information across a sequence of frames geometrically by finding which points across different frames correspond to the same point in 3D space. We then use Point-Voxel CNN [5] to convert these embedding vectors of the pointcloud into material properties for every point in the pointcloud. Thus, we train the 2D embedding generator and the 3D material property predictor jointly in an end-to-end fashion once and have it generalize to a variety of different real world scenes.

Importantly, our 2D CNN and the 3D Point-Voxel CNN [5] consider local and global information when predicting the BRDF for every point. This is important for the model to be able to implicitly reason about the (direct and indirect) lighting information of the scene. Briefly, our contributions are as follows:

1. The first described generalized approach for simultaneous SVBRDF and geometry recovery of casually captured scenes.

2. A novel approach for leveraging 2D multiview information when making predictions for each point's BRDF.

3. A novel loss function (*reflection loss*) for non-flat surfaces incorporating the effect of each material property on the final rendering and their dependence on one another.

5.2 Related Work

While there is a large body of work which focuses on recovering uniform BRDFs for a single material [111–113], we focus our discussion on methods of SVBRDF recovery as they are more closely related to our work.

Non-learning based methods. Traditional works on recovering SVBRDFs focus on capturing appearance under controlled light and view conditions [114–118]. While these works yield the most accurate SVBRDFs, they require dedicated hardware and strict acquisition setups which is often only suited for laboratory conditions.

Many works have focused on recovering SVBRDFs without the use of dedicated hardware. One approach is to assume known geometry of the captured object to simplify the SVBRDF recovery problem, such as assuming the capture to be of a near-flat surface [106,108] or having a pre-scanned object with a 3D scanner [107,109]. However, the obvious limitation of these approaches is that they will not work when the geometry of the object is unknown.

More closely related to our approach are ones that aim to recover the geometry and SVBRDF parameters simultaneously. One approach is to use Multi-View Stereo (MVS) for reconstruction of the object's geometry [110, 119]. However, MVS-based reconstructions are ineffective for challenging scenes with textureless regions. Xia *et al.* [120] recover the geometry by searching for the closed surface based on the recovered normal directions. However, they require the input to be the video capture of a single rotating object.

Recent works have been looking into using RGB-D sensors to simultaneously recover geometry and SVBRDF [46–50]. Wu *et al.* [46] and Ha *et al.* [48] use RGB-D inputs from a Kinect sensor to simultaneously recover the SVBRDF and geometry. However, they either use a mirror ball [46] or a 360-degree camera [48] at capture time for measuring environment illumination, whereas our proposed approach relies solely on the RGB-D input sequence under unknown illumination conditions. Zhang *et al.* [50] use RGB-D inputs from a Kinect sensor to recover the SVBRDF and geometry with some manual adjustments. Wu *et al.* [47] use RGB-D inputs from a Kinect sensor to recover the SVBRDF and geometry in a joint optimization framework, which alternates between solving for materials, environment illumination, camera poses and normals. They further make an assumption of distant environment lighting, which is typically not valid in indoor captures. However, both aforementioned methods simplify the domain of recovered SVBRDFs by relying on clustering materials and

per-segment BRDF association. This reduces the quality of results obtained when the scene/object contains a large number of different BRDFs. In contrast, we aim to recover continuous SVBRDF parameters separately for every vertex in the scene.

Most related to our approach here is the work by Schmitt *et al.* [49], who jointly recover the SVBRDF and geometry from a sequence of RGB-D images. However, their approach assumes a single object present in the scene with AprilTags [121] next to it and captured with exactly one point light source. In contrast, our approach was built for recovering the SVBRDF and 3D geometry of an entire multi-object scene, which was casually captured under unknown lighting conditions.

One thing to note is that with the exception of the work of Zhang *et al.* [50], all above approaches focus on recovering SVBRDF for a single object, not an entire scene. Moreover, even Zhang *et al.* [50] rely on manual adjustments and restrict BRDF properties to only the floor, ceiling, and walls. The standard approach for these methods is to refine the recovered SVBRDF and possibly the geometry and/or illumination [120] parameters for every captured object via iterative optimization. Doing this iterative inverse-rendering optimization process for each RGB-D sequence can be time consuming. In contrast, our approach trains a single neural network that generalizes to a variety of different RGB-D captured scenes. Moreover, these approaches make strong assumptions on geometry [106–109], lighting [47, 106, 108, 110], the capture setup [46, 48], or recovered materials [47, 50]. Since our method is learning based, we do not need to explicitly recover lighting conditions for recovering the SVBRDF. We can allow the network to implicitly reason about the illumination of the scene for recovering the SVBRDF.

Learning based methods. Recently there has been an increasing amount of interest in SVBRDF estimation from casual captures using learning-based methods. Due to the difficulty in recovering accurate SVBRDFs in a single forward step prediction, a large body of work combines learning based methods with the standard iterative refinement methods [26,45,122–125]. These approaches make a prediction for the SVBRDF first and then iteratively refine the latent representation based on the re-rendering error. These works are based on either having a single input image [122,125], or having multi-view images [26, 45, 124]. However, doing this iterative optimization process at inference time can be slow.

An interesting recently emerging line of work consists of approaches that take inspiration from Neural Radiance Fields (NeRF) [51–54]. NeRF based approaches for recovering the SVBRDF rely on fitting a separate neural network for every captured scene to recover the per-pixel BRDF parameters [52–54]. In contrast, our goal is to

train a neural network once, and have it generalize to a variety of different scenes.

Most related to our work are learning-based methods that aim to generalize to a variety of scenes. One class of works in this domain attempt to recover the SVBRDF from a single image [22, 126, 127]. While recovering SVBRDF from a single-view image is an inherently ill-posed problem as many different SVBRDFs can yield the same observation from a single view [55], these works hope that the model learns to predict the most likely SVBRDF based on learned priors. Thus, the network must make a best guess for the SVBRDF based on ambiguous limited information. Multi-image approaches for SVBRDF recovery aim to alleviate this [23, 56, 105]. However, all of the aforementioned generalized single image and multi-image methods assume a flat geometry for the captured object. As mentioned earlier, we believe that there are currently no generalized approaches for recovering the SVBRDF of a capture with non-flat geometry. Moreover, our work recovers the SVBRDF of a capture of a complex scene with multiple objects. We do this with much lesser assumptions on the scene (e.g., no assumptions on lighting and geometry).

5.3 Method

Overview. Learning representations from 2D images using convolutional neural networks is very efficient and effective. However, predicting the SVBRDF from a single 2D image of a complex indoor scene would require the network to rely heavily on learned priors and guessing the material appearance parameters. Therefore, aggregating the visual cues across different views can help in estimating more accurate SVBRDF parameters. We propose to extract an embedding map (i.e., per-pixel embedding vectors) for every frame and aggregate the information across a sequence of frames geometrically, by reconstructing a pointcloud and finding the points across different frames which correspond to the same point in 3D space (i.e., by using ∇PointFusion [9]). This allows us to aggregate information about a single point in space when viewed from different angles. We then use a deep network that operates on the entire pointcloud (Point-Voxel CNN [5]) to generate per-point BRDF parameters, allowing the model to implicitly reason about the structure and the global illumination of the entire scene.

Embedding vectors. The first step in our approach is to compute the vertex map in camera space for each RGB-D frame:

$$V_t(\mathbf{u}) = D_t(\mathbf{u})\mathbf{K}^{-1}[\mathbf{u}^T, 1]^T \in \mathbb{R}^3$$

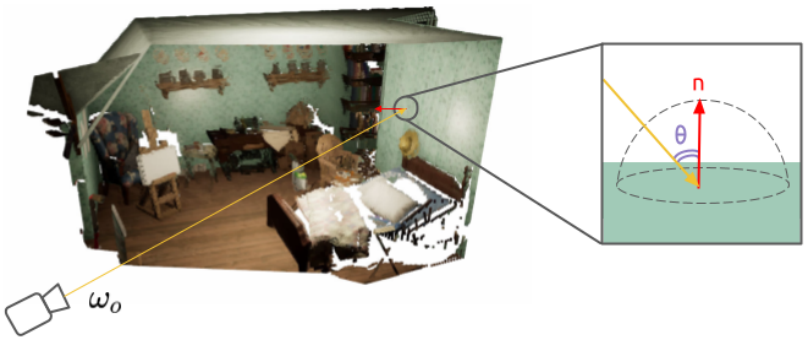

Figure 31: The viewing zenith angle (θ) is the angle between the viewing direction ω_o and the surface normal direction n.

where V_t and D_t are the vertex map and depth map of the frame at time step t respectively, $\mathbf{u} = [x, y]^T \in \mathbb{R}^2$ is the 2D pixel position, and $\mathbf{K}$ is the camera calibration intrinsics matrix. We are then going to estimate the surface normals N_t oriented towards the camera in the camera space for each frame by using the central-differences of the vertex map. Then, using the normals N_t and camera intrinsics matrix $\mathbf{K}$, we compute the cosine of the viewing zenith angle $\theta_t(\mathbf{u})$ for every pixel of frame t:

$$\cos(\theta_t(\mathbf{u})) = N_t(\mathbf{u}) \cdot \frac{-\mathbf{K}^{-1}[\mathbf{u}^T, 1]^T}{||\mathbf{K}^{-1}[\mathbf{u}^T, 1]^T||_2}$$

The cosine of the viewing zenith angle captures information about the viewing angle of the surface point corresponding to every pixel (see Fig. 31), which is important when estimating BRDF parameters as the BRDF is a function of the viewing direction. Since we are assuming an isotropic Cook-Torrance BRDF model (i.e., the reflected light is invariant to rotation of the surface about the surface normal), the viewing azimuth angle can be ignored. Thus, we concatenate the per-pixel cosine of the viewing zenith angle $\cos(\theta_t(\mathbf{u}))$ with the RGB-D input into a single tensor, and then feed each tensor independently to a U-Net architecture [128] to generate embedding maps Z_t for each frame t.

3D reconstruction. Next, we use ∇PointFusion [9] to merge each frame and the associated embedding map into the global map (pointcloud) in a fully differentiable manner. Note that in the merging step, pixels in different frames which correspond to the same point in the original 3D scene can be fused into a single point (see Fig. 32). We adopt the same mechanism as ∇PointFusion [9] for finding correspondences between the live frame and the global map points. If a corresponding global map point is found for a point in the live frame, we use the element-wise max operator for

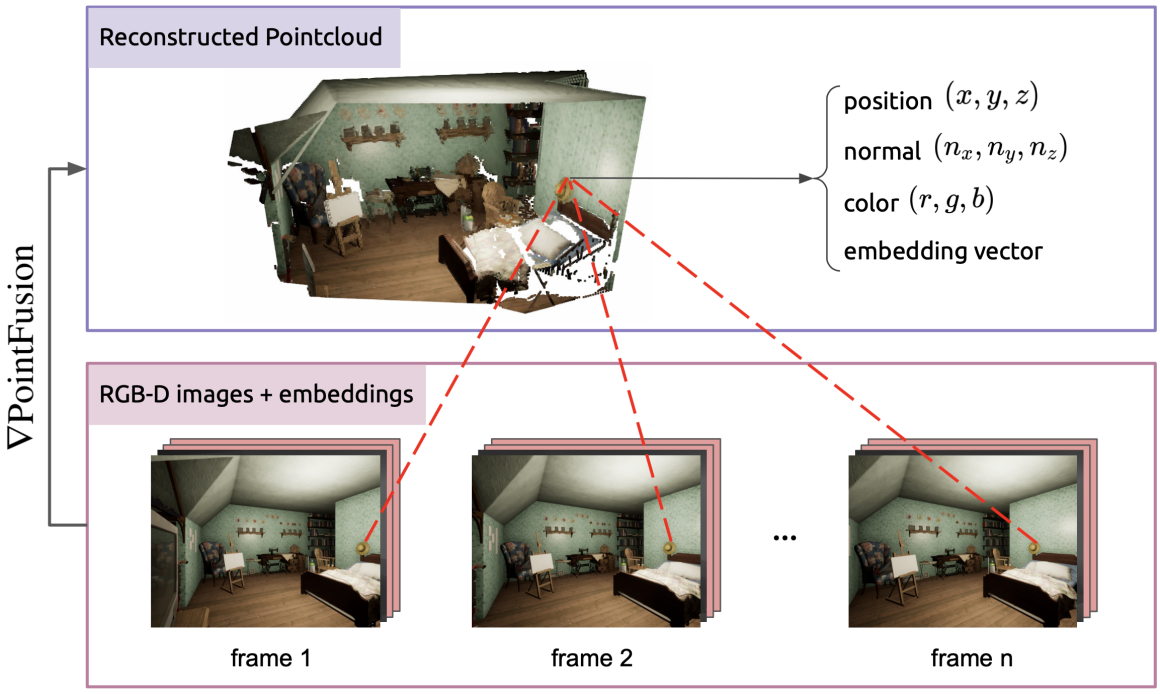

Figure 32: Embedding vector fusion. Embedding vectors of corresponding points get fused together using an element-wise max operation.

fusing the embedding vectors as follows:

$$\bar{\mathbf{z}}_i \leftarrow \max\left(\bar{\mathbf{z}}_i, Z_t(\mathbf{u})_i\right)$$

where $Z_t(\mathbf{u}) \in \mathbb{R}^m$ denotes the embedding vector of pixel $\mathbf{u}$ in frame t, $\bar{\mathbf{z}} \in \mathbb{R}^m$ is the embedding vector of the corresponding global map point, and i is the index of the embedding vector element. Since the ordering of images is not meaningful in recovering the SVBRDF parameters, the max operator ensures an order-invariant way of fusing embedding vectors from different views.

BRDF predictions. Finally, we feed the pointcloud vertices along with the associated normals and embedding vectors to the Point-Voxel CNN [5] to predict the BRDF parameters of each point. We jointly train this Point-Voxel CNN [5] and the embedding generator U-Net [128] by having a supervised loss on the predicted BRDF parameters. A high-level overview of our approach is shown in Fig. 33.

Reflection Loss. In our dataset, the input images have ground truth per-pixel BRDF parameters to be used for supervision. Thus, we associate the ground truth BRDF parameters with each point in the pointcloud as we are reconstructing the pointcloud by ∇PointFusion [9]. If a point in the live frame is fused with a point in the global map, we apply the vertex fusion mechanism of ∇PointFusion [9] for fusing

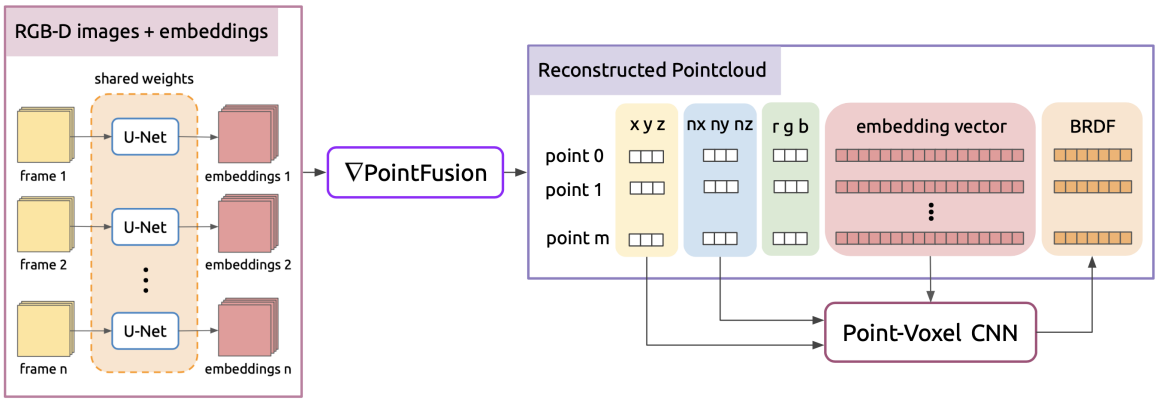

Figure 33: Scan2Material inference pipeline overview.

their associated BRDF parameters. This saves us storage, as we would not need to store the ground truth pointcloud for every scene. And we also save on computation, as we would not need an extra step for finding point associations between ground truth and predicted pointclouds.

Having the association with the correct BRDF parameters for every point, we define a *reflection loss* as our loss function. For each point in the pointcloud, the reflection loss samples light and view directions and computes the L_1 distance between the rgb reflection using the ground truth and predicted SVBRDF:

$$\text{reflection loss} := \sum_{1 \leq n \leq m} |\hat{f}_\text{r}(\mathbf{v}_n, \omega_\text{i}, \omega_\text{o}) - f_\text{r}(\mathbf{v}_n, \omega_\text{i}, \omega_\text{o})|$$

where $\hat{f}_\text{r}$ is the predicted SVBRDF, f_r is the ground truth SVBRDF, m is the total number of points in the pointcloud, $\mathbf{v}_n$ is the n'th vertex of the reconstructed pointcloud, ω_i is the incoming light direction, and ω_o is the outgoing view direction. The light and view directions are heuristics, and they can be sampled uniformly over the hemisphere. To reduce the variance in the reflection loss (due to the stochasticity involved in sampling light and view directions), one can take the average of this loss over multiple light and view directions. Moreover, we can extend this idea to a *disentangled reflection loss* or an *integrated reflection loss* by adopting the same disentangled loss equations as Saryazdi *et al.* [56] for every point in the pointcloud. Finally, the gradient of this error can be backpropagated to the Point-Voxel CNN [5] parameters, and also through Point-Voxel CNN [5] and ∇PointFusion [9] to U-Net [128] parameters.

The reflection loss has several important benefits. First, the reflection loss is very efficient, as it operates on each point independently and thus is trivial to parallelize.

Second, the reflection loss takes into account how much each BRDF parameter contributes to the appearance of that point. E.g., assuming two different sets of BRDF parameters yield the same L_1 loss, using the reflection loss would yield a lower loss for the BRDF that gives a more similar looking appearance to that of the ground truth under different light and view directions. The limitation of the reflection loss is that it requires knowledge of the ground truth SVBRDF parameters, thus it is only applicable to supervised training scenarios where the ground truth SVBRDFs are known.

5.4 Dataset

We could not find any publicly available dataset which contains image sequences of complex scenes with corresponding per-pixel BRDF parameters[1]. Generating such a dataset from real-world images is currently close to impossible. However, generating it from synthetic 3D models is feasible. It is important for these 3D models and their renderings to look as realistic as possible to reduce the domain gap with real-world images. Luckily, the Hypersim [1] dataset has already generated and released 77,400 photorealistic rendered images of 461 indoor scenes by using artist designed 3D models in their massive $\sim$ 1.9 Tb dataset. Some of the biggest benefits of Hypersim [1] are the high resolution (1024×768) and high-quality realistic renderings, the availability of many different types of annotations, and very important, the entire code for generating this dataset is open sourced. However, Hypersim [1] does not contain per-pixel BRDF information.

For achieving our goal, we extend the existing Hypersim [1] dataset with additional SVBRDF parameters. For each scene, Hypersim [1] has one or more camera trajectories, each of which capture 100 photorealistic images, in addition to other per-pixel labels such as semantic segmentation information, depth, normals direction, etc. The good news is that the 3D models[2] that Hypersim [1] was created from do have very realistic SVBRDF values in them, and so we extract the per-pixel SVBRDF values for the images that Hypersim [1] has generated using the original 3D models.

We use V-Ray [132] for rendering the per-pixel BRDF parameters. While V-Ray [132] allows for rendering of per-pixel BRDF parameters such as diffuse albedo, reflection albedo and reflection glossiness, there is some useful information that can

[1]The OpenRooms [129] dataset would meet our requirements, but unfortunately it is not publicly available at the time of writing this.

[2]The Evermotion Archinteriors Collection [130] is available for purchase on TurboSquid [131].

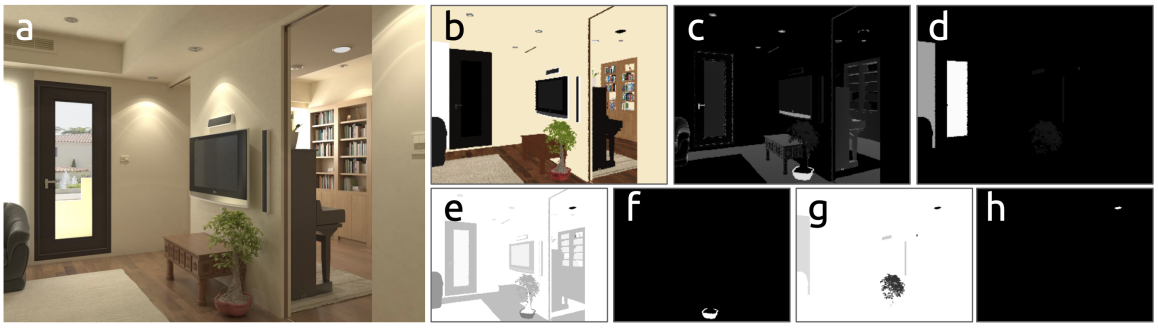

Figure 34: Overview of our dataset. For each rendered image in Hypersim [1] (a), our dataset adds the following ground truth values: Diffuse albedo (b); reflection albedo (c); refraction albedo (d); reflection glossiness (e); reflection ior (f); refraction glossiness (g); and a binary mask indicating whether each pixel is a light source or not (h).

not be directly rendered by V-Ray [132], such as knowing the pixels which contain a light source. Thus, we first create a correspondence image that associates each pixel in the rendered image with its associated material in the 3D model, and then we can parse extra information from the 3D assets by a simple lookup. The final labels that our dataset adds to Hypersim [1] are shown in Fig. 34. We plan to release this dataset publicly to facilitate research in this domain.

5.5 Experiments

At the time of writing this thesis, we were unable to carry out exhaustive training of our model, as the task of extending the Hypersim data with BRDF values is continuing. However, for testing out our network's capabilities, we did some initial experiments as described below.

Overfitting experiment. To demonstrate that our network has the representation capacity for learning the SVBRDF parameters, we overfit the proposed model on a single sequence of synthetic RGB-D data and visualize the results in Fig. 35. The training was done using purely 3D supervision as our loss function operates on the predicted SVBRDF parameters in the reconstructed pointcloud. In particular, we feed in a sequence of 4 RGB-D images to our model, and optimize our model using Adam [101] optimizer for 100 iterations.

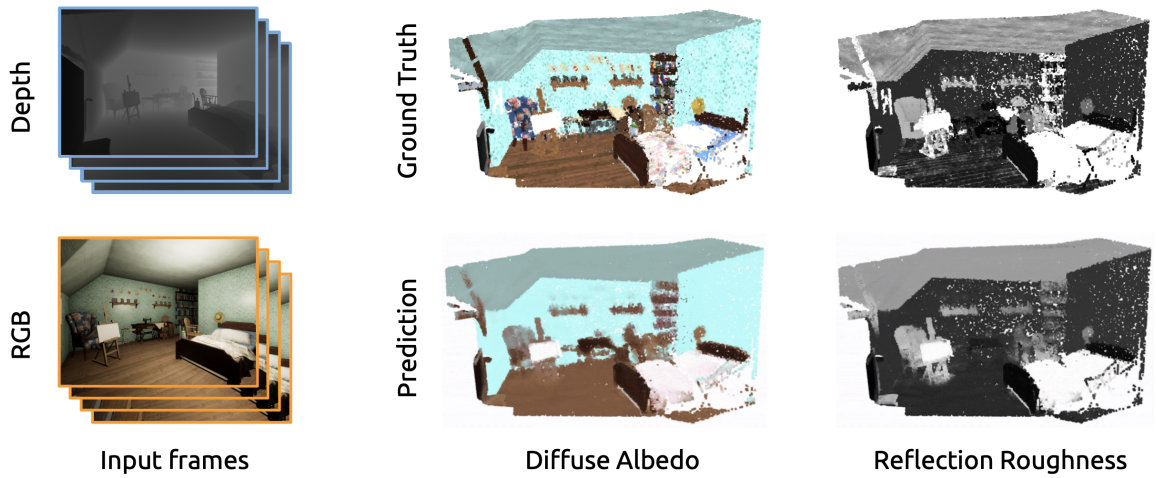

Figure 35: Overfitting experiment results. Scan2Material trained for 100 iterations on a single sequence.

5.6 Conclusions and Future Work

Conclusions. In this chapter, we have described work towards an end-to-end trainable model that can simultaneously reconstruct the 3D geometry and predict the SVBRDF of a complex scene. We showed that this model can be trained end-to-end using only direct 3D supervision. We also proposed a *reflection loss* that considers the effect of each BRDF parameter in the final appearance of that point. Lastly, since there is currently no public dataset available for supervised training of this model, we have embarked on the creation of a dataset of SVBRDF parameters for the complex indoor scenes of Hypersim [1].

Future work. In the short term, we want to train this model on the entire Hypersim [1] dataset with the SVBRDF parameters and evaluate the generalization capabilities of such a model. Once this is demonstrated, there are a lot of interesting ways that this line of work can be extended. One interesting line is to also recover the lighting information, which could potentially also enable unsupervised training of such a model by using differentiable renderers [122]. Another interesting extension would be to aim for recovering finer material information, such as diffuse roughness from the Oren-Nayar reflectance model [133].

Chapter 6

Conclusions

Representation learning approaches have made huge progress in solving problems that not too long ago defied our grasp, such as semantic segmentation and language translation. This has unleashed a lot of excitement in using representation learning based approaches in solving difficult tasks in other domains such as video understanding and multi-modal learning between text and images.

However, in the domain of 3D reconstruction, there has been an open question about how we want to blend these representation learning approaches with the current 3D reconstruction pipeline. In particular, there is a question about whether we want to place pre-trained models in some components of the 3D reconstruction pipeline; or replace the entire 3D reconstruction pipeline with a neural network; or find a sweet spot between the other two approaches.

In this thesis, we proposed approaches for using neural networks in conjunction with the 3D reconstruction pipeline such that they can be trained end-to-end based on a single end objective (e.g., to reconstruct an accurate 3D representation). We argue that doing so will 1) give us more interpretable models due to the modular nature of the 3D reconstruction pipeline, 2) allow us to use components from the 3D reconstruction pipeline that work well (e.g. the odometry and mapping mechanism) in conjunction with representation learning approaches for tackling tasks which have been difficult to solve by designing explicit algorithms (e.g., estimating BRDF parameters), 3) allow for end-to-end training for a single end optimization objective (such as having accurate reflective behaviors in the final 3D reconstruction), and 4) allow for using additional auxiliary objectives for each component to speed up training and avoid possible local minima. Moreover, we also discussed what the end objective function should be in order for the material appearance parameters predicted by the network to be individually accurate, and at the same time also faithfully recreate the

appearance of the material as a whole. Overall, there are a number of "firsts" in our work.

Concretely, in Chapter 3 we developed the first open-source library for differentiable dense SLAM, called `gradslam`. This was done by using the differentiable alternatives to the non-differentiable components in common SLAM pipelines, in addition to leveraging the automatic differentiation capability of PyTorch [67]. Due to having fully differentiable components, `gradslam` enables both using SLAM as a layer inside of deep neural networks for learning downstream tasks, and also using deep neural networks in subcomponents of the SLAM system to recover more accurate maps and state estimates.

In Chapter 4 we identified some of the problems of the rendering loss function which is used for training neural networks to estimate material appearance parameters from image(s) of a near-flat surface. We were the first to propose the disentangled rendering loss which alleviates many of the disadvantages of using the rendering loss for training a neural network (e.g., having multiple global minima under limited light and view samples), while maintaining the benefits (e.g., weighing the importance of each material appearance parameter based on its effect on the final appearance). The important take away of this chapter is that while some of the mathematical models for real-world phenomena are already differentiable, it is important to consider the affect of using these models in training our neural networks (e.g., on the loss landscape).

Finally, in Chapter 5 we described the first work towards an end-to-end trainable model for simultaneously reconstructing the 3D geometry and estimating the material appearance parameters of objects in a scene. One of the current limitations for training such a model is that there are currently no publicly available datasets with ground truth material appearance parameters. Thus, we extended an existing large-scale photorealistic synthetic dataset (Hypersim [1]) by generating corresponding material appearance parameters for each image. This was done by extracting the material appearance parameters from the original 3D assets. We intend to release this dataset publicly.

One of the main challenges with our end-to-end learning approach is the GPU memory limitation issue. This is in part due to the temporal dependence of the reconstruction process which restricts the input image sequence length. As an example, unrolling each computation in dense SLAM during training time as a graph requires an enormous amount of memory. Additionally, the spatial resolution of the reconstructed maps also plays a role in the GPU memory consumption. In particular, running a differentiable KinectFusion [16] algorithm using a coarse voxel

resolution $128 \times 128 \times 128$ ends up requiring 6 GB of GPU memory on average. One way to alleviate this problem is to use point-based reconstruction methods, such as ∇PointFusion. Another main challenge of our approach for material appearance parameter recovery is the requirement of having a large-scale labeled dataset. Capturing real-world datasets of complex scenes with accurate ground truth material appearance parameters is close to impossible, and generating photorealistic renderings from synthetic artist designed 3D models can be (financially and computationally) expensive.

There are still many interesting challenges on the horizon to be solved. One interesting idea is to use unsupervised training for simultaneously recovering the BRDF and geometry. One way to do this is through differentiable physically based renderers [134], by re-rendering the 3D reconstruction and comparing it to the ground truth input image. This is not only exciting due to its capability to get more accurate 3D reconstruction models, but more importantly because this can potentially enable learning rich representations for images from unlabeled data (analogous to what has happened with BERT [135] and GPT3 [65] in the natural language domain). Another interesting challenge that still remains is reconstructing complete 3D scenes that contain refractive materials, such as glass or water. In these cases, even the most expensive active depth sensors will fail to predict correct depth measurements. Therefore, using vision-based approaches for depth recovery such as monocular depth estimation might prove to be valuable in such scenarios.

The aforementioned are some of the very exciting directions for future research based on our work.

References

[1] M. Roberts and N. Paczan, "Hypersim: A photorealistic synthetic dataset for holistic indoor scene understanding." arXiv 2020. x, xi, xiii, 2, 9, 10, 15, 17, 62, 63, 64, 66

[2] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger, "Elasticfusion: Real-time dense slam and light source estimation," *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1697–1716, 2016. x, xi, 2, 3, 5, 20

[3] "Volumetric tsdf fusion of rgb-d images in python." `https://github.com/andyzeng/tsdf-fusion-python`. x, 3

[4] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015. x, 12

[5] Z. Liu, H. Tang, Y. Lin, and S. Han, "Point-voxel cnn for efficient 3d deep learning," in *Advances in Neural Information Processing Systems*, 2019. x, 12, 13, 55, 58, 60, 61

[6] "Shadows." `https://cglearn.codelight.eu/pub/computer-graphics/shadows`. x, 16

[7] T. H. M. Siddique and M. Usman, "3d object localization using 2d estimates for computer vision applications," 2020. x, 16

[8] "The phong model, introduction to the concepts of shader, reflection models and brdf." `https://www.scratchapixel.com/lessons/3d-basic-rendering/phong-shader-BRDF`. xi, 18

[9] J. Krishna Murthy, S. Saryazdi, G. Iyer, and L. Paull, "gradslam: Dense slam meets automatic differentiation," *arXiv*, 2020. xi, xii, 9, 10, 22, 23, 25, 26, 27, 28, 29, 32, 33, 34, 35, 37, 38, 58, 59, 60, 61

[10] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, 2012. xi, 31, 32, 33

[11] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "Scannet: Richly-annotated 3d reconstructions of indoor scenes," in *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017. xi, 31, 32, 34

[12] A. Dai, M. Nießner, M. Zollöfer, S. Izadi, and C. Theobalt, "Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration," *ACM Transactions on Graphics 2017 (TOG)*, 2017. xi, 34

[13] H. Zhou, B. Ummenhofer, and T. Brox, "Deeptam: Deep tracking and mapping," in *ECCV*, 2018. 2, 5, 23

[14] K. Tateno, F. Tombari, I. Laina, and N. Navab, "Cnn-slam: Real-time dense monocular slam with learned depth prediction," in *CVPR*, 2017. 2, 3, 5, 23

[15] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *ECCV*, 2014. 2, 5, 23

[16] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. W. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking.," in *ISMAR*, 2011. 2, 5, 22, 23, 25, 27, 28, 32, 66

[17] 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," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 598–626, 2015. 2, 5, 22, 28

[18] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb, "Real-time 3d reconstruction in dynamic scenes using point-based fusion," in *2013*, 2013. 2, 5, 22, 23, 25, 27, 29, 31, 32, 33

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NeurIPS*, 2012. 2, 22

[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017. 2

[21] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, B. Kingsbury, *et al.*, "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal processing magazine*, 2012. 2

[22] V. Deschaintre, M. Aittala, F. Durand, G. Drettakis, and A. Bousseau, "Single-image svbrdf capture with a rendering-aware deep network," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 4, p. 128, 2018. 2, 6, 9, 40, 41, 54, 58

[23] V. Deschaintre, M. Aittala, F. Durand, G. Drettakis, and A. Bousseau, "Flexible svbrdf capture with a multi-image deep network," in *Computer Graphics Forum*, vol. 38, pp. 1–13, Wiley Online Library, 2019. 2, 6, 7, 9, 40, 41, 45, 47, 48, 54, 58

[24] Z. Li, K. Sunkavalli, and M. Chandraker, "Materials for masses: Svbrdf acquisition with a single mobile phone image," in *Computer Vision – ECCV 2018* (V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, eds.), (Cham), pp. 74–90, Springer International Publishing, 2018. 2, 6, 9, 40, 41

[25] X. Li, Y. Dong, P. Peers, and X. Tong, "Modeling surface appearance from a single photograph using self-augmented convolutional neural networks," *ACM Trans. Graph.*, vol. 36, July 2017. 2, 6, 40

[26] M. Boss, V. Jampani, K. Kim, H. Lensch, and J. Kautz, "Two-shot spatially-varying brdf and shape estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3982–3991, 2020. 2, 57

[27] S. Yang and S. Scherer, "Cubeslam: Monocular 3-d object slam," *IEEE Transactions on Robotics*, vol. 35, pp. 925–938, Aug 2019. 2, 5, 24

[28] B. Mu, S. Liu, L. Paull, J. Leonard, and J. P. How, "Slam with objects using a nonparametric pose graph," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4602–4609, Oct 2016. 2, 5, 24

[29] P. Parkhiya, R. Khawad, J. K. Murthy, B. Bhowmick, and K. M. Krishna, "Constructing category-specific models for monocular object-slam," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1–9, May 2018. 2, 5, 24

[30] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser, "3dmatch: Learning local geometric descriptors from rgb-d reconstructions," in *CVPR*, 2017. 3

[31] J. Guo, Y. Guo, J. Pan, and W. Lu, "Brdf analysis with directional statistics and its applications," *IEEE transactions on visualization and computer graphics*, vol. PP, 10 2018. 4

[32] S. Bell, K. Bala, and N. Snavely, "Intrinsic images in the wild," *ACM Trans. on Graphics (SIGGRAPH)*, vol. 33, no. 4, 2014. 5

[33] M. Bloesch, J. Czarnowski, R. Clark, S. Leutenegger, and A. J. Davison, "Codeslam—learning a compact, optimisable representation for dense visual slam," in *CVPR*, 2018. 5, 23

[34] S. Zhi, M. Bloesch, S. Leutenegger, and A. J. Davison, "Scenecode: Monocular dense semantic reconstruction using learned encoded scene representations," in *CVPR*, 2019. 5, 23

[35] J. Czarnowski, T. Laidlow, R. Clark, and A. Davison, "Deepfactors: Real-time probabilistic dense monocular slam," in *IEEE RAL*, 2020. 5

[36] B. D. Lucas, T. Kanade, *et al.*, "An iterative image registration technique with an application to stereo vision," 1981. 6, 24

[37] C. Kerl, J. Sturm, and D. Cremers, "Robust odometry estimation for rgb-d cameras," in *ICRA*, 2013. 6, 24

[38] R. Garg, V. K. BG, G. Carneiro, and I. Reid, "Unsupervised cnn for single view depth estimation: Geometry to the rescue," in *ECCV*, 2016. 6, 24

[39] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," in *CVPR*, 2017. 6, 24

[40] R. Li, S. Wang, Z. Long, and D. Gu, "Undeepvo: Monocular visual odometry through unsupervised deep learning," in *ICRA*, 2018. 6, 24

[41] M. Jaderberg, K. Simonyan, A. Zisserman, *et al.*, "Spatial transformer networks," in *Neurips*, 2015. 6, 24

[42] A. Handa, M. Bloesch, V. Pătrăucean, S. Stent, J. McCormac, and A. Davison, "gvnn: Neural network library for geometric computer vision," in *ECCV Workshop on Geometry Meets Deep Learning*, 2016. 6, 24

[43] E. Riba, D. Mishkin, D. Ponsa, E. Rublee, and G. Bradski, "Kornia: an open source differentiable computer vision library for pytorch," in *Winter Conference on Applications of Computer Vision*, 2020. 6, 24, 30

[44] S. R. Marschner, S. H. Westin, E. P. Lafortune, K. E. Torrance, and D. P. Greenberg, "Image-based brdf measurement including human skin," in *Rendering Techniques' 99*, pp. 131–144, Springer, 1999. 6

[45] D. Gao, X. Li, Y. Dong, P. Peers, K. Xu, and X. Tong, "Deep inverse rendering for high-resolution svbrdf estimation from an arbitrary number of images," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, p. 134, 2019. 7, 40, 41, 57

[46] H. Wu and K. Zhou, "Appfusion: Interactive appearance acquisition using a kinect sensor," in *Computer Graphics Forum*, vol. 34, pp. 289–298, Wiley Online Library, 2015. 7, 54, 56, 57

[47] H. Wu, Z. Wang, and K. Zhou, "Simultaneous localization and appearance estimation with a consumer rgb-d camera," *IEEE transactions on visualization and computer graphics*, vol. 22, no. 8, pp. 2012–2023, 2015. 7, 54, 55, 56, 57

[48] H. Ha, S.-H. Baek, G. Nam, and M. H. Kim, "Progressive acquisition of svbrdf and shape in motion," in *Computer Graphics Forum*, vol. 39, pp. 480–495, Wiley Online Library, 2020. 7, 54, 56, 57

[49] C. Schmitt, S. Donne, G. Riegler, V. Koltun, and A. Geiger, "On joint estimation of pose, geometry and svbrdf from a handheld scanner," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3493–3503, 2020. 7, 56, 57

[50] E. Zhang, M. F. Cohen, and B. Curless, "Emptying, refurnishing, and relighting indoor spaces," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 6, pp. 1–14, 2016. 7, 55, 56, 57

[51] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *European Conference on Computer Vision*, pp. 405–421, Springer, 2020. 7, 57

[52] M. Boss, R. Braun, V. Jampani, J. T. Barron, C. Liu, and H. Lensch, "Nerd: Neural reflectance decomposition from image collections," *arXiv preprint arXiv:2012.03918*, 2020. 7, 55, 57

[53] S. Bi, Z. Xu, P. Srinivasan, B. Mildenhall, K. Sunkavalli, M. Hašan, Y. Hold-Geoffroy, D. Kriegman, and R. Ramamoorthi, "Neural reflectance fields for appearance acquisition," *arXiv preprint arXiv:2008.03824*, 2020. 7, 55, 57

[54] P. P. Srinivasan, B. Deng, X. Zhang, M. Tancik, B. Mildenhall, and J. T. Barron, "Nerv: Neural reflectance and visibility fields for relighting and view synthesis," *arXiv preprint arXiv:2012.03927*, 2020. 7, 55, 57

[55] S. Saryazdi, C. Murphy, and S. Mudur, "The Problem of Entangled Material Properties in SVBRDF Recovery," in *Workshop on Material Appearance Modeling* (R. Klein and H. Rushmeier, eds.), The Eurographics Association, 2020. 9, 10, 58

[56] S. Saryazdi, C. Murphy, and S. Mudur, "Disentangled rendering loss for supervised material property recovery," in *Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 1: GRAPP,*, pp. 113–121, INSTICC, SciTePress, 2021. 9, 10, 58, 61

[57] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. 11

[58] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. 13

[59] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013. 13

[60] P. Beckmann and A. Spizzichino, *The scattering of electromagnetic waves from rough surfaces*. 1987. 18

[61] K. E. Torrance and E. M. Sparrow, "Theory for off-specular reflection from roughened surfaces," *J. Opt. Soc. Am.*, vol. 57, pp. 1105–1114, Sep 1967. 18

[62] R. L. Cook and K. E. Torrance, "A reflectance model for computer graphics," *ACM Trans. Graph.*, vol. 1, p. 7–24, Jan. 1982. 18

[63] B. Walter, S. Marschner, H. Li, and K. Torrance, "Microfacet models for refraction through rough surfaces.," pp. 195–206, 01 2007. 18, 49

[64] D. Moulds, C. Guarnera, A. Ghosh, C. Denk, and M. Glencross, "BRDF representation and acquisition," 6 2016. 18

[65] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020. 22, 67

[66] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016. 22

[67] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019. 23, 30, 31, 66

[68] B. D. Brabandere, W. V. Gansbeke, D. Neven, M. Proesmans, and L. V. Gool, "End-to-end lane detection through differentiable least-squares fitting," *CoRR*, vol. abs/1902.00293, 2019. 24

[69] A. Kendall, M. Grimes, and R. Cipolla, "Posenet: A convolutional network for real-time 6-dof camera relocalization," in *ICCV*, 2015. 24

[70] D. S. Chaplot, E. Parisotto, and R. Salakhutdinov, "Active neural localization," in *International Conference on Learning Representations*, 2018. 24

[71] S. K. Gottipati, K. Seo, D. Bhatt, V. Mai, K. Murthy, and L. Paull, "Deep active localization," *IEEE Robotics and Automation Letters*, vol. 4, pp. 4394–4401, Oct 2019. 24

[72] F. Steinbrücker, J. Sturm, and D. Cremers, "Real-time visual odometry from dense rgb-d images," in *ICCV Workshops*, 2011. 24

[73] C. Tang and P. Tan, "Ba-net: Dense bundle adjustment network," *ICLR*, 2019. 24

[74] R. Clark, M. Bloesch, J. Czarnowski, S. Leutenegger, and A. J. Davison, "Ls-net: Learning to solve nonlinear least squares for monocular stereo," *ECCV*, 2018. 24

[75] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997. 24

[76] L. von Stumberg, P. Wenzel, Q. Khan, and D. Cremers, "Gn-net: The gauss-newton loss for deep direct SLAM," *CoRR*, vol. abs/1904.11932, 2019. 24

[77] L. Han, M. Ji, L. Fang, and M. Nießner, "Regnet: Learning the optimization of direct image-to-image pose registration," *CoRR*, vol. abs/1812.10212, 2018. 24

[78] E. Grefenstette, B. Amos, D. Yarats, P. M. Htut, A. Molchanov, F. Meier, D. Kiela, K. Cho, and S. Chintala, "Generalized inner loop meta-learning," *arXiv preprint arXiv:1910.01727*, 2019. 24

[79] M. Lampton, "Damping–undamping strategies for the levenberg–marquardt nonlinear least-squares method," *Computers in Physics*, 1997. 26

[80] F. Richards, "A flexible growth function for empirical use," *Journal of experimental Botany*, 1959. 26

[81] C. V. Nguyen, S. Izadi, and D. Lovell, "Modeling kinect sensor noise for improved 3d reconstruction and tracking," in *2012 second international conference on 3D imaging, modeling, processing, visualization & transmission*, IEEE, 2012. 27

[82] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 303–312, 1996. 27

[83] S. Tulsiani, T. Zhou, A. A. Efros, and J. Malik, "Multi-view supervision for single-view reconstruction via differentiable ray consistency," in *CVPR*, 2017. 28

[84] J. Gwak, C. B. Choy, M. Chandraker, A. Garg, and S. Savarese, "Weakly supervised 3d reconstruction with adversarial constraint," in *2017 International Conference on 3D Vision (3DV)*, 2017. 28

[85] H. Igehy, "Tracing ray differentials," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp. 179–186, ACM Press/Addison-Wesley Publishing Co., 1999. 29

[86] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283, 2016. 30

[87] J. Krishna Murthy, E. Smith, J.-F. Lafleche, C. Fuji Tsang, A. Rozantsev, W. Chen, T. Xiang, R. Lebaredian, and S. Fidler, "Kaolin: A pytorch library for accelerating 3d deep learning research," *arXiv:1911.05063*, 2019. 30

[88] N. Ravi, J. Reizenstein, D. Novotny, T. Gordon, W.-Y. Lo, J. Johnson, and G. Gkioxari, "Pytorch3d." `https://github.com/facebookresearch/pytorch3d`, 2020. 30, 31

[89] J. Valentin, C. Keskin, P. Pidlypenskyi, A. Makadia, A. Sud, and S. Bouaziz, "Tensorflow graphics: Computer graphics meets deep learning," 2019. 30

[90] A. Handa, T. Whelan, J. McDonald, and A. Davison, "A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM," in *ICRA*, 2014. 31

[91] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015. 33, 35

[92] Y. Dong, "Deep appearance modeling: A survey," *Visual Informatics*, vol. 3, no. 2, pp. 59 – 68, 2019. 40

[93] E. L. Denton *et al.*, "Unsupervised learning of disentangled representations from video," in *Advances in neural information processing systems*, pp. 4414–4423, 2017. 41

[94] J.-T. Hsieh, B. Liu, D.-A. Huang, L. F. Fei-Fei, and J. C. Niebles, "Learning to decompose and disentangle representations for video prediction," in *Advances in Neural Information Processing Systems*, pp. 517–526, 2018. 41

[95] M. Chen, Q. Tang, S. Wiseman, and K. Gimpel, "A multi-task approach for disentangling syntax and semantics in sentence representations," *arXiv preprint arXiv:1904.01173*, 2019. 41

[96] Z. Shu, E. Yumer, S. Hadap, K. Sunkavalli, E. Shechtman, and D. Samaras, "Neural face editing with intrinsic image disentangling," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5541–5550, 2017. 41

[97] B. Lu, J.-C. Chen, and R. Chellappa, "Unsupervised domain-specific deblurring via disentangled representations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 10225–10234, 2019. 41

[98] X. Liu, B. V. Kumar, P. Jia, and J. You, "Hard negative generation for identity-disentangled facial expression recognition," *Pattern Recognition*, vol. 88, pp. 1–12, 2019. 41

[99] Z. Li, Z. Xu, R. Ramamoorthi, K. Sunkavalli, and M. Chandraker, "Learning to reconstruct shape and spatially-varying reflectance from a single image," *ACM Trans. Graph.*, vol. 37, Dec. 2018. 41

[100] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241, Springer, 2015. 47

[101] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014. 48, 63

[102] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283, 2016. 48

[103] C. Schlick, "An inexpensive brdf model for physically-based rendering," *Computer Graphics Forum*, vol. 13, no. 3, pp. 233–246, 1994. 49

[104] B. Smith, "Geometrical shadowing of a random rough surface," *IEEE Transactions on Antennas and Propagation*, vol. 15, no. 5, pp. 668–671, 1967. 49

[105] L.-P. Asselin, D. Laurendeau, and J.-F. Lalonde, "Deep svbrdf estimation on real materials," *arXiv preprint arXiv:2010.04143*, 2020. 54, 58

[106] R. A. Albert, D. Y. Chan, D. B. Goldman, and J. F. O'Brien, "Approximate svBRDF estimation from mobile phone video," vol. 37, p. 12, The Eurographics Association, June 2018. 54, 56, 57

[107] Z. Zhou, G. Chen, Y. Dong, D. Wipf, Y. Yu, J. Snyder, and X. Tong, "Sparse-as-possible svbrdf acquisition," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 6, pp. 1–12, 2016. 54, 56, 57

[108] M. Aittala, T. Weyrich, J. Lehtinen, *et al.*, "Two-shot svbrdf capture for stationary materials.," *ACM Trans. Graph.*, vol. 34, no. 4, pp. 110–1, 2015. 54, 56, 57

[109] Y. Dong, G. Chen, P. Peers, J. Zhang, and X. Tong, "Appearance-from-motion: Recovering spatially varying surface reflectance under unknown lighting," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 6, pp. 1–12, 2014. 54, 56, 57

[110] G. Nam, J. H. Lee, D. Gutierrez, and M. H. Kim, "Practical svbrdf acquisition of 3d objects with unstructured flash photography," *ACM Transactions on Graphics (Proc. SIGGRAPH Asia 2018)*, vol. 37, no. 6, pp. 267:1–12, 2018. 54, 56, 57

[111] Z. Xu, J. B. Nielsen, J. Yu, H. W. Jensen, and R. Ramamoorthi, "Minimal brdf sampling for two-shot near-field reflectance acquisition," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 6, pp. 1–12, 2016. 55, 56

[112] R. Vidaurre, D. Casas, E. Garcés, and J. Lopez-Moreno, "Brdf estimation of complex materials with nested learning," in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 1347–1356, 2019. 55, 56

[113] K. Kim, J. Gu, S. Tyree, P. Molchanov, M. Nießner, and J. Kautz, "A lightweight approach for on-the-fly reflectance estimation," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 20–28, 2017. 55, 56

[114] W. Matusik, *A data-driven reflectance model*. PhD thesis, Massachusetts Institute of Technology, 2003. 56

[115] K. J. Dana, B. Van Ginneken, S. K. Nayar, and J. J. Koenderink, "Reflectance and texture of real-world surfaces," *ACM Transactions On Graphics (TOG)*, vol. 18, no. 1, pp. 1–34, 1999. 56

[116] H. P. Lensch, J. Kautz, M. Goesele, W. Heidrich, and H.-P. Seidel, "Image-based reconstruction of spatially varying materials," in *Rendering Techniques 2001*, pp. 103–114, Springer, 2001. 56

[117] H. P. Lensch, J. Kautz, M. Goesele, W. Heidrich, and H.-P. Seidel, "Image-based reconstruction of spatial appearance and geometric detail," *ACM Transactions on Graphics (TOG)*, vol. 22, no. 2, pp. 234–257, 2003. 56

[118] M. Holroyd, J. Lawrence, and T. Zickler, "A coaxial optical scanner for synchronous acquisition of 3d geometry and surface reflectance," *ACM Transactions on Graphics (TOG)*, vol. 29, no. 4, pp. 1–12, 2010. 56

[119] Z. Zhou, Z. Wu, and P. Tan, "Multi-view photometric stereo with spatially varying isotropic materials," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1482–1489, 2013. 56

[120] R. Xia, Y. Dong, P. Peers, and X. Tong, "Recovering shape and spatially-varying surface reflectance under unknown illumination," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 6, pp. 1–12, 2016. 56, 57

[121] J. Wang and E. Olson, "Apriltag 2: Efficient and robust fiducial detection," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4193–4198, IEEE, 2016. 57

[122] Z. Li, M. Shafiei, R. Ramamoorthi, K. Sunkavalli, and M. Chandraker, "Inverse rendering for complex indoor scenes: Shape, spatially-varying lighting and svbrdf from a single image," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2475–2484, 2020. 57, 64

[123] S. Bi, Z. Xu, K. Sunkavalli, M. Hašan, Y. Hold-Geoffroy, D. Kriegman, and R. Ramamoorthi, "Deep reflectance volumes: Relightable reconstructions from multi-view photometric images," *arXiv preprint arXiv:2007.09892*, 2020. 57

[124] S. Bi, Z. Xu, K. Sunkavalli, D. Kriegman, and R. Ramamoorthi, "Deep 3d capture: Geometry and reflectance from sparse multi-view images," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5960–5969, 2020. 57

[125] Z. Li, Z. Xu, R. Ramamoorthi, K. Sunkavalli, and M. Chandraker, "Learning to reconstruct shape and spatially-varying reflectance from a single image," *ACM Transactions on Graphics (TOG)*, vol. 37, no. 6, pp. 1–11, 2018. 57

[126] Z. Li, K. Sunkavalli, and M. Chandraker, "Materials for masses: Svbrdf acquisition with a single mobile phone image," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 72–87, 2018. 58

[127] S. Sengupta, J. Gu, K. Kim, G. Liu, D. W. Jacobs, and J. Kautz, "Neural inverse rendering of an indoor scene from a single image," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 8598–8607, 2019. 58

[128] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015. 59, 60, 61

[129] Z. Li, T.-W. Yu, S. Sang, S. Wang, S. Bi, Z. Xu, H.-X. Yu, K. Sunkavalli, M. Hašan, R. Ramamoorthi, and M. Chandraker, "Openrooms: An end-to-end open framework for photorealistic indoor scene datasets," 2020. 62

[130] "Evermotion archinteriors collection." `http://www.evermotion.org`. 62

[131] "Turbosquid." `http://www.turbosquid.com`. 62

[132] "Chaos group v-ray." `http://www.chaosgroup.com`. 62, 63

[133] M. Oren and S. K. Nayar, "Generalization of lambert's reflectance model," in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, (New York, NY, USA), p. 239–246, Association for Computing Machinery, 1994. 64

[134] A. Tewari, O. Fried, J. Thies, V. Sitzmann, S. Lombardi, K. Sunkavalli, R. Martin-Brualla, T. Simon, J. Saragih, M. Nießner, R. Pandey, S. Fanello, G. Wetzstein, J.-Y. Zhu, C. Theobalt, M. Agrawala, E. Shechtman, D. B. Goldman, and M. Zollhöfer, "State of the art on neural rendering," 2020. 67

[135] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. 67