ABSTRACT

Title of Dissertation: PHOTOREALISTIC 3D MAPPING OF REAL-WORLD SCENES: METHODS AND APPLICATIONS

Jaehoon Choi, Doctor of Philosophy, 2026

Dissertation Directed by: Professor Dinesh Manocha
University of Maryland, College Park, Department of Computer Science

Photorealistic mapping, which aims to reconstruct the three-dimensional world, is a foundational problem in computer vision and graphics. When this reconstruction faithfully captures both the geometry and appearance of the real environment, it produces a digital twin of the real environment. Such high-quality digital twins enable immersive user experiences and scalable data generation through real-to-simulation-to-real pipelines. However, constructing such digital twins remains challenging due to limitations in sensing, reconstruction accuracy, rendering fidelity, and data acquisition and synthesis efficiency. We advance photorealistic 3D mapping through three interconnected research goals: sensing and perception, mapping, and generation.

The first part of our dissertation focuses on sensing and perception, with an emphasis on single-view depth estimation for indoor and dynamic environments to enhance or replace external depth sensors. By leveraging self-supervised learning and integrating sparse geometric priors derived from 3D LiDAR sensors, we introduce SelfDeco. Our method enables accurate metric depth estimation without relying on dense ground-truth depth maps. Complementing this approach, we propose DnD, which

incorporates multi-view 3D reconstructions to infer dense depth in dynamic environments, achieving a 3.6%–10.2% reduction in root-mean-square error compared to prior methods.

The second part studies mapping and explores how classical geometry and graphics pipelines can be combined with neural representations. For object-level reconstruction, we present TMO, which leverages neural implicit surface representations to reconstruct complete geometry, improving geometric accuracy while reducing texture artifacts. To support unbounded scene reconstruction, we introduce LTM, a mesh-based neural rendering approach that produces compact yet geometrically faithful representations. LTM generates optimized meshes that achieve comparable rendering quality while reducing triangle count by $73\times$ and memory usage by $11\times$. Bridging mesh-based modeling and point-based neural rendering, we further present MeshGS, which integrates explicit mesh geometry with 3D Gaussian splatting to achieve high-quality rendering while preserving structural consistency. Finally, we extend these pipelines to omnidirectional sensing through IM360, a framework that enables spherical structure-from-motion and large-scale indoor mapping using 360° cameras.

The third part of the dissertation addresses data generation by leveraging reconstructed digital twins to synthesize high-quality training data for downstream perception tasks. Building on 3D Gaussian splatting, we propose UAVTwin, which constructs high-fidelity digital twins that support controllable data augmentation in scenes with multiple moving humans. B y enabling realistic data generation with accurate annotations, our approach substantially improves learning in difficult-to-capture real-world scenarios, as validated by performance gains in downstream tasks such as person detection and human action recognition. Finally, we introduce UAV4D, which extends dynamic Gaussian splatting to aerial settings and enables robust novel-view rendering of dynamic UAV scenes, achieving a 1.5 dB increase in PSNR and improved visual sharpness.

PHOTOREALISTIC 3D MAPPING OF REAL-WORLD SCENES: METHODS AND APPLICATIONS

by

Jaehoon Choi

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2026

Advisory Committee:
Professor Dinesh Manocha, Chair/Advisor
Professor Maria K. Cameron, Dean's Representative
Professor Ming Lin
Professor Ruohan Gao
Dr. Heesung Kwon

© Copyright by
Jaehoon Choi
2026

Dedication

늘 곁에 계신 헌신적인 어머니와 하늘에 계신 그리운 아버지께

To my devoted mother, always by my side, and to my father in heaven, whom I dearly miss.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my advisor, Professor Dinesh Manocha, for his unwavering support, invaluable guidance, and thoughtful advice throughout my Ph.D. journey. I am also sincerely grateful to Dr. Heesung Kwon, who collaborated with me on many projects, supported difficult decisions, and consistently inspired new and exciting research directions. I would also like to thank my committee members, Professor Maria, Professor Ruohan, and Professor Ming, for their time, thoughtful feedback, and support throughout the dissertation process. I am profoundly grateful to my mother, the strongest and wisest person I know, whose infinite love and support sustained me at every step of this journey. I also wish to honor the memory of my father, whose guidance and values shaped who I am today. I miss him deeply and continue to carry his influence with me. I am thankful to my sister, with whom I have always shared mutual support and constant communication, especially during times when I could not be physically close to my family. Finally, I am deeply grateful to my girlfriend for her unwavering encouragement and support, and for traveling repeatedly to Maryland, Seattle, and San Diego to be by my side throughout this journey.

I am thankful to Professor Changick Kim, my master's advisor, who taught me the research skills that form the foundation of my work and shaped my research mindset as I explored new domains. I am deeply grateful to my long-standing collaborator, Dongki, whose passion and constant motivation have been a tremendous source of inspiration throughout my Ph.D. journey. I appreciate Hyungtae, Sungmin, and Chris for their support and for the many discussions that helped us pursue impactful research directions. I also sincerely appreciate my other collaborators, Yonghan, Jing, Gershom, and Adil, for the meaningful discussions, shared challenges, and continuous encouragement that greatly shaped my research. I would like to express my special gratitude to Donghwan and Suyong, whose close mentorship, thoughtful guidance, and sustained support played a particularly important role in

shaping my research direction and strengthening my work throughout my Ph.D. journey. I would also like to acknowledge those who provided guidance and mentorship across various research projects during my internships, including Rajvi, Qinbo, Changil, Jiabin, Yi, Yinhao, and Amir. Their technical insights and mentorship were instrumental in shaping my work and helping me navigate complex research challenges, and I sincerely appreciate their support.

I would like to extend my heartfelt gratitude to my friends Geonsun, Jongho, Sanghyun, Brian, Kyungjun, Sungbok, Suho, Yooyeon, Hyojung, Keonwoo, Minwook and Yoonkyung for their friendship and encouragement throughout this journey. I am also sincerely thankful to Xijun, Ruiqi, Adarsh, Kasun, Yi-Ling, Niall, Vishnu, Pooja, Laura, Bhrij, Jim, Sanjoy, Mohamed, Samarth and all other GAMMA members for their support and joyful companionship, which truly made this experience memorable.

Table of Contents

Dedication ii

Acknowledgements iii

Table of Contents v

List of Tables x

List of Figures xiii

Abbreviations xx

Chapter 1: Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Photorealistic Mapping . . . 2
- 1.3 Dissertation Overview . . . 5

I. Sensing and Perception 12

Chapter 2: SelfDeco: Self-Supervised Monocular Depth Completion in Challenging Indoor Environments 13
- 2.1 Introduction . . . 13
- 2.2 Related Work . . . 15
- 2.3 Method . . . 16
  - 2.3.1 Training Framework . . . 16
  - 2.3.2 Network Architecture . . . 20
- 2.4 Experiments . . . 24
  - 2.4.1 Datasets . . . 24
  - 2.4.2 Implementation Details . . . 26
  - 2.4.3 Experiments on NAVERLABS indoor datasets . . . 26
  - 2.4.4 Experiments on Public Benchmark Datasets . . . 31
- 2.5 Conclusion . . . 31

Chapter 3: DnD: Dense Depth Estimation in Crowded Dynamic Indoor Scenes 33
- 3.1 Introduction . . . 33
- 3.2 Related Work . . . 35
- 3.3 Method . . . 37
  - 3.3.1 Absolute-Scale Depth Loss . . . 38
  - 3.3.2 Photometric Consistency Loss . . . 38
  - 3.3.3 Flow-Guided Shape Constraint . . . 39
  - 3.3.4 Normal-Guided Scale Constraint . . . 41
  - 3.3.5 Overall Loss Function . . . 43

3.4 Experiments . . . 43
3.4.1 Experimental Results on NAVERLABS dataset . . . 46
3.4.2 Results on NYUv2 and TUM RGB-D datasets . . . 50
3.5 Conclusion . . . 51

II. Mapping 53

Chapter 4: TMO: Textured Mesh Acquisition of Objects with a Mobile Device by using Differentiable Rendering 54
4.1 Introduction . . . 54
4.2 Related Work . . . 57
4.3 Method . . . 59
4.3.1 RGBD-aided Structure from Motion . . . 59
4.3.2 Geometry Reconstruction . . . 61
4.3.3 Texture Optimization . . . 64
4.4 Experiments . . . 66
4.4.1 Experiment Settings . . . 66
4.4.2 Experimental Results . . . 67
4.5 Conclusion . . . 73

Chapter 5: LTM: Lightweight Textured Mesh Extraction and Refinement of Large Unbounded Scenes for Efficient Storage and Real-time Rendering 75
5.1 Introduction . . . 75
5.2 Related Work . . . 78
5.3 Method . . . 80
5.3.1 Initialization of Geometry and Appearance . . . 81
5.3.2 Mesh Extraction and Decimation . . . 83
5.3.3 Appearance and Geometry Optimization . . . 85
5.4 Experiments . . . 89
5.4.1 Experimental Results . . . 89
5.5 Conclusion . . . 94

Chapter 6: MeshGS: Adaptive Mesh-Aligned Gaussian Splatting for High-Quality Rendering 95
6.1 Introduction . . . 95
6.2 Related Work . . . 98
6.3 Method . . . 99
6.3.1 Geometry Reconstruction . . . 100
6.3.2 Mesh-based Gaussian Splatting . . . 102
6.3.3 Training for Tightly-bound and Loosely-bound Gaussian Splats . . . 104
6.4 Experiments . . . 106
6.4.1 Experiment Settings . . . 106
6.4.2 Experimental Results . . . 107
6.4.3 Ablation Study . . . 109
6.4.4 Compositional scenes . . . 112
6.5 Conclusion . . . 113

Chapter 7: IM360: Large-scale Indoor Mapping with 360° Cameras 115
7.1 Introduction . . . 115
7.2 Related Work . . . 118
7.2.1 Structure from Motion . . . 118
7.2.2 Textured Mesh Reconstruction . . . 120
7.3 Method . . . 120
7.3.1 Dense Matching and Structure from Motion . . . 121
7.3.2 Geometric Reconstruction . . . 124
7.3.3 Texture Optimization . . . 125
7.4 Experiments . . . 126
7.4.1 Experiment Settings . . . 126
7.4.2 Experimental Results . . . 127
7.4.3 Additional Experiments and Ablation Study . . . 132
7.5 Conclusion . . . 133

III. Generation 135

Chapter 8: UAVTwin: Neural Digital Twins for UAVs using Gaussian Splatting 136
8.1 Introduction . . . 136
8.2 Related Work . . . 139
8.3 Building Digital Twin . . . 140
8.3.1 Preliminaries: 3D Gaussian Splatting . . . 141
8.3.2 Multi-sequence Gaussian Splatting (MsGS) . . . 142
8.3.3 Mask Refinement . . . 144
8.3.4 Geometry Reconstruction . . . 145
8.3.5 Training Objectives for MsGS . . . 146
8.3.6 Mesh Reconstruction . . . 146
8.4 Neural Data Generation . . . 148
8.4.1 Camera Trajectory Generation . . . 148
8.4.2 Synthetic Human Placement . . . 149
8.4.3 Scene Composition . . . 149
8.5 Experiments . . . 151
8.5.1 Datasets and Implementation Details . . . 151
8.5.2 Background Rendering . . . 152
8.6 Conclusion . . . 153

Chapter 9: UAV4D: Dynamic Neural Rendering of Human-Centric UAV Imagery using Gaussian Splatting 155
9.1 Introduction . . . 155
9.2 Related Work . . . 158
9.3 Method . . . 160
9.3.1 Preliminaries . . . 160
9.3.2 Initialization . . . 161
9.3.3 Joint Human-Scene Reconstruction . . . 163
9.3.4 3D Gaussian Scene Modeling . . . 165

9.4 Experiments . . . 167
9.4.1 Quantitative Comparison . . . 168
9.4.2 Qualitative Comparison . . . 171
9.5 Conclusion . . . 172

Chapter 10: Conclusion, Limitations, Future Works 173
10.1 Summary . . . 173
10.2 Limitations and Future Works . . . 175
10.2.1 Sensing and Perception: . . . 175
10.2.2 Mapping: . . . 176
10.2.3 Generation: . . . 178

Appendix A: Appendix for Dense Depth Estimation in Crowded Indoor Dynamic Scenes 180
A.1 Datasets . . . 180
A.2 Evaluation Metrics . . . 182
A.3 Implementation Details . . . 182
A.4 DnD with Localization . . . 183

Appendix B: Appendix for Textured Mesh Acquisition of Objects with a Mobile Device by using Differentiable Rendering 185
B.1 Data Collection . . . 185
B.2 Implementation Details . . . 186

Appendix C: Appendix for Lightweight Textured Mesh Extraction and Refinement of Large Unbounded Scenes for Efficient Storage and Real-time Rendering 187
C.1 Implementation Details . . . 187
C.2 Additional Explanations . . . 188
C.3 Additional Qualitative Comparisons . . . 189

Appendix D: Appendix for Neural Digital Twins for UAVs using Gaussian Splatting 192
D.1 Details of Human Recognition Experiments . . . 192
D.1.1 Details of Person Detection Experiments . . . 192
D.1.2 Details of Action Recognition Experiments . . . 192
D.2 Multi-view Regularization . . . 193
D.3 Implementation Details . . . 194
D.4 Mask Refinement . . . 196

Appendix E: Appendix for Dynamic Neural Rendering of Human-Centric UAV Imagery usingn Gaussian Splatting 199
E.1 Dataset & Training Details . . . 199
E.1.1 Dataset Details . . . 199
E.1.2 Training Details . . . 200
E.2 Baseline Details . . . 202
E.3 Experimental Results . . . 203
E.3.1 Zoom-in Human Regions . . . 203

Bibliography 205

List of Tables

2.1 Depth completion results on NAVERLABS indoor dataset. "S" represents *supervised* training methods and "SS" denotes *self-supervised* training methods. Different from other depth completion methods, MD2 is based on self-supervised depth estimation with monocular video sequences. 28

2.2 depth completion results on the NYUv2 dataset "S" represents *supervised* training methods and "SS" denotes *self-supervised* training methods. 30

2.3 depth completion results on the KITTI validation set. "S" represents *supervised* training methods and "SS" denotes *self-supervised* training methods. 30

3.1 Quantitative comparisons with the state-of-the-art depth estimation algorithms. F means the evaluation results in the human regions and F+B indicates the evaluation results on depth values over the entire scene. MS and Dept denote the Metro Station dataset and the Department Store dataset, respectively. DnD (L_{ph} only) is trained only using the photometric consistency loss. In the MS dataset, DnD shows improvement in depth for human regions by 3.6 % in the RMSE metric. In terms of depth for entire regions, DnD reduces the RMSE by 10.2 %. 44

3.2 Contributions of our proposed modules to the evaluation results on the Metro Station (MS) dataset. F means the evaluation results on depth values in the human regions and F+B indicates the evaluation results on depth values over the entire scene. The median scaling is applied to DnD (L_{ph} only) for absolute scale depth prediction. Compared to DnD (L_{ph} only), we improve the RMSE by 26.4% in the entire scene and by 32.2% in the human regions. 47

3.3 Quantitative comparisons of the NYUv2 and TUM RGB-D dataset. In both static and dynamic scenes, DnD shows improved performance compared to recent methods. . . . 51

4.1 Quantitative comparison on our dataset. Ac and Av denote the AR-catpure dataset and ARKit-video dataset, respectively. We improve the LPIPS metric by 8.9% in AR-capture and by 3.8% in ARKit-video. 71

4.2 Quantitative comparison on DTU [1]. Chamfer distance is employed as the evaluation metric. COLMAP is superior to other neural geometry reconstructions. Our training process can improve the NeuS by 8 %. 71

5.1 **Quantitative Comparison on mip-NeRF 360 Dataset.** "-" denotes a missing implementation. The best and second best-performing algorithms for each metric are bolded and underlineed. 90

5.2 **Disk Storage (MB), Number of Vertices (V) and Faces (F).** The unit for the number of vertices and faces is 10^3. "M" denotes the 10^6 unit. We save 73$\times$ the number of triangles and reduce 11$\times$ disk storage. 91

5.3 **Quantitative Comparison.** We report PSNR ↑, SSIM ↑, and LPIPS ↓ on Deep Blending and Tanks & Temples datasets. 93

5.4 **Disk Storage (MB), #Vertices (V) and #Faces (F).** 93

5.5 **FPS and GPU memory** on mip-NeRF 360 dataset 94

6.1 **Quantitative Comparison on outdoor mip-NeRF 360 Dataset.** Splat-based rendering demonstrates superior performance compared to mesh-based neural rendering techniques such as LTM [2] and BakedSDF [3]. MeshGS* did not utilize any regularization techniques to tightly align Gaussian splats; instead, it solely applied image loss L_{img} for training. Compared to GaMeS [4] and SuGaR [5], our method enhances the PSNR by 0.9 dB and 1.3 dB, respectively. 108
6.2 **Quantitative Comparison on indoor mip-NeRF 360 Dataset.** Splat-based rendering exhibits superior performance when compared to mesh-based neural rendering techniques. MeshGS* did not utilize any regularization techniques to tightly align Gaussian splats; instead, it solely applied image loss L_{img} for training. Compared to GaMeS [4] and SuGaR [5], our method enhances the PSNR by 2.0 dB and 0.2 dB, respectively. . . 109
6.3 **Quantitative Comparison on Deep Blending Dataset.** We report PSNR ↑, SSIM ↑, and LPIPS ↓ on Deep Blending datasets [6] 109

7.1 **Quantitative Evaluation of Camera Localization Performance** on the Matterport3D dataset [7]. Our method demonstrates superior results in terms of registration accuracy and achieves comparable performance in pose estimation across most scenes. 130
7.2 **Quantitative evaluation of camera Localization Performance** on the Stanford2D3D dataset [8]. Our method outperforms all other approaches in terms of registration accuracy and pose estimation. 130
7.3 **Quantitative Comparison of Rendering Performance** on the Matterport3D Dataset. Our method shows higher rendering quality of our method, IM360,across all scenes, achieving a 3.5 PSNR over the TexRecon [9]. The best-performing algorithms for each metric are shown in boldface. 131

8.1 **Quantitative Comparison on Okutama-Action Dataset.** Except for 3DGS [10], all methods are designed for surface reconstruction. The "Mask" column indicates whether a mask is applied when measuring the training loss. Compared to 2DGS [11] and PGSR [12], our method enhances the PSNR by 1.23 dB and 2.79 dB, respectively. The best and second best-performing algorithms for each metric are bolded and underlined. 149
8.2 **Human Recognition Comparison on the Okutama-Action [13] of YOLOv8 family [14], R3D [15], Swin3D [16].** Compared to models trained with "Real" dataset, we improve the mAP_{50} metric by 2.5 % - 13.7 % and top-1 accuracy by 2% - 7%. 152
8.3 **Quantitative Comparison on NeRF-OSR Dataset.** We report PSNR ↑, SSIM ↑, and LPIPS ↓. 152

9.1 **Quantitative Comparison** of our method with recent comparative works on the three datasets. Our method achieves the best performance across every category except for PSNR on the Manipal-UAV dataset. Please see the Appendix for detailed comparisons on the four scenes in each dataset. Red, orange, and yellow indicate the first, second, and third best performing algorithms for each metric, respectively. 168

9.2 **Quantitative Results.** (a) shows PSNR and SSIM across datasets, focusing on **human-only** regions cropped using precise human masks. Due to the small size of the human regions, it is not possible to compute the LPIPS metric. Consequently, we report only the PSNR and SSIM metrics. (b) presents ablation on VisDrone dataset. See Appendix for details. 168

A.1 (a) Most large-scale indoor datasets do not have dynamic scenes except for Baidu [17]. (b) All datasets collected from the internet do not contain metric 3D models. To train depth estimation algorithms in crowded indoor dynamic scenes, a dataset must contain three properties: dynamic, metric and large amount of images. Different from previous datasets (a) and (b), NAVERLABS contains large amounts of dynamic scenes with the metric 3D model in indoor environments. 181

A.2 The performance with or without Kapture [18] in the MS dataset. The upper line in the table shows the results of the camera poses, which are used in reconstruction of the 3D model (SfM + MVS). The lower line in the table indicates the performance of the whole pipeline including visual localization under the sparse 3D model from SfM (without MVS). Although the SfM point clouds are highly sparse and visual localization methods has pose uncertainty, the comparable results show the robustness of our framework. 184

D.1 **Details of Train Dataset.** "K" denotes the 10^3 unit. 193

D.2 **AP Comparison on the Okutama-Action [13] of YOLOv8 family.** 193

D.3 **Quantitative Comparison on NeRF-OSR** 196

D.4 **Quantitative Comparison on DroneSplat Dataset.** In the manuscript, we present the mean values across six scenes. This table provides detailed results for each individual scene. 197

D.5 **Quantitative Comparison on Okutama Dataset.** Drone1-Noon comprises six video sequences, each with distinct characteristics. In the manuscript, we present the mean values across these six scenes, while this table provides detailed results for each individual scene. 197

D.6 **Quantitative Comparison on Okutama Dataset.** Drone1-Morning comprises seven video sequences, each with distinct characteristics. In the manuscript, we present the mean values across these six scenes, while this table provides detailed results for each individual scene. 197

D.7 **Quantitative Comparison on Okutama Dataset.** "-" denotes a missing implementation. Drone2-Noon comprises five video sequences, each with distinct characteristics. In the manuscript, we present the mean values across these six scenes, while this table provides detailed results for each individual scene. 197

D.8 **Quantitative Comparison on Okutama Dataset.** Drone2-Morning comprises seven video sequences, each with distinct characteristics. In the manuscript, we present the mean values across these six scenes, while this table provides detailed results for each individual scene. 198

List of Figures

1.1 Overview of our research. The upper boxes (in orange) represent the closed-loop pipeline, while the lower boxes (in gray) illustrate our work across specific research areas. Each gray box corresponds to a chapter number of the thesis and highlights the key research topics that connect and extend each stage of the study. The core methods introduced in this thesis are organized across Chapters 2–9: SelfDeco [19] in Chapter 2, DnD [20] in Chapter 3, TMO [21] in Chapter 4, LTM [2] in Chapter 5, MeshGS [22] in Chapter 6, IM360 [23] in Chapter 7, UAVTwin [24] in Chapter 8, and UAV4D [25] in Chapter 9. . 10

2.1 Depth completion results in challenging scenes: Metro Station (top) and the Department Store (bottom) from NAVERLABS indoor dataset. Our method takes an RGB image and a sparse depth map (projection of LiDAR pointcloud) as input, and predicts a dense depth map. (low high; grey means empty depth values.) 15

2.2 Overview of our proposed framework. The black block with a dash-dotted line shows the depth completion network (details in Fig. 2.4). 17

2.3 Examples of auto-masks computed after one epoch. Auto masks (black pixels) usually contain low-texture scenes or non-Lambertain surfaces where photometric loss is unreliable. 18

2.4 Our model consists of three main components: (1) an RGB encoder that extracts RGB features, reducing the spatial resolution of the input images to consider the global context; (2) a depth encoder that extracts sparse depth features from sparse input via sparse convolutional blocks; (3) a decoder that takes both RGB and depth features, and transforms them into dense depth maps. Data flow contains convolution operations. . . 21

2.5 Samples from the NAVERLABS indoor dataset. We include more samples and demonstrate many challenging features from our datasets in supplementary video. 25

2.6 SD represents a sparse depth. We design 4 architectures: (a) 1e-1d: an encoder and a decoder; (b) 2e-1d: an RGB encoder, a depth encoder, and a decoder; (c) 2es-1d: an RGB encoder, a depth encoder with sparse convolutions, and a decoder; (d) 2es-1dp: the same architecture as Fig. 2.4. 27

2.7 Examples of depth artifacts generated by 1e-1d model. The top (NYUv2) and bottom (Dept. 1F) figures in third column shows irregular sparse patterns copied from sparse inputs and blurred depth boundaries 27

2.8 Qualitative comparison with "UNET" [26], "nUNET" [27], and "CSPN" [28] on the NAVERLABS indoor dataset. As mentioned in Sec. 2.4.1.1, the groundtruth of the Metro Station dataset is the results of MVS. (low high; grey means empty depth values.) 29

3.1 With traditional 3D reconstruction methods [29, 30], we can obtain a 3D model on complex and crowded indoor environments. Our novel approach, DnD, takes both an RGB image and a sparse depth map projected from this 3D model as input. Our method predicts dense and absolute-scale depth maps of single view dynamic scenes. 34

3.2 Overview of our proposed approach. The black box with a dotted line shows our monocular depth estimation network consisting of an RGB encoder, a depth encoder, and a decoder. For training, the network takes consecutive temporal frames (I_t, $I_{t'}$) with corresponding sparse depth maps (D_t, $D_{t'}$) projected from the 3D model as input. Our training method is a combination of four terms. 1) The depth loss encourages our network to encode absolute scale from a depth input. 2) The photometric consistency loss is based on view synthesis and regularizes the network training for static background regions. 3) The flow-guided shape constraint enables the network to complete missing pixels in human regions with proper depth values and eliminate visual artifacts. 4) The normal-guided scale constraint enables our network to estimate accurate and absolute-scale depths on moving people. 37
3.3 (a) The process to obtain a surface normal from the pixel coordinates and the corresponding depth values. (b) shows that the ground normal $\tilde{n}$ is not aligned with the y-direction of the camera coordinates. The bottom images indicate the process for propagating the reliable scale for the person in the dashed red circle. The red point in the third column means the small patch B, indicating the human's ground contact point. In the fourth column, the yellow and blue points show the ground-patch intersection and uniformly sampled points in the human mask respectively. 41
3.4 Qualitative comparisons of the Metro Station dataset (1-2 rows) and the Department Store dataset (3-4 rows). Our network takes an image (1st column) and a depth input projected from the 3D model (2nd column), and predicts dense depth maps (7th column). Ground truth (8th column) collected from 3D LiDAR is used to evaluate our method. Depth results (3-6 columns) are estimated by the existing depth estimation algorithms. Compared to MiDaS and MC, DnD generates detailed and high quality depth maps in non-human regions and also shows depth maps in human regions with sharp depth discontinuities. All color maps use the jet color map (low high; grey means empty depth values). 46
3.5 Qualitative comparisons of the proposed modules. (a) We note that a part of the human is cut off in the depth map. The visual artifacts that look similar to RGB image patterns often appear in human depth regions. (b) FSC encourages our network to capture sharper details, fill in missing parts of the human regions, and remove visual artifacts. (c) We observe that NSC estimates depth with sharp depth discontinuities and also recovers absolute scale depth values for human regions. All color maps use the jet color map (low high; grey means empty depth values). 49
3.6 (a) shows $\delta_{1.25}$ metric for our method corresponding the number of uniform-sampled input depth points, and (b) indicates the performance degradation for scenes with high density crowds in MS and Dept: 8.8% and 9.7% for FMVS (Fast-MVSNet [31]), 20% and 5.2% for MC [32], 10.2% and 17.1% for BTS [33], 6.2% and 6% for ours. 50

4.1 Example reconstruction results collected from a smartphone in the wild. (a) Data acquisition setup. (b) Images captured from a smartphone. (c) A reconstructed mesh. (d) A novel view of textured mesh. Our proposed method can reconstruct the high-quality geometric mesh with a visually realistic texture. 55
4.2 Example objects collected by a smartphone in the wild 56

4.3 Overview of our pipeline. It is comprised of three steps: RGBD-aided Structure from Motion (Sec. 4.3.1), Geometry Reconstruction (Sec. 4.3.2), and Texture Optimization (Sec. 4.3.3). Our method builds on classical 3D reconstruction and texture mapping. To overcome the limitations of classical methods, we apply a neural geometry representation and neural texture fine-tuning to update 3D geometry and texture maps. 60
4.4 Overview of Geometry Reconstruction (Section 4.3.2) and Texture Optimization (Section 4.3.3). Our goal in geometry reconstruction is to train geometry network f_{sdf} and color network f_{color} via volumetric rendering with two training steps. In step 1, we leverage prior geometric information from the MVS algorithm [34]. Then, our method utilizes a sparse voxel octree to efficiently sample points near the surface. In texture optimization, we first utilize a classical texture reconstruction to generate texture maps for a given mesh. Then, our method applies a differentiable rendering to fine-tune texture maps. 63
4.5 We show the benefits of RGBD-aided Structure from Motion. (a) illustrates the 3D point clouds obtained from initial depths with initial poses. In (c), we accumulate the 3D point clouds (by using filtered depths and refined poses) from RGBD-aided SfM. For a detailed description, we visualize the zoomed-in blue and red regions using filtered depth. Initial depth maps and poses from ARkit are input to the geometry reconstruction in Section 4.3.2 and its results are shown in (b). The results of our pipeline in (d) lead to higher quality reconstructions than (b) which does not apply RGBD-aided SfM. 68
4.6 Reconstruction of our method, VolSDF [35], TSDF-Fusion [36], and a mesh from point clouds of ACMP [34] processed with screened Poisson surface reconstruction [37]. Note that the neural surface reconstruction is more perceptually convincing in the ARKit-video dataset. 69
4.7 Qualitative comparison between (a) Ours, (b) Waechter et al. [9], (c) CMO [38], (d) ACMP [34], and (e) NeuS [39]. The result of our method is perceptually closer to the original scene. 70
4.8 Qualitative comparisons of texture optimization algorithms based on differentiable rendering: (a) our method, (b) Lin et al. [40], and (c) Munkberg et al. [41]. 72

5.1 Example results using LTM showing mesh geometry (top-left) and novel-view synthesis (right). Bottom right plot compares performance of different methods on mip-NeRF360 outdoor dataset. Our method achieves comparable rendering quality to BakedSDF while reducing the mesh size considerably. 76
5.2 **Overview of Our Pipeline**. It consists of three steps: Initialization (Sec. 5.3.1), Mesh Processing (Sec. 5.3.2), and Optimization (Sec. 5.3.3). Our method focuses on baking the initial appearance and geometry representation, trained by volumetric rendering, into an explicit mesh and a memory-efficient texture conducive to real-time rendering. We employ classical mesh decimation to minimize storage requirements, introduce the optimization stage to facilitate rasterization-based rendering and refine the detailed mesh structure. 80

5.3 **Our Approach:** In Section 5.3.1, we start training both the appearance f_{color} and geometry f_{sdf} representations for a large-scale scene using volumetric rendering (shown in orange). Then, in Section 5.3.2, our method extracts the mesh from f_{sdf} and significantly simplifies its structure, resulting in the lightweight mesh M (shown in blue). In Section 5.3.3, we jointly train vertex deformation Δx and f_{color} computed from appearance modeling. 82
5.4 **Visual comparison of Our Method, NeRF2Mesh [42], BakedSDF [3], and MobileNeRF [43].** Our method and NeRF2Mesh show shading mesh and wireframe mesh extracted without texture, respectively. 90
5.5 **Qualitative Comparisons with Existing Methods.** We visually compare our proposed technique with NeRF2Mesh [42], BakedSDF* [3], and MobileNeRF [43]. The visual results of BakedSDF* generated in its first stage through volumetric rendering. Red and blue dotted lines emphasize subtle differences in rendering quality. 92
5.6 **Zoom-in Shading Mesh** to see more geometric details. 94

6.1 **Our Approach:** In Section 6.3.1, we jointly train the geometry f_{sdf} and appearance g_{color} using differentiable volumetric rendering (shown in orange). Then, we extract the lightweight mesh M which is important for the initialization of Gaussian splats. Next, in Section 6.3.2., we present the mesh-based Gaussian splatting for training, which involves the removal of Gaussian splats (green Gaussian splats) occluded by mesh surface. Given the distance d_i between each Gaussian splat and its corresponding triangle, we distinguish between tightly-bound Gaussian splats (blue splat) and loosely Gaussian splats (red splat). In Section 6.3.3, we introduce the training and densification strategy for both tightly-bound Gaussian splats and loosely Gaussian splats. 100
6.2 The examples of extracted mesh from BakedSDF [3], SuGaR [5], and LTM [2]. We employ LTM [2] to extract a lightweight mesh with moderate triangle count, using it for the initialization of Gaussian splats. The primary reason is that LTM utilizes a minimal number of triangles while maintaining geometric quality. Zoomed-in regions visualize that the mesh frequently lacks highly detailed geometry or complex structures. "K" and "M" denotes the 10^3 and 10^6 units, respectively. 102
6.3 **Qualitative Comparisons with Existing Methods.** We visually compare our method with SuGaR [5], 3DGS [10], and LTM [2]. For a detailed description, we visualize the zoomed-in blue and red regions. 110
6.4 **Qualitative Comparisons of Our Method and SuGaR [5]**. We show shading mesh without texture. 111
6.5 **The number of Gaussian Splats** The unit for the number of Gaussian splats is 10^3. Compared to the original 3DGS, we utilize 30% fewer Gaussian splats. MeshGS* did not utilize any regularization techniques to tightly align Gaussian splats; instead, it solely applied image loss L_{img} for training. MeshGS-(a) and MeshGS*-(c) shows the number of tightly-bound Gaussian splats. MeshGS-(b) and MeshGS*-(d) denotes the number of loosely-bound Gaussian splats. 111

6.6 **Visualization results of both loosely-bound and tightly-bound Gaussian splats.** The second column shows wireframe mesh extracted without texture. The third column visualizes only the tightly-bound Gaussian splats, while the fourth column illustrates the loosely-bound Gaussian splats. The fifth column presents the final rendering results achieved by our method. 112
6.7 **Scene Composition**: Rendering obtained by embedding the synthetic human to a reconstructed scene. The first row visualizes the shading mesh without texture. The second row shows the composition of a human in the scene reconstructed by our method. The third row visualizes the composition of the same human in the scene generated by LTM [2]. 113

7.1 While the wide field of view of the omnidirectional camera significantly reduces the data acquisition time, it introduces challenges such as sparse views and occlusions. According to the Table, Matterport3D [7] and Stanford2D3D [8] are captured with significantly lower view density compared to ScanNet [44]. Our approach provides an effective solution to handle issues in sparsely scanned indoor environments and enables efficient reconstruction and rendering using omnidirectional cameras. 116
7.2 **Overview of Our Pipeline**. It is comprised of three steps: Spherical Structure from Motion (Sec. 7.3.1), Geometry Reconstruction (Sec. 7.3.2), and Texture Optimization (Sec. 7.3.3). In Section 7.3.1, spherical SfM is utilized to estimate the camera poses for omnidirectional images in challenging indoor environments. Then, in Section 7.3.2, our method apply cubemap projection and estimate monocular depth and normal to train geometry f_{sdf} representation using volumetric rendering. In Section 7.3.3, we initlize texture using classical texture mapping and jointly optimize diffuse, specular texture along with small MLP for modeling appearance. 122
7.3 Our method, which applies cubemap projection and trains neural surface reconstruction using multiple perspective images, achieves highly accurate geometry reconstruction. In contrast, neural surface reconstruction designed for ERP images (OmniSDF [45]) and Gaussian Splatting-based surface reconstruction (SuGaR [46] and VCR-GauS [47]) fail to generate a clean mesh. 124
7.4 We provide a visual comparison between sparse matching perspective SfM and our proposed dense matching spherical SfM. In (a), ERP images are converted into a cubemap representation, after which feature matching is performed across all 36 possible image pairs, resulting in sparse and noisy correspondence matches. (b) demonstrates our approach, which directly finds dense and accurate correspondences on ERP images, thereby facilitating the construction of a detailed 3D structure. 128
7.5 **Qualitative Comparison of SfM results on Matterport3D.** By leveraging dense features on equirectangular projection (ERP) images, our method effectively finds correspondences in textureless regions. 129
7.6 We visualize the shading meshes extracted without texture on Matterport3D. Leveraging the estimated poses from IM360, we achieve accurate surface reconstruction, demonstrating the effectiveness of our SfM method in generating decent geometric meshes from omnidirectional data. 132

7.7 Localization results in multi-floor scenes of Matterport3D. Multi-floor scenes pose significant challenges for visual localization due to narrow corridors and staircases connecting different floors. Despite these difficulties, IM360 demonstrates the ability to estimate poses, ensuring robust localization in large-scale indoor environments. . . . 133
7.8 **Qualitative Comparisons with Existing Methods.** We visually compare our proposed approach with TexRecon [9], SparseGS [48], and ZipNeRF [49]. Our method can render high frequency details and results in lower noise. 134

8.1 UAVTwin processes video captured by the UAV as input, enabling data generation for training UAV-based human recognition methods. 137
8.2 **UAVTwin framework.** In Section 8.3, our approach first constructs a digital twin using UAV-based images captured at different times. We introduce MsGS, a novel 3DGS method to analyze varying appearance images and reconstruct a clean mesh, Gaussian splats, and an MLP for novel-view synthesis. Then, in Section 8.4, our method generates data by compositing foreground humans rendered in Blender with backgrounds rendered using trained Gaussian splats. 142
8.3 **The core components of MsGS.** From a video sequence captured at different times with varying appearances, we define a sequence embedding q_i. For novel-view image rendering, we select a specific sequence. Our color MLP f takes as input the sequence embedding, viewing direction embedding, per-Gaussian embedding, and base color. The output color is modulated to account for appearance variations. 143
8.4 **Mask Refinement.** (a) is an example of training images with dynamic objects. (b) is its segmentation masks $\overline{M}$ using GroundingSAM [50]. (c) is the entity segmentation masks $\hat{M}$ [51]. Based on the SAM masks in (b), we add entity masks with high photometric loss (red dotted boxes) and remove those with low photometric loss (blue dotted boxes), resulting in the refined masks shown in (e). 144
8.5 **Qualitative Results of Synthetized Data.** The first row illustrates the camera trajectory, while the second row presents a sample training image from the sequence used as a per-sequence embedding input for rendering our MsGS. Data is generated using different trajectories with sequence embeddings extracted from training data: (a)+(e) orbit (Noon 1.2.2), (b)+(f) altitude varying (Morning 2.1.1), (c)+(g) yaw rotation (Morning 2.1.7), and (d)+(h) translational (Noon 2.2.2). 147
8.6 **Qualitative Comparison of MsGS (Our method), GOF [52], 2DGS [11], PGSR [12], and 3DGS [10].** In the first row, we show shading mesh without texture. The red dotted lines illustrates the superior geometric quality of our method. We visualize the rendering results (2-4 rows). Red dotted lines and yellow arrow emphasize difference in rendering quality. 150

9.1 (Top) Comparison between the Existing Benchmarks for Dynamic Neural Rendering and UAV captured Datasets. UAVs typically cover wide areas from a top-down perspective, capturing dynamic scenes with a distinct setup: they contain numerous human instances, each occupying only a tiny fraction of the image area. (Bottom) Comparison of various recent dynamic gaussian splatting methods. UAV4D is among the first to enable rendering of UAV scenes using only a single-view camera. 156

9.2 **Overview of Our Approach**: (a) We begin the initialization process for reconstructing the background scene and the human mesh. (b) We reconstruct the background mesh, determine the optimal scale for alignment, and position the human mesh by identifying the ground contact points. (c) We jointly optimize the human and background Gaussians for rendering. . . . 161
9.3 **Human Placement.** We identify the 2D contact point from the bounding box and use the depth from the mesh to unproject the 3D ground contact point. . . . 166
9.4 **Qualitative Comparison of VisDrone [53] (top row) and Okutama-Action Dataset [13] (bottom row).** We visualize the zoomed-in blue and red regions, which emphasize the dynamic humans. Our method demonstrates superior capability in reconstructing small, moving humans compared to other existing approaches. . . . 169
9.5 **Qualitative Comparison of Manipal-UAV Dataset [54].** We visualize the zoomed-in blue and red regions, which emphasize the dynamic humans. Our method demonstrates superior capability in reconstructing small, moving humans compared to other existing approaches. . . . 170
9.6 **Scene Editing Samples**: Due to the nature of scene decomposition, we can move or remove the humans within the black-dotted boxes. The fourth image illustrates the decomposition of the background and humans. For additional visual results, please refer to the supplementary materials. . . . 171

A.1 The network architecture of our proposed method. The red block, green block, and blue block are an RGB encoder, a depth encoder, and a decoder. We concatenate a projected depth map and a binary mask, which indicates where the projected depth values exist. The convolutional filter is defined as (filter size, stride size, input channel, output channel). The RGB encoder adopts a ResNet 34 [55] backbone pretrained with ImageNet [56], and the depth encoder consists of two convolutional blocks and three residual blocks. . . . 182
A.2 DnD (2-3 columns) estimates depth from the image and the projected depth map, which is derived from the 3D model. Since DnD with Kapture [18] (4-5 columns) exploit the sparse 3D model from SfM, the projected depth maps are also highly sparse, and the background details of the estimated depth maps decrease. However, the overall qualities, especially moving people, are predicted robustly. (low high; grey means empty depth values.) . . . 184

C.1 Qualitative comparison between (a) Ours, (b) NeRF2Mesh [42], and (c) BakedSDF [3]. Our method and NeRF2Mesh show shading mesh and wireframe mesh extracted without texture, respectively. . . . 189
C.2 Qualitative comparison between (a) Ours, (b) NeRF2Mesh [42], (c) BakedSDF* [3], and (d) MobileNeRF [43]. The visual results of BakedSDF* generated in its first stage through volumetric rendering. . . . 190
C.3 Qualitative comparison between (a) Ours and (b) NeRF2Mesh [42]. Our method and NeRF2Mesh show shading mesh and wireframe mesh extracted without texture, respectively. . . . 190
C.4 Qualitative comparison between (a) Ours, (b) NeRF2Mesh [42], and (c) MobileNeRF [43]. . . . 191

C.5 Comparison of mesh quality when optimized with decimation (columns 1, 3) vs. without decimation (columns 2, 4). 191

D.1 **Qualitative Comparison of MsGS (Ours), "MsGS w/o Refine", "MsGS w/o Mask"** on the DroneSplat dataset. Without masks, the 4th columns shows that significant artifacts appear in regions occupied by transient and dynamic objects. When using masks refined by the mask refinement module, our method (2nd column) effectively reduces artifacts compared to "MsGS w/o Refine" (3rd column), which relies solely on SAM masks. 195

E.1 **Visual Examples of Initial Estimates.** Each initial estimate is used as input for the subsequent human-scene reconstruction process. 200

E.2 **Visual Examples of Zoom-in Human Regions** on VisDrone and Okutama-Action Datasets 204

List of Abbreviations

3D	Three Dimension
AR	Augmented Reality
MLP	Multilayer Perceptron
NeRF	Neural Radiance Fields
SfM	Structure-from-Motion
TSDF	Truncated Signed Distance Field
UAV	Unmanned Aerial Vehicles
VIO	Visual-Inertial Odometry
VR	Virtual Reality
Root-Mean-Square Error	RMSE

Chapter 1: Introduction

1.1 Motivation

Understanding and reconstructing the three-dimensional (3D) world is fundamental for autonomous robotic systems and interactive AR/VR platforms that operate in real environments. Because these systems must navigate, localize, and interact with physical spaces, they require accurate 3D representations to support downstream tasks such as collision avoidance, motion planning, and scene understanding. In robotics, accurate sensing and depth reconstruction reduce the risk of collisions [57] and enable safer and more reliable navigation in complex environments [58, 59]. AR applications [60, 61] similarly rely on precise 3D mapping. Reconstructing real-world environments in 3D and deploying them in VR further enables fully immersive user experiences.

Beyond end-user applications, digital twins of real-world environments enable synthetic data generation [62–67] for training deep learning models to perceive and understand 3D scenes. This capability supports a pipeline for sensing, reconstructing, generating, and re-perceiving the 3D world. Such workflows are commonly referred to as real-to-sim pipelines [67,68], in which real-world data are used to build accurate and realistic 3D environments. These simulated environments allow models and robots to learn in simulation and subsequently be deployed in real-world settings through simulation-to-real transfer.

To enable this pipeline, real-to-simulation approaches seek to construct photorealistic and geometrically accurate digital twins from real-world data. The effectiveness of a real-to-simulation pipeline depends on three key requirements. First, the digital twin must be metrically and geometrically accurate to faithfully represent the underlying physical environment. Second, high-quality rendering is essential for achieving visual realism in the digital twin that closely matches the real-world environ-

ment. If data acquisition or synthesis is costly or difficult, the approach becomes impractical for large-scale learning and evaluation. We address these challenges through three research goals: sensing and perception, mapping, and generation. Advancing each of these goals contributes to meeting the core requirements needed for digital twins to effectively support a wide range of downstream applications.

1.2 Photorealistic Mapping

Depth Estimation: Depth information is a fundamental component of 3D reconstruction across a wide range of applications [69–71]. Active sensors, such as LiDAR and RGB-D cameras, directly provide depth measurements that can be used to construct 3D maps. In parallel, depth estimation from monocular images has advanced rapidly in recent years. Similar to other computer vision tasks, learning-based depth estimation methods can be broadly categorized into supervised [69, 72–74] and self-supervised approaches [75–77]. In the 3D domain, however, many learning-based methods are limited by the availability of accurate ground-truth depth data, which is often expensive or impractical to obtain at scale. Seminal works such as SfM-Learner [75] and Monodepth2 [76] extended self-supervised learning paradigms to depth estimation, enabling networks to be trained using only image sequences without requiring ground-truth depth annotations. More recently, depth foundation models such as MiDaS [74] have emerged by leveraging large-scale datasets that combine real-world and synthetic data. These models demonstrate strong generalization across diverse environments and open new possibilities for depth perception in scenarios where traditional sensors are unavailable, unreliable, or impractical. However, most learning-based depth estimation approaches focus on monocular imagery and therefore inherently suffer from scale ambiguity, meaning they can recover relative depth relationships and scene structure but cannot determine absolute or metric distances.

Structure-from-Motion: Structure-from-Motion (SfM) is a foundational area of research in 3D computer vision. Most photogrammetry pipelines rely on incremental SfM to estimate camera poses and recover sparse geometric structure from collections of images. One of the most influential milestones in this field is COLMAP [29], which has become a widely adopted standard for multi-view reconstruction. Classical SfM pipelines typically consist of establishing feature correspondences across multiple images, solving multi-view geometry problems, and performing bundle adjustment to jointly refine camera poses and 3D structure. This pipeline has proven to be robust and reliable across a wide range of scenarios. More recently, learning-based SfM approaches [78, 79], often referred to as 3D foundation models, have emerged. These methods employ large-scale neural networks, frequently based on transformer architectures [80], and are trained on extensive datasets to directly infer camera poses, point clouds, and other 3D attributes.

Surface Reconstruction: Classical surface reconstruction methods, such as Poisson surface reconstruction [37] and TSDF fusion [81], can be used to generate meshes from point clouds. Traditionally, meshes are reconstructed by first applying multi-view stereo to calibrated images to obtain dense point clouds, followed by Poisson surface reconstruction. When depth sensors are available, depth measurements from multiple viewpoints can be fused to construct a signed distance function, from which a mesh is extracted using the Marching Cubes algorithm [82]. More recently, neural surface reconstruction methods [39] have demonstrated strong performance. Neural representations such as Neural Radiance Fields (NeRF) and 3D Gaussian Splatting have been used to reconstruct implicit surfaces, including signed distance fields or explicit meshes, enabling high-quality 3D surface reconstruction.

Neural Rendering: Traditionally, mesh-based rendering [83] has been widely used in computer graphics to visualize 3D models on 2D screens through rasterization [84]. With the introduction of Neural Radiance Fields (NeRF) [85], neural rendering has gained significant attention due to its ability to

produce photorealistic 3D renderings. NeRF represents a scene using a multilayer perceptron (MLP) that encodes RGB radiance and density, which are rendered through differentiable volumetric rendering [86]. Despite their high rendering quality, NeRF-based methods are known for extremely slow rendering speeds, which limit their practicality in real-time applications. To address this limitation, the milestone work on 3D Gaussian Splatting [10] was introduced. 3D Gaussian Splatting represents a 3D scene as a collection of anisotropic 3D Gaussians, each encoding both geometric and appearance information. By leveraging a rasterization-based rendering process, this approach achieves high-fidelity visual results while enabling real-time rendering performance.

360 Imaging: Advances in specialized sensing technologies, such as fisheye [44] and 360° cameras [7, 8], have the potential to significantly enhance a wide range of computer vision techniques. These sensors help overcome limitations [44, 87, 88] of conventional camera hardware and open up many promising research opportunities. In particular, 360° cameras provide extremely wide fields of view, which can drastically reduce scanning time and capture substantially more scene information compared to conventional perspective cameras. However, due to their unique imaging characteristics, many 3D reconstruction problems [23] such as structure from motion, feature matching, and depth estimation remain relatively underexplored and less mature for omnidirectional cameras than for standard perspective camera models.

Digital Twin: A digital twin [89] is a virtual representation of physical products, environments, or processes, commonly realized within computer game engines or industrial CAD systems. Recent advances in computer vision and computer graphics have significantly expanded the applicability of digital twins across a wide range of domains, including robotics, mixed reality, autonomous driving, and facility management. Traditionally, constructing digital twins has relied heavily on specialized sensing hardware such as LiDAR or depth cameras, as well as labor-intensive, hand-crafted CAD

modeling workflows. However, recent progress in 3D foundation models [74], Neural Radiance Fields (NeRF) [85], and 3D Gaussian Splatting [10] has substantially lowered the barrier to creating high-fidelity virtual replica of real-world environments.

1.3 Dissertation Overview

The core of our research can be distilled into a three-stage pipeline, as illustrated in Fig. 1.1. (1) **Sensing**: Single-view depth estimation to enhance or complement external sensors for 3D world reconstruction. (2) **Mapping**: Reconstructing accurate 3D geometry and enabling photorealistic rendering using neural fields obtained from conventional or 360° cameras. (3) **Generation**: Applying the digital twin to generate data for training aerial recognition algorithms.

Part I: Sensing and Perception

Our research focuses on single-view (monocular) depth estimation from two complementary perspectives. First, we investigate training methodologies for scenarios in which obtaining depth annotations is extremely difficult or impractical. Second, we aim to achieve accurate metric depth estimation in challenging settings such as dynamic environments or map-free scenarios. Although 3D LiDAR [90] is a widely used sensing modality, it often produces sparse point clouds and is susceptible to artifacts introduced by dynamic elements, including moving people.

In Chapter 2, to address these limitations, we introduces SelfDeco [19], a self-supervised algorithm for training depth completion networks without relying on dense ground-truth depth maps. Our approach is designed for challenging indoor environments characterized by textureless regions, glossy and transparent surfaces, dynamic elements such as moving people, wide and diverse depth ranges, and complex ego-motion during data capture. Our proposed architecture leverages deep stacks of sparse

convolutional blocks to extract informative sparse depth features, together with pixel-adaptive convolutions to effectively fuse image and depth information. We evaluate SelfDeco on the NYUv2 [91], KITTI [92], and NAVERLABS indoor datasets [93], and observe improvements of 5–34% in root-mean-square error (RMSE) compared to existing approaches [26, 28, 94–96].

In Chapter 3, instead of relying on 3D LiDAR or depth sensors, we explore an approach that incorporates 3D priors derived from maps reconstructed using classical structure-from-motion techniques [29, 30] into learning-based monocular depth estimation for dynamic scenes. Our proposed method, DnD [20], leverages sparse 3D points to infer dense depth maps in dynamic environments, eliminating the need for iterative 3D model reconstruction. Unlike supervised approaches that rely on dense depth annotations from active sensors, DnD operates without such supervision, making it well suited for real-world deployment. To handle depth estimation for non-rigidly moving people without explicitly tracking their motion, we introduce two complementary constraints. We demonstrate that this approach achieves consistent improvements over recent depth estimation methods on the NAVERLABS dataset [93], which includes complex and crowded scenes, improving RMSE by 3.6%–10.2% compared to comparative methods [26, 31–33, 74].

Part II: Mapping

Our work explores how to integrate classical computer vision and graphics algorithms [83, 97] with learning-based methods to build practical 3D reconstruction systems. In Chapter 4, we present TMO [21], a method for capturing high-quality textured meshes of 3D objects. This approach enables the collection of 3D assets consisting of geometry and texture, which form core components of real-world 3D maps. We observe that traditional reconstruction pipelines, including Structure from Motion (SfM) [29], Multi-View Stereo (MVS) [30], and classical texture mapping [9], often introduce se-

vere artifacts in both geometry and texture. To address these issues, we propose an RGBD-aided SfM method that combines the complementary strengths of Visual-Inertial Odometry (VIO) [98] and SfM. For geometry reconstruction, we adopt a neural implicit surface representation [39] with volumetric rendering, which produces more complete and smoother surfaces than classical methods optimized primarily for robustness in unconstrained environments. We evaluate geometric accuracy using Chamfer distance, and our method achieves an 8% improvement on the DTU dataset [1]. After extracting a mesh from the learned implicit representation, classical texture mapping [9] is applied, followed by differentiable rendering to fine-tune texture maps using a photometric loss, significantly reducing artifacts and improving visual quality.

Extending beyond object-level reconstruction, we develop a similar reconstruction method for large-scale scenes using mesh-based neural rendering. Recent neural surface reconstruction methods [3] extract explicit meshes with appearance encoded in texture maps as neural features, but rely on the Marching Cubes algorithm [82], resulting in high storage overhead. This limitation poses significant challenges for mobile and large-scale applications. In Chapter 5, we introduce LTM [2], which achieves realistic rendering with compact mesh representations optimized for unbounded, large-scale environments while preserving geometric fidelity. We propose a method for producing optimized meshes with low triangle budgets and high fidelity in both geometry and appearance by combining advances in neural SDF baking with classical mesh simplification techniques [99], together with a joint appearance–geometry refinement step. The resulting meshes achieve visual quality comparable to or better than recent neural meshing and baking methods while maintaining high geometric accuracy despite substantial reductions in triangle count. This makes the representations efficient for storage, transmission, and rendering on mobile hardware. We validate the proposed method on large unbounded scenes from the mip-NeRF 360 [88], Tanks & Temples [100], and Deep Blending datasets [6],

achieving comparable rendering quality with a 73× reduction in triangle count and an 11× reduction in memory footprint.

In Chapter 6, we propose MeshGS [22], a novel hybrid rendering approach that integrates mesh representations with 3D Gaussian splatting to achieve high-quality rendering of reconstructed real-world scenes. While 3D Gaussian splatting [10] has gained attention for its ability to produce high-fidelity rendering, many graphics applications, including games and AR/VR systems, continue to rely on mesh-based representations. MeshGS introduces a distance-based Gaussian splatting strategy that aligns Gaussian splats with the underlying mesh surface while removing redundant splats that do not meaningfully contribute to rendering quality. Based on their distance to the mesh surface, Gaussian splats are categorized as tightly bound or loosely bound. Tightly bound splats are flattened and explicitly aligned with mesh geometry to preserve structural consistency, while loosely bound splats model residual appearance details and compensate for artifacts in reconstructed meshes. We further propose a training strategy that explicitly binds Gaussian splats to mesh geometry and jointly optimizes both types of splats. Several regularization terms are introduced to precisely align tightly bound Gaussian splats with the mesh surface during training. We validate the effectiveness of MeshGS on large, unbounded scenes from the mip-NeRF 360 [88] and Deep Blending datasets [6]. Our method surpasses recent mesh-based neural rendering approaches [2, 3, 42] by achieving up to a 2 dB improvement in PSNR and outperforms mesh-based Gaussian splatting [4, 5] methods by 1.3 dB PSNR, particularly on outdoor scenes in the mip-NeRF 360 dataset, demonstrating improved rendering quality.

To address the limitations of perspective cameras, including small fields of view and inefficiency in mapping large spaces, we extend the mapping pipeline to omnidirectional 360-degree cameras. We first introduce EDM [101], a dense feature matching method designed for omnidirectional images that performs robustly in indoor environments with textureless regions and repetitive patterns. Building on

these dense features, Chapter 7 presents IM360 [23], a complete pipeline for large-scale indoor mapping using 360° cameras. Our pipeline is formulated using a spherical Structure from Motion approach based on unit bearing vectors and spherical projection. By leveraging dense matching features specifically designed for 360° images, our approach for SfM demonstrates improved robustness and accuracy in image registration. In addition, we incorporate mesh-based neural rendering techniques and introduce a texture optimization method that refines texture maps and accurately captures view-dependent appearance by jointly modeling diffuse and specular components. We evaluate the proposed pipeline on large-scale indoor scenes, demonstrating its effectiveness in real-world settings. IM360 achieves a 3.5 dB improvement in PSNR for textured mesh reconstruction and attains state-of-the-art performance in camera localization and registration on the Matterport3D [7] and Stanford2D3D [8] datasets.

Part III: Generation

Mapping technologies enable the creation of digital twins that support realistic and accessible simulation environments. Many learning-based perception algorithms are constrained by the limited availability of training datasets with accurate ground-truth annotations. This challenge is particularly pronounced in UAV-based perception due to the high degrees of freedom in UAV motion and imaging viewpoints. While graphics engines can generate synthetic training data, such data often suffers from a significant domain gap [102]. In contrast, digital twin enables realistic data generation by faithfully replicating real-world scenes, substantially reducing this gap.

In Chapter 8, we extend the pipeline to a 3D Gaussian Splatting framework through UAVTwin [24]. Our method constructs digital twins that support data augmentation for UAV perception by synthesizing dynamic foreground elements, including multiple moving humans, within complex background scenes. We combine 3D Gaussian splatting for background reconstruction with controllable

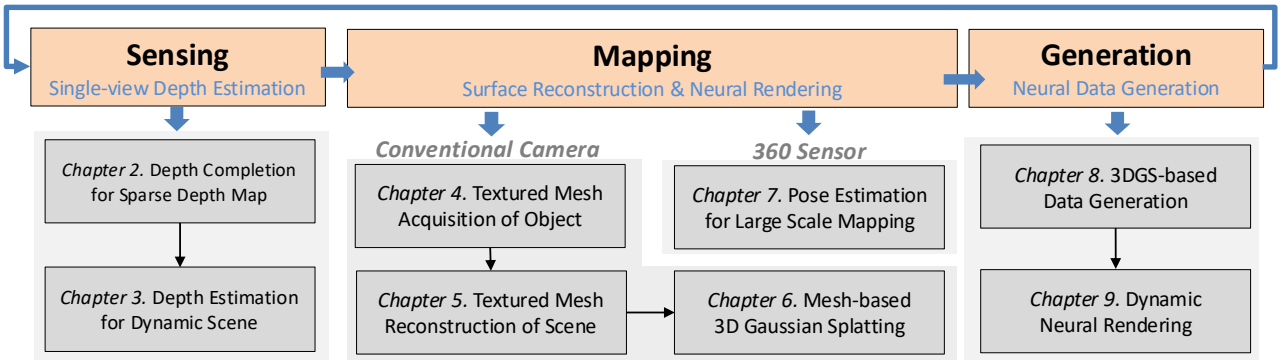


Figure 1.1: Overview of our research. The upper boxes (in orange) represent the closed-loop pipeline, while the lower boxes (in gray) illustrate our work across specific research areas. Each gray box corresponds to a chapter number of the thesis and highlights the key research topics that connect and extend each stage of the study. The core methods introduced in this thesis are organized across Chapters 2–9: SelfDeco [19] in Chapter 2, DnD [20] in Chapter 3, TMO [21] in Chapter 4, LTM [2] in Chapter 5, MeshGS [22] in Chapter 6, IM360 [23] in Chapter 7, UAVTwin [24] in Chapter 8, and UAV4D [25] in Chapter 9.

synthetic human models that exhibit diverse appearances and motions, enabling automated generation of high-quality synthetic datasets with accurate annotations. UAVTwin substantially improves learning in environments with significant dynamic variation, where conventional 3D Gaussian splatting methods often suffer from rendering artifacts. To address these artifacts, we introduce a novel appearance modeling strategy and a mask refinement module that enhance 3D Gaussian Splatting training in the presence of multiple dynamic objects and large appearance variations. We demonstrate improved neural rendering quality, achieving a 1.23 dB increase in PSNR compared to recent methods [10–12, 52]. In addition, we validate the effectiveness of data augmentation by showing performance gains in downstream tasks including person detection and human action recognition.

In Chapter 9, we introduce UAV4D [25], an approach for photorealistic rendering of dynamic real-world scenes captured by UAVs. Despite recent advances in dynamic neural rendering, existing approaches do not adequately address UAV-specific challenges such as monocular capture, top-down viewpoints, and scenes containing multiple small, moving humans. UAV4D leverages a 3D foundation

model [79] to estimate scene backgrounds and camera poses, together with a human mesh reconstruction model to jointly reconstruct background geometry and human actors. We introduce a novel strategy that resolves scale ambiguity arising from bone-length perspective and identifies human–scene contact points, enabling consistent placement of human and background meshes within a shared world coordinate system. The SMPL human model [103] and reconstructed background mesh are used to initialize Gaussian splats, enabling unified and coherent rendering of dynamic scenes. We evaluate the proposed framework on three challenging UAV-captured datasets [13, 53, 54], each containing 10–50 humans and distinct scene characteristics. The results demonstrate clear improvements over existing methods [104–106] in novel-view synthesis, achieving a 1.5 dB increase in PSNR and improved visual sharpness.

I. Sensing and Perception

Chapter 2: SelfDeco: Self-Supervised Monocular Depth Completion in Challenging Indoor Environments

2.1 Introduction

Depth completion has been widely studied in the field of robot navigation, computer vision, and autonomous driving. Its goal is to convert a sparse depth map from active depth sensors such as LiDAR or RGB-D cameras to a dense depth map. In practice, robots with accurate sensing and dense depth maps have a lower chance of collision [57] and are better able to compute safe navigation routes through challenging environments [58, 59]. Currently, most recent methods for depth completion rely on supervised learning [107–109], requiring large amounts of high-quality and dense depth maps as ground truth. Unfortunately, generating dense depth maps is expensive and challenging due to the sparse and noisy depth values from other active sensors. Even existing expensive 3D LiDAR sensors only have a limited number of scan lines and provide sparse depth measurements.

Self-supervised methods with monocular videos that are able to train depth completion neural networks without the need for dense depth maps. These methods [75, 76, 110, 111] are based on an unsupervised learning framework that use depth and camera ego-motion from monocular videos and use photometric loss as supervision. This loss measures the difference between a reference image and a synthesized image obtained by the depth-guided reprojection of other views into the reference view. However, the previous self-supervised training process for depth completion [26, 112] may not work well on public indoor datasets such as NYUv2 [91]. This is due to large areas of textureless regions and scenes captured by complex ego-motions of a handheld camera [113, 114]. Large indoor environments are more challenging not only because of the above issues but also because of large regions of glossy and transparent surface, non-Lambertian surfaces, longer and diverse ranges, and moving people. To

overcome these issues, our proposed training process requires effective ways to compute complex camera ego-motions attached to a robot, mask out regions where photometric loss is unreliable, and output depth maps robust to diverse depth range. In this paper, we address the problem of developing self-supervised depth completion methods that work robustly on general indoor environments.

Main Results: We improve the training framework by leveraging learning-based local features to obtain accurate relative poses, making robust matching with auto-masking and minimum reprojection loss [76], and predicting inverse depth using a sigmoid function. However, we observe that merely masking out all regions still leads to depth artifacts including blurred boundaries and irregular patterns copied from sparse depth input. Therefore, we also propose a novel architecture with two unique components: sparsity-aware convolutions that extract features properly from sparse input and pixel-adaptive convolutions which propagate this information to the whole depth prediction with the guidance of image features. We highlight the effectiveness of our method by evaluating our approach on challenging datasets such as the Metro Station or the Department Store (Fig. 2.1). We also demonstrate the applicability of our method on datasets for which prior self-supervised depth estimation and completion fail to provide reliable depth maps. **Our novel contributions** can be summarized as follows:

- We develop a robust and stable training framework for self-supervised depth completion in challenging indoor environments.
- We propose a novel architecture that can extract sparse depth features guided by high-level image features.
- Our method outperforms prior self- and fully-supervised methods on challenging indoor datasets like NAVERLABS. We also achieve the state-of-the-art performance among self-supervised methods on public benchmark datasets, including KITTI and the NYUv2 dataset based on vari-

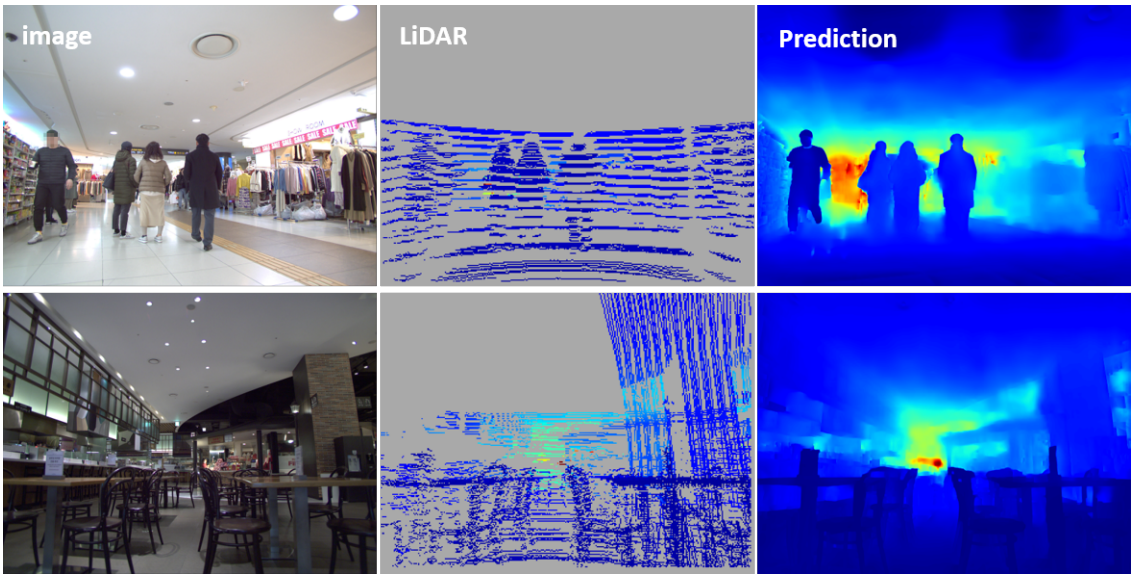


Figure 2.1: Depth completion results in challenging scenes: Metro Station (top) and the Department Store (bottom) from NAVERLABS indoor dataset. Our method takes an RGB image and a sparse depth map (projection of LiDAR pointcloud) as input, and predicts a dense depth map. (low high; grey means empty depth values.)

ous evaluation metrics.

2.2 Related Work

Recently, depth completion methods using a deep neural network have shown promising results. Uhrig et al. [115] propose a sparsity-invariance convolution layer to handle sparse depth or features by using mask normalizations. Eldesokey et al. [27, 116] propose the normalized convolution to produce both a confidence map and dense depth output. In [108], they further extend their work into uncertainty estimation with depth completion methods. Teixeira et al. [117] use this confidence propagation from [116] to compute dense depth from LiDAR. Cheng et al. [28, 107] utilize the neural network to learn the affinity among neighboring pixels in order to perform the propagation process. In addition to using RGB images, some methods take advantage of information from other modalities, e.g., surface normals [118–120] or segmentation [121]. Yang et al. [95] predict the posterior over depth maps

with a conditional prior network. Some works [122, 123] only focus on an indoor depth completion task. However, all aforementioned works on indoor scenarios are limited to a small-scale indoor environment and handle non-LiDAR indoor data (e.g., on the NYU-Depth-v2 [91] and Matterport3D [7] datasets). In this paper, we introduce challenging indoor scenarios such as the department store or metro station containing LiDAR sensor data.

The lack of dense depth maps is a significant limitation of supervised depth completion. As an alternative, several current works study self-supervised depth completion methods. Ma et al. [26] present self-supervised methods based on a photometric loss with pose estimation using the PnP method. Yang et al. [95] present a conditional prior network (CPN) that predicts a probability of depths at each pixel. Zhang et al. [124] propose an end-to-end framework to jointly estimate pose and depth without the PnP method. Yoon et al. [96] combine monocular depth estimation and Gaussian process-based depth regression for depth completion.

2.3 Method

2.3.1 Training Framework

In Fig. 2.2, our proposed depth completion model takes a target view image I_t and a sparse depth S_t as input, and outputs a dense inverse-depth $\hat{d}_t$. We invert $\hat{d}_t$ to compute a dense depth map $\hat{D}_t$. In our method, the source view images contains its two adjacent temporal frames, i.e., $I_s \in \{I_{t-1}, I_{t+1}\}$, although including a larger temporal context is possible. Also, we estimate the relative pose $T_{t\to s}$ between the target image I_t and the set of source images I_s by using a Perspective-n-Point (PnP) [125] with Random Sample Consensus (RANSAC) [126].

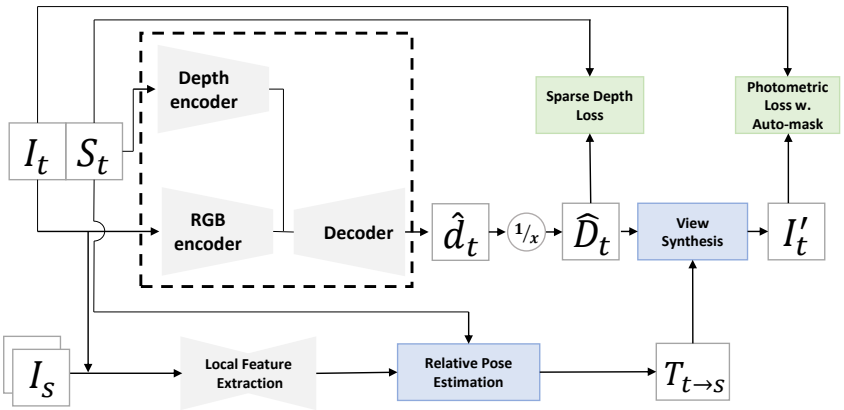


Figure 2.2: Overview of our proposed framework. The black block with a dash-dotted line shows the depth completion network (details in Fig. 2.4).

2.3.1.1 Relative Pose Estimation

Although most prior works [76, 124] adopt learning-based pose estimation, we observe that learning-based pose estimation reveals degradation in performance when applied to challenging environments such as indoor scenes [113, 114, 127]. Additionally, according to recent findings [128], learning-based pose estimation methods suffer from scale ambiguity problems and make it difficult to provide absolute-scale depth supervision. Unlike self-supervised depth estimation, pose estimation via the PnP method is aligned with the depth completion task because our model needs to predict absolute scale depth.

Our method first detects learning-based features and computes their corresponding descriptors via learning-based algorithms. We select R2D2 [129], which shows better performance than traditional hand-crafted features on the visual localization benchmarks. Given the predicted feature in source and target images, we can compute high-quality correspondences by using a FLANN-based search algorithm [130] and then output the relative camera pose by solving PnP with corresponding sparse

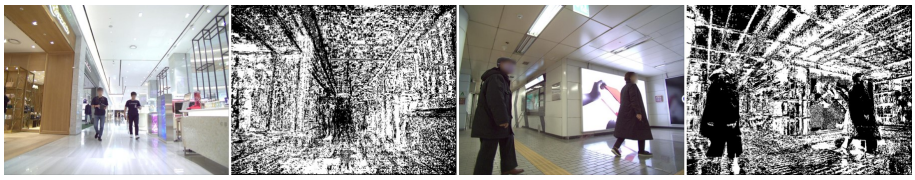


Figure 2.3: Examples of auto-masks computed after one epoch. Auto masks (black pixels) usually contain low-texture scenes or non-Lambertain surfaces where photometric loss is unreliable.

depths. Compared to hand-crafted features used by [26], dense correspondences obtained by R2D2 enable us to fully utilize sparse depth measurements with corresponding feature points for PnP pose estimation.

2.3.1.2 Photometric Loss

Given the camera intrinsic matrix K and the relative pose $T_{t\to s}$, we compute the warped pixel coordinates and synthesize target image I'_t from source images I_s via a bilinear sampling function. Following the self-supervised depth estimation [75, 110], we use the combination of the L1 and SSIM [131] as the photometric loss L_{ph}:

$$\begin{aligned} L_{ph}(I_t, I'_t) &= \alpha \frac{1 - \text{SSIM}(I_t, I'_t)}{2} + (1-\alpha) \parallel I_t - I'_t \parallel, \\ I'_t(x) &= I_s \langle \pi(K T_{t\to s} \hat{D}_t(x) K^{-1} x_h) \rangle \ . \end{aligned} \tag{2.1}$$

where $x_h = [x^T, 1]^T$ are the homogeneous coordinates from the target image, π gives pixel coordinates, and $\langle \cdot \rangle$ denotes the bilinear sampling function. We adopt the minimum reprojection loss and auto-masking from [76]. Our methods often face the occlusion issue, meaning that some pixels from frame I_t do not appear in either I_{t-1} or I_{t+1} frame due to moving people. To mitigate this occlusion

issue, our per-pixel minimum photometric loss is defined as follows,

$$L_{photo} = \min_{I_s} L_{ph}(I_t, I'_t). \tag{2.2}$$

Additionally, auto-masking encourages our method to mask out static pixels for which the appearance does not change between the current frame and temporally adjacent frames. This auto-masking results in removing pixels belonging to low-texture scenes (e.g. ceiling or fluorescent light) that have undesirable effects on the minimum photometric loss. Furthermore, these masks often remove non-Lambertian surfaces (e.g. mirrors or digital bilborad screens), which are common in the department store or the metro station.

2.3.1.3 Sparse Depth Loss

Our proposed model can learn to encode absolute scale from sparse depth loss by aligning the depth prediction $\hat{D}_t$ with sparse depth S_t. We use the L1-norm to penalize the difference between $\hat{D}_t$ and S_t over Ω, where sparse depth is available,

$$L_{depth} = \sum_{x \in \Omega} \left| \hat{D}_t(x) - S_t(x) \right| . \tag{2.3}$$

2.3.1.4 Total Loss Function

Following [26, 76, 110], we also incorporate an edge-aware smoothness loss function L_{smooth} over the predicted depth map to regularize the depth in texture-less low-image gradient regions.

$$L_{smooth} = \mid \partial_x d^* \mid e^{-|\partial_x I_t|} + \mid \partial_y d^* \mid e^{-|\partial_y I_t|}, \tag{2.4}$$

where ∂_x, ∂_y denotes the gradients along the either the x or y direction. $d^* = \hat{d}_t / \overline{\hat{d}_t}$ is the normalized inverse depth [111]. Our overall loss function is a weighted sum over the aforementioned losses,

$$L_{tot} = L_{photo} + \lambda_d L_{depth} + \lambda_s L_{smooth} \tag{2.5}$$

where λ_d and λ_s denote weighting terms selected through a grid search.

2.3.2 Network Architecture

In our training framework, the objective function defines a trade-off between a sparse depth loss, which enables our network to predict absolute scale depth on sparse regions, and a photometric loss, which encourages the network to predict dense depth where sparse information does not exist. In challenging indoor environments, self-supervised methods with photometric loss have shown that finding the optimal depth value is very difficult because the loss only comes from the appearance difference between target and synthesized images; measuring these differences is difficult due to the large non-texture regions, repeating structures, and occlusions.

The sparse depth loss might easily overwhelm the whole self-supervised training, and the depth

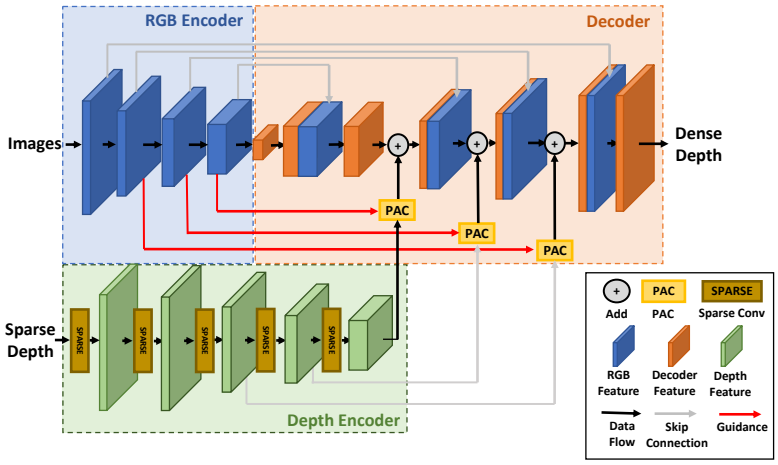


Figure 2.4: Our model consists of three main components: (1) an RGB encoder that extracts RGB features, reducing the spatial resolution of the input images to consider the global context; (2) a depth encoder that extracts sparse depth features from sparse input via sparse convolutional blocks; (3) a decoder that takes both RGB and depth features, and transforms them into dense depth maps. Data flow contains convolution operations.

predictions often result in overfitting on sparse depth input. Both feature extraction from the sparse depth and feature fusion schemes of multi-modal data are critical to avoid such negative effects. Our network adopts a late fusion scheme that uses each encoder for different sources because image and sparse depth features are heterogeneous. Throughout several experiments in Sec. 2.4.1.1, a late fusion scheme is shown to be particularly suitable for self-supervised depth completion. We focus on designing an effective architecture that leverages sparse depth features to fill the empty regions rather than getting stuck producing sparse depth prediction.

2.3.2.1 Sparse Convolution

In order to extract features from sparse depth, recent depth completion methods fall into two groups. One group [27, 115] designs shallow networks with sparse convolution layers due to the poor performance of standard convolution layers on sparse input. In contrast, the other group [26, 28, 107] demonstrates that employing standard convolution layers yields better performance with their training

schemes; these methods also perform well on different levels of sparsity. Comparing both approaches, we conclude that deep stacks of standard convolution layers are key to handling sparse depth input. Therefore, we choose to combine these two approaches by stacking sparse convolution layers [115] deeper. Sparse convolution blocks take a sparse depth feature and a binary observation mask as inputs, which both have the same spatial size. Since an observation mask indicates the reliability of each location of the feature maps, the convolution operations only attend on reliable input features which are involved with sparse depth input. As a result, the depth encoder is able to propagate depth information from reliable sparse depth input to its surrounding depth features. In contrast to sparsity invariant convolutions [115], we do not apply mask normalization, which causes a degradation problem due to the small values extracted from the depth encoder when stacking deep layers.

2.3.2.2 Pixel-Adaptive Convolution (PAC)

Recent works utilize operations like concatenation or element-wise addition to fuse the information from both the RGB and the depth encoder. However, this simple method is not the optimal way to fuse the information from two modalities because extracted depth features are not guided by RGB images in a late-fusion manner. To achieve this, we require spatially-variant convolution weights that different convolution kernels depending on image features are applied to different spatial positions of the depth features. Thus, we adopt pixel-adaptive convolutions (PACs) introduced by Su et al. [132]. Assume that the PACs take a sparse depth feature map $\mathbf{v}$ and an image feature $\mathbf{f}$ as input, and produce the output sparse depth feature $\mathbf{v}'$. We denote the convolution weight as $\mathbf{W}$ and the bias term as $\mathbf{b}$. PACs augment the spatially invariant convolution weight $\mathbf{W}$ with a local-adaptive kernel function $\mathbf{K}$.

Following the notation in [132], the PAC convolution is defined as

$$\mathbf{v}'_i = \sum_{j \in \Omega(i)} K(\mathbf{f}_i, \mathbf{f}_j)\mathbf{W}[\mathbf{p}_i - \mathbf{p}_j]\mathbf{v}_j + \mathbf{b} \tag{2.6}$$

where $\mathbf{f}$ are image features from the RGB encoder and act as guidance for the PACs in the depth encoder. High-level image features are preferable because they contain image context information. The kernel function $\mathbf{K}$ has a standard Gaussian form: $K(\mathbf{f}_i, \mathbf{f}_j) = exp(-\frac{1}{2}(\mathbf{f}_i - \mathbf{f}_j)^T(\mathbf{f}_i - \mathbf{f}_j))$. This kernel computes the correlation between image features and guides the standard convolutional weights to generate depth features depending on the content learned from the RGB encoder. This spatially-variant kernel is helpful because, for instance, depth features on a person should not be applied to generate depth maps on the wall. Thus, PACs enable our proposed architecture to generate intermediate depth features consistent with image content.

2.3.2.3 Decoder

The decoder is similar to Ma et al [26] with one major difference. Unlike their method, our proposed network predicts inverse depth instead of depth. Thus, we replace the ReLU nonlinearity function with sigmoid activation. We observe that applying a sigmoid function is critical for training stability and is more robust to diverse depth ranges on challenging indoor scenes. Without inverse depth prediction, we fail to train the network proposed by [26] on our challenging indoor datasets.

2.4 Experiments

2.4.1 Datasets

To demonstrate that our proposed method performs robustly in the challenging environment, we experiment with NAVERLABS indoor dataset. Additionally, we evaluate our method on two benchmarks: the NYUv2 dataset and KITTI. For indoor scenes, we report five evaluation metrics: root mean squared error (RMSE), mean absolute relative error (ABS Rel), and δ_i ,which means the percentage of predicted pixels for which the relative error is less than a threshold i. The RMSE is represented in mm scale. For the KITTI benchmark, we use four official metrics for evaluation.

2.4.1.1 NAVERLABS indoor dataset

NAVERLABS dataset [133] comprises data collected from three different places: Department Store B1 (Dept. B1), Department Store 1F (Dept. 1F), and Metro Station (MS). All data were collected using a mapping robot equipped with two 16-channel LiDAR sensors and six RGB cameras with 2592×2048 resolution. LiDAR SLAM is used to estimate sensor poses for our dataset, and this pose information enables us to generate depth maps. We captured 17K images for Dept. 1F, 32K images for Dept. B1, and 61K images for MS. The MS dataset was captured at a highly crowded metro station.

We accumulate consecutive LiDAR sweeps and project the point clouds onto each of six cameras in order to generate depth maps. These depth maps can be used as sparse input and ground truth for evaluation. In order to create sparse input, we accumulate 0.3 seconds of LiDAR scans, which correspond to one LiDAR scan since the LiDAR is sampled at 10 Hz (see the second column in Fig. 2.5). In the Dept. 1F and Dept. B1 datasets, we collect 1 second of LiDAR scans to generate ground

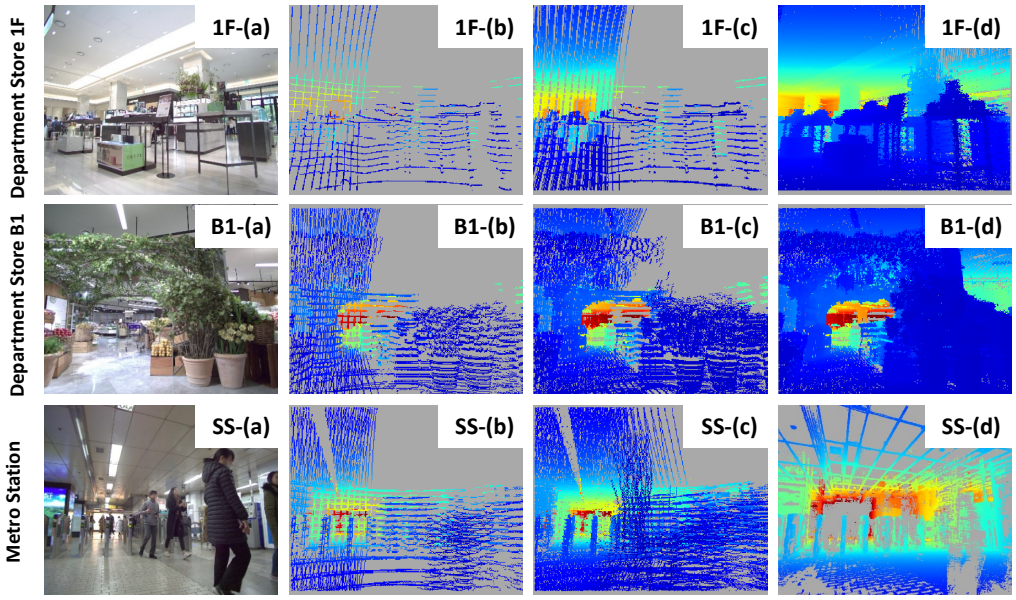


Figure 2.5: Samples from the NAVERLABS indoor dataset. We include more samples and demonstrate many challenging features from our datasets in supplementary video.

truth depth for training supervised depth completion algorithms (see the third column in Fig. 2.5). However, the depth maps generated from the MS dataset are too noisy to serve as ground truth due to noise caused by moving objects (see SS-(c) in Fig. 2.5). For evaluation, we select the high-quality depth maps generated from projecting a LiDAR 3D model onto the camera frames in the Dept. 1F and Dept. B1 datasets (see 1F-(d) and B1-(d) in Fig. 2.5). In contrast, we use the SfM method [29] to reconstruct a sparse 3D model of the MS dataset and MVS system [30] to produce dense depth maps instead of noisy depth maps from LiDAR point clouds. Given the pose from LiDAR SLAM, we can obtain a 3D model aligned with the LiDAR data. SS-(d) in Fig. 2.5 generated by the SfM is the example of depth maps for evaluation.

2.4.1.2 NYU-Depth-v2 dataset

The NYUv2 dataset [134] comprises RGB images and depth maps collected from Microsoft Kinect in 464 indoor scenes. The raw training dataset contains 268K images. We downsampled 10 times to remove redundant frames, resulting in 47k frames sampled for the training set. Our method is tested on the 654 official labeled test set for evaluation.

2.4.1.3 KITTI depth completion dataset

The KITTI depth completion dataset [115] provides RGB images, sparse lidar points, and dense ground truth depth. Following the official split, it contains 86k frames for the training set, 1k frames for the validation set, and 1k frames for the evaluation.

2.4.2 Implementation Details

We adopt ResNet34 [135] as the RGB encoder, and it was pretrained on ImageNet [56]. The proposed deep networks were implemented with PyTorch [136] and trained on 4 Tesla V100 GPUs. We trained our network from scratch for 30 epochs, with a batch size of 16 using Adam [137], where $\beta_1 = 0.9$, $\beta_2 = 0.999$; we used an input resolution of 1024×760 for NAVERLABS, 640×480 for NYUv2, and 1024×320 for KITTI. We used an initial learning rate of 10^{-4} for the first 10 epochs and halved it every 10 epochs. In loss functions, we set the α to 0.85, λ_d to 0.001 and λ_s to 0.1.

2.4.3 Experiments on NAVERLABS indoor datasets

We compare our proposed method with 4 baseline approaches with public codes: 1) UNET is a U-shaped ResNet [26]; 2) nUNET is an early-fusion encoder-decoder network combined with multiple

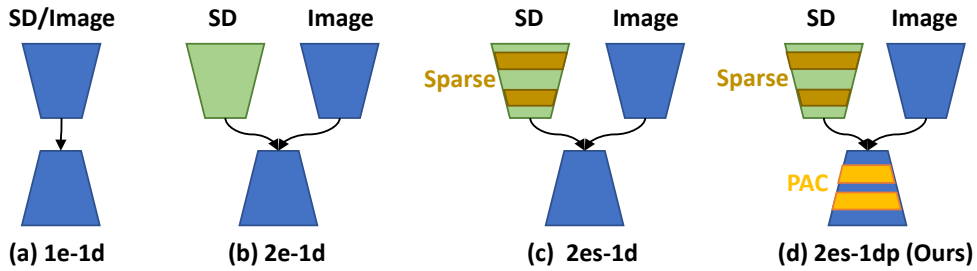


Figure 2.6: SD represents a sparse depth. We design 4 architectures: (a) 1e-1d: an encoder and a decoder; (b) 2e-1d: an RGB encoder, a depth encoder, and a decoder; (c) 2es-1d: an RGB encoder, a depth encoder with sparse convolutions, and a decoder; (d) 2es-1dp: the same architecture as Fig. 2.4.

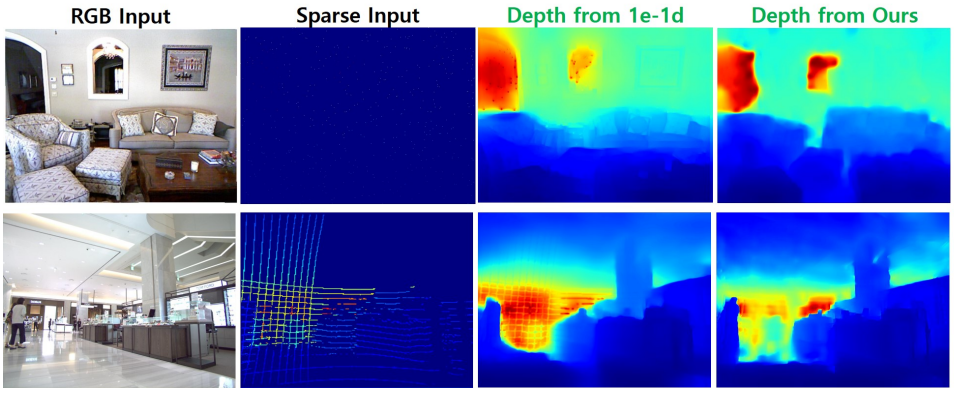


Figure 2.7: Examples of depth artifacts generated by 1e-1d model. The top (NYUv2) and bottom (Dept. 1F) figures in third column shows irregular sparse patterns copied from sparse inputs and blurred depth boundaries

normalization convolutions [27]; 3) CSPN learns the affinity to perform sparse depth propagation [28]; 4) Monodepth2 (MD2) is based on ResNet34 [76]. The same relative pose generated from our method is applied to training MD2 for a fair comparison. MD2 benefits from the use of ground truth median scaling at inference, while other depth completion methods utilize unmodified network prediction. Except for 4), the aforementioned methods are based on supervised training with ground truth depth. As mentioned in Section 2.4.1, it is difficult to create ground truth depth for the Metro Station dataset (MS). In the case of the MS dataset, we only use the sparse depth maps as weak supervisions for supervised depth completion. Quantitative results are shown in Table 2.1, and the associated qualitative comparisons are shown in Fig. 2.8. We train the model from Ma et al. [26] with its network architecture

Table 2.1: Depth completion results on NAVERLABS indoor dataset. "S" represents *supervised* training methods and "SS" denotes *self-supervised* training methods. Different from other depth completion methods, MD2 is based on self-supervised depth estimation with monocular video sequences.

(a) Dept. 1F Dataset

Method	T	RMSE ↓	ABS Rel ↓	$\delta_{1.25}$ ↑	$\delta_{1.25^2}$ ↑	$\delta_{1.25^3}$ ↑
UNET [26]	S	5043.5	0.571	0.273	0.463	0.553
nUNET [27]	S	4993.8	0.171	0.898	0.933	0.951
CSPN [28]	S	2857.1	0.266	0.808	0.886	0.924
MD2 [26]	SS	3862.4	0.493	0.488	0.716	0.818
1e-1d	SS	2899.6	0.256	0.784	0.869	0.911
2e-1d	SS	2801.4	0.265	0.783	0.868	0.909
2es-1d	SS	2700.3	0.242	0.790	0.875	0.915
Ours	SS	**2692.9**	**0.234**	**0.815**	**0.887**	**0.926**

(b) Dept. B1 Dataset

Method	T	RMSE ↓	ABS Rel ↓	$\delta_{1.25}$ ↑	$\delta_{1.25^2}$ ↑	$\delta_{1.25^3}$ ↑
UNET [26]	S	3048.5	0.346	0.599	0.711	0.764
nUNET [27]	S	2963.1	0.324	0.635	0.732	0.779
CSPN [28]	S	2162.2	0.199	0.774	0.890	0.938
MD2 [26]	SS	3592.7	0.338	0.47	0.729	0.862
1e-1d	SS	2210.5	0.186	0.823	0.914	0.947
2e-1d	SS	2157.5	0.170	0.832	0.916	0.949
2es-1d	SS	2116.8	0.183	0.829	0.916	0.949
Ours	SS	**2024.1**	**0.178**	**0.841**	**0.922**	**0.953**

(c) Metro Station Dataset

Method	T	RMSE ↓	ABS Rel ↓	$\delta_{1.25}$ ↑	$\delta_{1.25^2}$ ↑	$\delta_{1.25^3}$ ↑
UNET [26]	S	4029.7	0.749	0.201	0.219	0.23
nUNET [27]	S	2705.1	0.472	0.444	0.528	0.582
MD2 [26]	SS	1321.2	0.166	0.8	0.916	0.939
Ours	SS	**1248.4**	**0.138**	**0.866**	**0.923**	**0.942**

and the relative poses from our method, which are more accurate. Note that Ma et al. failed to produce meaningful results. We observe that the network just copies the sparse inputs, meaning that the sparse depth loss only serves as supervision for training while the photometric loss is unreliable.

In contrast, our method successfully performs depth completion in the NAVLERLABS dataset. This indicates that our training framework is more robust to variant supervision compared to other self-supervised depth completion methods. The performance of our method is superior to existing supervised completion methods. From columns 4 through 6 of Fig. 2.8, we can observe that supervised

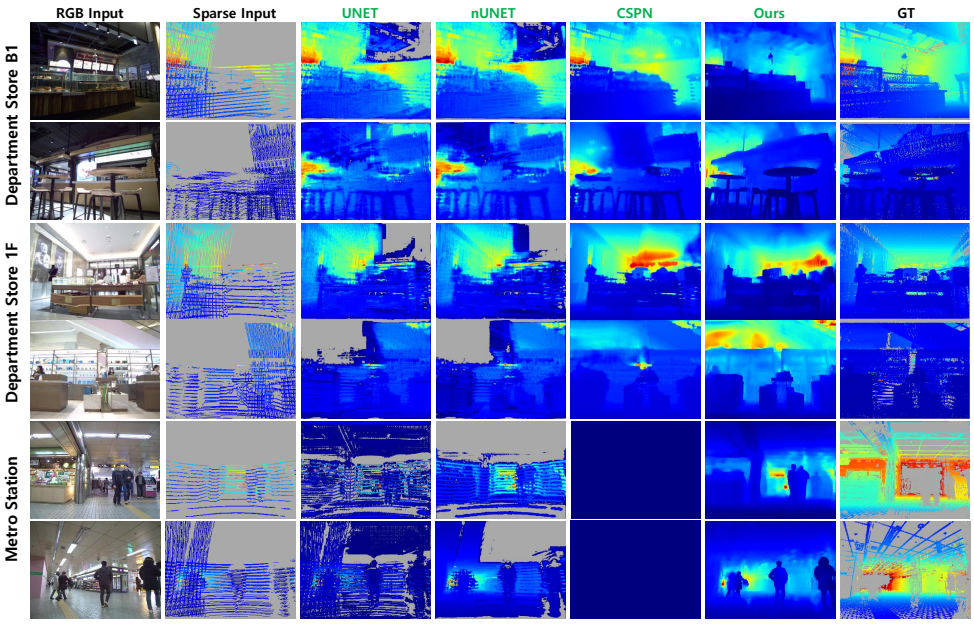


Figure 2.8: Qualitative comparison with "UNET" [26], "nUNET" [27], and "CSPN" [28] on the NAVERLABS indoor dataset. As mentioned in Sec. 2.4.1.1, the groundtruth of the Metro Station dataset is the results of MVS. (low high; grey means empty depth values.)

depth completion methods rely heavily on the density of ground truth depth. We notice that these algorithms fail to fill the holes where LiDAR information does not exist (e.g., the upper part of the image) due to the sensor configuration. In contrast, thanks to the effectiveness of photometric loss, the proposed method succeed in filling empty regions such as the ceiling, which LiDAR sensors are not able to measure, with accurate depth values. In addition, our method even predicts the correct depth map on shiny or transparent surface which cannot be scanned accurately by sensors (see First row in Fig. 2.8). The experimental results of CSPN bolster this observation because our method achieves performance comparable to the supervised setting in Table 2.1-(a), (b), while its training with weak supervision collapsed on the MS dataset. If we collect the high-quality dense depth maps (see the last column in Fig. 2.8), the supervised depth completion methods can be superior to our method. However, collecting dense depth maps requires a high cost and can be impractical in certain environments such

Table 2.2: depth completion results on the NYUv2 dataset "S" represents *supervised* training methods and "SS" denotes *self-supervised* training methods.

Method	T	RMSE ↓	ABS Rel ↓	$\delta_{1.25}$ ↑	$\delta_{1.25^2}$ ↑	$\delta_{1.25^3}$ ↑
(a) 500 Samples						
Ma et al. [94]	S	0.204	0.043	97.8	99.6	99.9
CSPN [28]	S	0.117	0.016	99.2	99.9	100
Ma et al. [26]	SS	0.271	0.068	-	-	-
Ours	SS	0.178	0.033	98.1	99.7	100
(b) 200 Samples						
Ma et al. [94]	S	0.230	0.044	97.1	99.4	99.8
Yang et al. [95]	SS	0.569	0.171	-	-	-
Yoon et al. [96]	SS	0.309	-	-	-	-
Ours	SS	0.240	0.048	96.6	99.4	99.9

Table 2.3: depth completion results on the KITTI validation set. "S" represents *supervised* training methods and "SS" denotes *self-supervised* training methods.

Method	T	RMSE [mm]	MAE [mm]	iRMSE [1/km]	iMAE [1/km]
Uhrig et al. [115]	S	1601.33	481.27	4.94	1.78
Ma et al. [26]	S	814.73	249.95	2.80	1.21
Yoon et al. [96]	SS	1593.37	547.00	27.98	2.36
Ma et al. [26]	SS	1343.33	358.66	4.28	1.64
Yang et al. [95]	SS	1310.03	347.17	-	-
Ours	SS	1212.89	346.12	3.54	1.29

as the MS dataset.

To investigate the impact of our network architecture, we conduct ablation studies for different network configurations on the NAVERLABS dataset. Figure 2.6 illustrates the network variants, and their quantitative results are shown in Table 2.1. Comparing the results of 1e-1d with those of 2e-1d, we observe that fusing the RGB and depth feature at the earliest stage worsens the results. With the early fusion, we often observe depth with blurrier depth discontinuities, as seen in Fig. 2.7. With PAC, depth maps can eliminate either irregular lattice patterns from LiDAR or sparse points from sampled depth maps. We conclude that the late fusion is more effective for self-supervised learning because image and depth features are heterogeneous data, and early fusion is more likely to produce noisy

initial depth values.

2.4.4 Experiments on Public Benchmark Datasets

To verify the performance of our method on public datasets, we train and evaluate our method on the NYUv2 dataset. Following previous works, we uniformly sample 200 and 500 sparse depth points separately for sparse input. Except for supervised depth completion methods, our proposed method achieves the best performance among self-supervised depth completion methods in Table 2.2.

In order to prove the generalization capability of our method, we train and evaluate our method on an outdoor dataset, the KITTI depth completion dataset. The comparison results are shown in Table 2.3. We observe that our method outperforms other self-supervised depth completion methods on every metric. Although Yang [95] pretrains their network using a virtual dataset such as the Virtual KITTI dataset [138], which is similar to the benchmark dataset, our method performs better than their method. We include qualitative results in the supplementary video.

2.5 Conclusion

In this paper, we investigated the self-supervised depth completion framework for a challenging indoor environment. We provide two main research contributions. First, we introduced a training framework that leads to stable self-supervised training. Second, we proposed a novel architecture to combine image features and sparse depth features effectively. In indoor environments, our method outperforms other self-supervised methods and is competitive with supervised methods. Furthermore, it generalizes better to all public benchmark datasets. As part of future work, we would like to apply this method to complement depth sensors for mobile robots in indoor environments and and evaluate

its benefits for robot navigation. A limitation of the current method is the high computational cost for embedded platforms. For robotic tasks, we would like to explore network compression algorithms to reduce computational complexity and latency.

Chapter 3: DnD: Dense Depth Estimation in Crowded Dynamic Indoor Scenes

3.1 Introduction

There is considerable interest in using robotics and augmented reality technologies in crowded real-world spaces corresponding to malls, airports, or public places. In order to perform safe navigation or combine real and virtual worlds, robots [59,139,140] or mobile devices [60,61] need a 3D geometric representation of large-scale indoor environments. While there is considerable progress in terms of capturing depth using LiDARs or stereo cameras, existing devices still have their own limitations. For example, 3D LiDARs [90] tend to produce sparse depth maps for distant objects and may result in noisy point cloud maps due to the high level of occlusions caused by multiple moving people. Moreover, because of the high prices and large volumes of the depth sensors, there is a critical need to consider the case where only a single camera is available.

Given a large number of video frames, traditional 3D reconstruction methods such as structure-from-motion (SfM) and multi-view stereo (MVS) [29,30,141,142] can generate a 3D model. Recently, visual localization techniques [143] have been developed to allow mobile devices to obtain the location and camera pose from which the image is taken. Given the 3D model and visual position, however, mobile devices moving through crowded indoor environments are only able to capture sparse and highly noisy depth maps. This is because traditional reconstruction methods are based on both a static scene assumption (that static areas of the scenes can be observed from two different viewpoints) and correct feature matching in two or more images. However, the moving pedestrians in crowded indoor environments tend to violate the static scenes assumption. Moreover, traditional 3D reconstruction methods may fail to perform the correct matching on non-textured (e.g., walls), specular, and reflective regions (e.g., glass) of the scene. Consequently, these problems lead to a 3D model on complex and

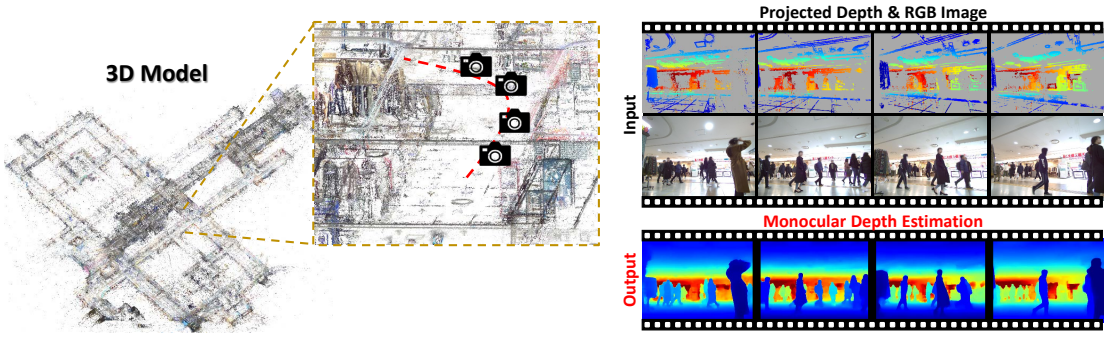


Figure 3.1: With traditional 3D reconstruction methods [29,30], we can obtain a 3D model on complex and crowded indoor environments. Our novel approach, DnD, takes both an RGB image and a sparse depth map projected from this 3D model as input. Our method predicts dense and absolute-scale depth maps of single view dynamic scenes.

crowded indoor environments.

Main Results: To address these limitations, we explore new methods that can utilize the 3D model generated using traditional 3D reconstruction methods [29, 30] into learning-based depth estimation algorithms for dynamic scenes. Given the 3D model, our method can be used for general applications because it can compute dense depth maps of dynamic scenes without reconstructing this 3D model iteratively. In contrast to supervised learning methods [32, 69, 144], our method does not rely on dense depth maps generated from depth sensing devices. Our approach, shown in Fig. 3.1, takes an RGB image and a sparse depth map projected from the 3D model as input and outputs a dense depth map. Given the pose obtained from the SfM [29], we propose using the photometric consistency loss, which enables our method to estimate dense depth maps, and depth loss, forcing our network to learn absolute-scale depth. Although these loss functions are useful for providing dense depth maps in static background regions, there still exist great challenges in estimating depths of multiple non-rigidly moving objects, i.e. the pedestrians. To overcome this limitation, we propose two constraints: 1) a flow-guided shape constraint to refine the depth maps for human regions by filling missing parts of the human regions and removing visual artifacts, and 2) a normal-guided scale constraint to force

our neural network to learn the absolute scale depth in human regions guided by depth values in the human's ground contact point.

Compared to traditional reconstruction approaches or recent learning-based methods [31–33], our approach (DnD) shows a better ability to predict plausible depth in both human and non-human regions, though we only use a monocular camera. We evaluate our approach on the NAVERLABS dataset [133], the first dataset that provides both metric 3D SfM models and dynamic scenes collected from a department store and a metro station. The main contributions of this paper are summarized as follows:

- We introduce a novel approach to estimate the depth maps using both dynamic scenes collected from a moving monocular camera and given sparse depth maps. We train the monocular depth estimation network with a 3D model generated by traditional 3D reconstruction algorithms [29, 30].
- We present two novel constraints based on optical flow and surface normal that improve the accuracy of our monocular depth estimation network to predict absolute scale depths for moving people.
- We highlight the benefits of our approach over the state-of-the-art methods on crowded indoor environments and observe 3.6% -10.2% improvement in RMSE. Furthermore, our method works well in diverse indoor datasets like TUM RGB-D and NYUv2.

3.2 Related Work

Structure from Motion and Multi-view Stereo Traditional SfM systems [29,142,145] rely on matching features across two or more images of the same scene and using epipolar geometry [97] to recon-

struct depth. Multi-view stereo (MVS) algorithms estimate the dense 3D structure of a scene with multiple calibrated images from the arbitrary viewpoints [141]. Recently, some researchers have presented learning-based MVS methods [31, 146] that build on a neural network to learn the regularization of 3D cost volume. However, traditional SfM and MVS methods often produce sparse and erroneous 3D reconstruction due to incorrect feature matches and dynamic objects [147]. Due to static scene assumption, they either drop pixels with low confidence or estimate incorrect depth values for dynamic objects.

Monocular Depth Estimation The existing learning-based depth estimation methods fall into two groups. One group is based on supervised learning [33, 72, 73, 148–151], which requires a large-scale dataset with groundtruth depth maps. However, all of these methods require dense groundtruth collected from active depth sensors. Other recent works explore the idea of training with weak supervisions, e.g., using ordinal depth relations as groundtruth [71, 152–154] and leveraging multi-view stereo reconstruction algorithms to generate pseudo groundtruth from internet photo collections [69, 144]. The other group is based on self-supervised learning using either rectified stereo image pairs [110, 155] or monocular video sequences [75, 76, 156–159] as training data. Video-based depth estimation methods [160–162] use temporally consecutive frames to estimate depth over time during inference under the assumption of handling only static scenes. Several works for monocular depth completion [19, 26, 27, 95, 120] have been proposed to capitalize on sparse depth maps with corresponding images, resulting in dense depth estimations.

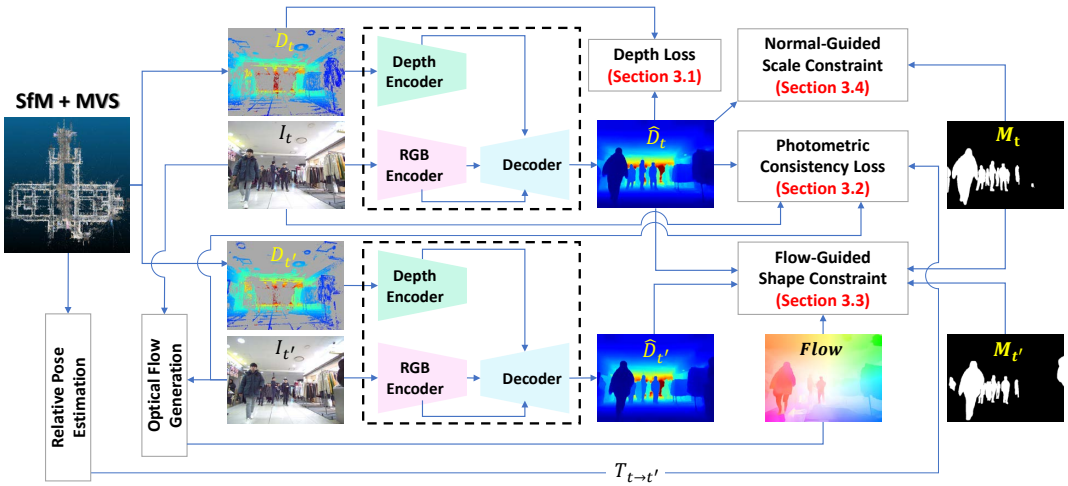


Figure 3.2: Overview of our proposed approach. The black box with a dotted line shows our monocular depth estimation network consisting of an RGB encoder, a depth encoder, and a decoder. For training, the network takes consecutive temporal frames (I_t, $I_{t'}$) with corresponding sparse depth maps (D_t, $D_{t'}$) projected from the 3D model as input. Our training method is a combination of four terms. 1) The depth loss encourages our network to encode absolute scale from a depth input. 2) The photometric consistency loss is based on view synthesis and regularizes the network training for static background regions. 3) The flow-guided shape constraint enables the network to complete missing pixels in human regions with proper depth values and eliminate visual artifacts. 4) The normal-guided scale constraint enables our network to estimate accurate and absolute-scale depths on moving people.

3.3 Method

We present a learning-based approach to estimate dense and absolute-scale depth maps in scenes from complex and crowded indoor environments. Intuitively, most scenes consist of a static background (e.g., a wall) and dynamic objects (e.g., moving people). The sparse depth maps projected from the 3D model have absolute-scale depth values in small regions of the static background. The rest of the regions in the projected depth maps, including the static background and dynamic objects, have empty depth values. Our proposed approach trains a dense depth estimation network from monocular video sequences with the corresponding sparse depth maps. As shown in Fig. 3.2, our depth estimation model takes a current image I_t with a sparse depth map D_t and temporally adjacent images $I_{t'}$ with

sparse depth maps $D_{t'}$. The sparse depth maps D_t and $D_{t'}$ are projected from the 3D model. The $I_{t'}$ includes two temporal frames I_{t-1} and I_{t+1}. Our depth estimation model predicts dense depth maps $\hat{D}_t$ and $\hat{D}_{t'}$ to compute a flow-guided shape constraint.

3.3.1 Absolute-Scale Depth Loss

Generally, monocular depth estimation methods suffer from an inherent scale ambiguity problem. To mitigate this limitation, we use an absolute-scale depth input D_t as a groundtruth. We apply the L1 loss to penalize the differences between the depth input D_t and the depth prediction $\hat{D}_t$ on the pixels where D_t values exist. The depth loss is formulated as,

$$L_d = \sum_{p \in \Omega} \mid \hat{D}_t(p) - D_t(p) \mid, \tag{3.1}$$

where Ω indicates valid points that involve available sparse depths. This loss enables our network to learn absolute-scale depth values in the regions of the depth input and further extrapolate depth values with absolute scales in empty regions including a static background and moving people.

3.3.2 Photometric Consistency Loss

To train our network without groundtruth, we turn to previous self-supervised monocular depth estimation methods [75,76] that have utilized the photometric loss as the main loss function for training the network. We are able to generate a synthesized frame I'_t by reprojecting temporally adjacent frames $I_{t'}$ to the current frame I_t with the camera intrinsic matrix K, the predicted depth $\hat{D}_t$, and the relative pose $T_{t \to t'}$. This relative pose between consecutive temporal frames is computed from the absolute poses from SfM [29]. The photometric consistency loss with a combination of L1 and SSIM [163] is

formulated as follows:

$$L_{ph} = \alpha \frac{1 - \text{SSIM}(I_t, I'_t)}{2} + (1 - \alpha) \parallel I_t - I'_t \parallel,$$
$$I'_t(p) = I_{t'} \langle \pi(K T_{t \to t'} \hat{D}_t(p) K^{-1} \tilde{p}) \rangle, \tag{3.2}$$

where $\alpha = 0.85$, $\tilde{p}$ is the homogeneous coordinate of p, π means projection from homogeneous to image coordinates, and $\langle \cdot \rangle$ indicates the bilinear sampling function. Note that one current frame and two temporally adjacent frames are used to compute the photometric consistency loss. Following [76], we remove the occluded and out-of-view pixels, and also mask out low texture regions via minimum reprojection loss and auto-masking techniques. In particular, this photometric consistency loss encourages our network to predict dense depth maps over the static backgrounds.

3.3.3 Flow-Guided Shape Constraint

The loss functions described in the previous sections are still limited in their ability to provide accurate depths when there are moving people. This is because both MVS and photometric consistency loss operate under the assumption of a single moving camera and a static scene. This assumption implies that inconsistencies between different views of each image are only derived from the camera ego-motion in static scenes [97]. If we correctly estimate the motion of moving people between the consecutive frames, the view consistency for 3D reconstruction can be obtained for two different views. However, with multiple moving people in dynamic scenes, triangulation-based methods fail to achieve a scene consistent depth map. Since multiple moving people in scenes undergo non-rigid deformations, it is difficult to explicitly estimate their 3D motion [32, 164, 165].

Instead of modeling object motions in 3D, we leverage monocular video frames to estimate temporally coherent depth maps with respect to moving camera motions and non-rigidly moving people. To achieve this, we use optical flow F, a human mask M_t from the current frame I_t, and human masks $M_{t'}$ from the temporally adjacent frames $I_{t'}$. Given consecutive frame pairs, the optical flow describes which pixel pairs have the same intensities. Human masks are also used to find correct regions that overlap due to the high-level of occlusion, non-rigid deformation, and complex ego-motions of the moving camera.

$$M = M_t \cap F_{t' \to t}(M_{t'}), \tag{3.3}$$

where M is the overlapped region of the flow-warped human mask $M_{t'}$ and the current human mask M_t to prevent occlusion issues, meaning that some pixels from M_t do not appear in $F_{t' \to t}(M_{t'})$. In dynamic scenes captured by the moving camera, the absolute depth values for dynamic human regions can be inconsistent. Thus, we apply the optical flow field to compare consecutive inverse depth predictions for human regions in gradient domains. Our proposed flow-guided shape constraint is defined as

$$\begin{aligned} \overline{\nabla}(d^*(p)) &= \frac{\nabla d^*(p)}{\mid d^*(p) + \nabla d^*(p) \mid + \mid d^*(p) \mid}, \\ L_f &= \frac{1}{\mid M \mid} \sum_{p \in M} \mid \overline{\nabla} d_t^*(p) - \overline{\nabla} F_{t' \to t}(d_{t'}^*(p)) \mid \end{aligned} \tag{3.4}$$

where ∇ and $\overline{\nabla}$ denote the gradient and scale-invariant gradient and $d_t^* = d_t / \mu(d_t)$ indicates the mean-normalized inverse depth. Our flow-guided shape constraint enforces smooth gradients and completes missing parts of human regions with accurate depth values. To regularize the depth in a static back-

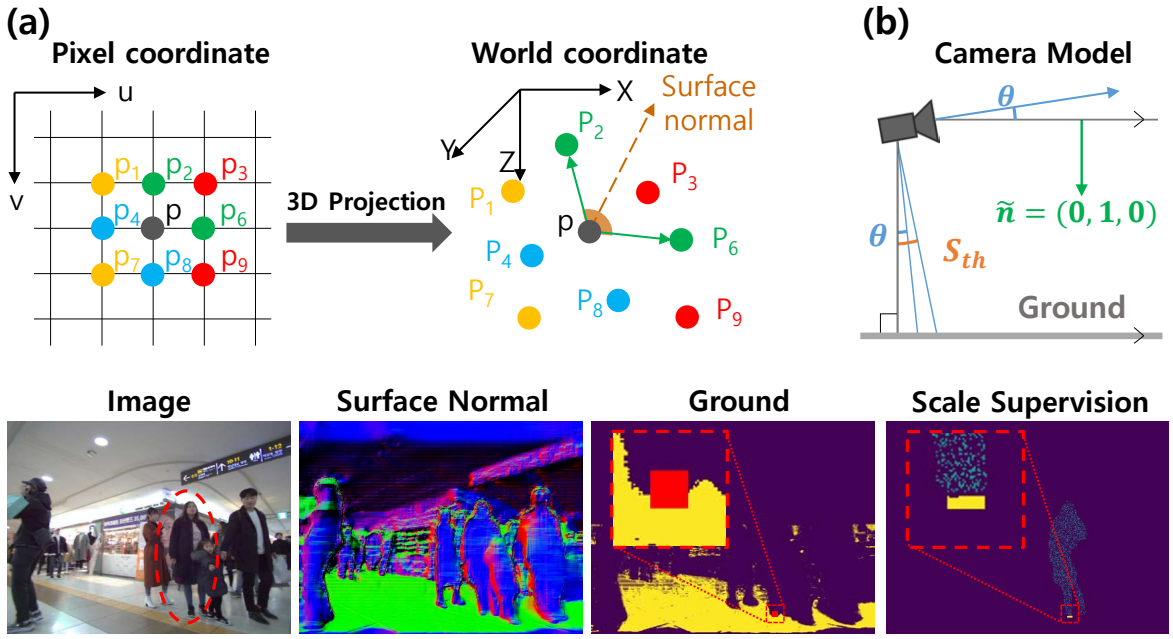


Figure 3.3: (a) The process to obtain a surface normal from the pixel coordinates and the corresponding depth values. (b) shows that the ground normal $\tilde{n}$ is not aligned with the y-direction of the camera coordinates. The bottom images indicate the process for propagating the reliable scale for the person in the dashed red circle. The red point in the third column means the small patch B, indicating the human's ground contact point. In the fourth column, the yellow and blue points show the ground-patch intersection and uniformly sampled points in the human mask respectively.

ground, we impose the edge-aware depth smoothness loss L_s over pixels in non-human regions M^C:

$$L_s = \sum_{p \in M^C} \mid \nabla d_t^*(p) \mid e^{-|\nabla I_t(p)|}. \tag{3.5}$$

3.3.4 Normal-Guided Scale Constraint

To achieve scale-aware depth estimation, accurate depth estimation with an absolute scale for moving people is crucial. We leverage absolute-scale depth values in the static background to constrain the scale of depth values for moving people. Intuitively, people are standing on the ground and the depth values of the humans are nearly the same as the ones on the ground. The human's ground contact

point can be a significant geometric cue for scale-aware depth estimation. Thus, we introduce weak supervision, which forces several randomly selected points corresponding to the particular human mask to be consistent with the ground contact points' depth values.

Inspired by [158, 159], we estimate the ground area using the surface normal. The 8-neighbors convention is used to specify the normal direction, as seen in Fig. 3.3 (a). Those 8 points at the 2D coordinate are split into 4 pairs, where each pair of vectors is perpendicular, e.g., $(i, j) = \{(2, 6), (3, 9), \cdots\}$. The average of the four normals determines the final surface normal for the position p:

$$N_t(p) = \frac{1}{4} \sum_{\overrightarrow{pp_i} \perp \overrightarrow{pp_j}} \frac{\overrightarrow{PP_i} \times \overrightarrow{PP_j}}{\| \overrightarrow{PP_i} \times \overrightarrow{PP_j} \|_2}, \tag{3.6}$$

where P means the reconstructed 3D point clouds for pixel p by the predicted depth and the operator $\times$ denotes the cross product. Among many pixel points p, we only need points in ground regions. The true normal direction of ground $\tilde{n} = (0, 1, 0)$ and the estimated normal $N_t(p)$ should be matched. As in Fig. 3.3 (b), the camera is not perpendicular to the ground by a difference of θ for the camera model, and there exists the uncertainty of the estimated normal $N_t(p)$. Thus, we set S_{th} as a threshold to find the ground area and calculate the cosine similarity between the true ground normal $\tilde{n}$ and the predicted normal $N_t(p)$:

$$G = \{p \mid \mid cos^{-1} \frac{\tilde{n} \cdot N_t(p)}{\| \tilde{n} \| \| N_t(p) \|} \mid < S_{th}\}, \tag{3.7}$$

where the operator $\cdot$ denotes the inner product. Then, in order to determine the human's ground contact point, we detect small patches B centered at the bottom of the pixel in the human instances. These pixels in both the small patches and the ground area are possible candidates for the human's ground

contact point. However, simply applying the constant value as supervision risks severe degradation of depth values in the human region because moving people have exquisite and isometric shapes. Therefore, we uniformly sample the pixels M' in the human mask M_t and constrain them by the median value in the intersection of B and G:

$$L_n = \sum_{p \in M'} \frac{\mid \hat{D}_t(p) - med(\hat{D}_t(B \cap G)) \mid}{\hat{D}(p)}, \tag{3.8}$$

where the operator med takes the median value of factors.

3.3.5 Overall Loss Function

Our overall loss function is a weighted sum over the previously introduced losses,

$$L_{total} = \lambda_d L_d + \lambda_p L_{ph} + \lambda_s L_s + \lambda_f L_f + \lambda_n L_n \tag{3.9}$$

where $\lambda_d, \lambda_p, \lambda_s, \lambda_f$ and λ_n denote weights on the respective loss terms selected through a grid search.

3.4 Experiments

To validate our method, we experiment with the NAVERLABS Dataset, NYUv2 and TUM RGB-D dataset. The depth value is represented in the metric scale (m). We evaluate our method using the standard metrics [148].

NAVERLABS Indoor Localization: The NAVERLABS dataset [1] [166] comprises scenes collected from two different places: Department Store (Dept) and Metro Station (MS). This dataset is split into

[1] is available at https://challenge.naverlabs.com/

Table 3.1: Quantitative comparisons with the state-of-the-art depth estimation algorithms. F means the evaluation results in the human regions and F+B indicates the evaluation results on depth values over the entire scene. MS and Dept denote the Metro Station dataset and the Department Store dataset, respectively. DnD (L_{ph} only) is trained only using the photometric consistency loss. In the MS dataset, DnD shows improvement in depth for human regions by 3.6 % in the RMSE metric. In terms of depth for entire regions, DnD reduces the RMSE by 10.2 %.

Method	Data	Scaling	F+B / F (Lower is better)				F+B / F (Higher is better)		
			Abs Rel	Sq Rel	RMSE	RMSE log	$\delta_{1.25}$	$\delta_{1.25^2}$	$\delta_{1.25^3}$
Fast-MVSNet [31]	MS	-	0.383 / 0.754	0.60 / 1.37	2.05/ 2.95	0.123/ 0.213	0.634 / 0.358	0.787 / 0.547	0.866 / 0.697
DepthComple [26]	MS	-	0.800 / 0.863	0.7 / 0.79	4.52 / 4.67	– / –	0.041 / 0.024	0.087 / 0.050	0.139 / 0.083
MiDaS [74]	MS	median	0.413 / 0.459	0.33 / 0.42	2.77 / 2.86	0.181 / 0.191	0.380 / 0.352	0.627 / 0.604	0.785 / 0.773
MC [32]	MS	median	0.294 / 0.315	0.30 / 0.22	2.05 / 2.53	0.122 / 0.162	0.585 / 0.459	0.822 / 0.741	0.906 / 0.847
BTS [33]	MS	-	0.396 / 0.696	0.66 / 1.24	1.96 / 2.70	0.120 / 0.198	0.665 / 0.401	0.792 / 0.589	0.862 / 0.726
DnD (L_{ph} only)	MS	median	0.326 / 0.596	1.38 / 4.16	2.39 / 3.60	0.107 / 0.177	0.711 / 0.545	0.840 / 0.705	0.898 / 0.795
DnD	MS	-	**0.189 / 0.240**	**0.20 / 0.16**	**1.76 / 2.44**	**0.084 / 0.133**	**0.806 / 0.677**	**0.881 / 0.798**	**0.919 / 0.856**
Fast-MVSNet [31]	Dept	-	0.551 / 0.889	0.98 / 1.97	3.24 / 4.11	– / 0.243	0.431 / 0.288	0.655 / 0.498	0.791 / 0.654
DepthComple [26]	Dept	-	0.842 / 0.897	0.76 / 0.84	5.88 / 6.65	– / –	0.014 / 0.058	0.028 / 0.028	0.093 / 0.046
MiDaS [74]	Dept	median	0.461 / 0.519	0.46 / 0.54	4.39 / 3.72	0.208 / 0.210	0.375 / 0.332	0.579 / 0.554	0.729 / 0.728
MC [32]	Dept	median	0.428 / 0.385	0.40 / **0.29**	3.91 / 3.36	0.190 / 0.178	0.364 / 0.384	0.626 / 0.676	0.786 / 0.822
BTS [33]	Dept	-	0.584 / 1.066	1.36 / 2.80	3.06 / 4.40	0.159 / 0.260	0.561 / 0.327	0.721 / 0.488	0.808 / 0.614
DnD (L_{ph} only)	Dept	median	0.289 / 0.388	0.47 / 0.60	2.60 / 3.24	0.109 / 0.148	0.663 / 0.564	0.837 / 0.759	0.911 / 0.847
DnD	Dept	-	**0.213 / 0.250**	**0.32** / 0.30	**2.36 / 3.04**	**0.084 / 0.116**	**0.761 / 0.707**	**0.889 / 0.836**	**0.932 / 0.886**

60K images for training and 835 images for testing in MS, and 25K images for training and 443 images for testing in Dept. To evaluate the crowdedness of indoor datasets, we use a crowd density, which is a ratio between the area of human pixels in scenes and the area of all image pixels. Most public indoor datasets [7, 44, 134, 167–169] do not have scenes with moving people ($<$ 0.1% crowd density). In NAVERLABS, Dept and MS have 6.87% and 12.9% crowd density, the most crowded dataset. Additionally, Dept and MS have 6.7 and 3.6 people per scene on average. Different types of datasets [32, 71, 153, 164] that include diverse dynamic scenes contain only images or ordinal depth relations for pairs of points.

NAVERLABS used all the images collected from 6 cameras to build a 3D model via COLMAP [29, 30]. The groundtruth poses for the input images were initially calculated by LiDAR SLAM and further refined by bundle adjustment with prior results. Afterwards, metric depth images were generated by the COLMAP MVS algorithm. For evaluation, we accumulated 0.3 seconds of LiDAR scans

corresponding to one scan line since the LiDAR is sampled at 10 Hz. It is difficult to accumulate more LiDAR sweeps because pedestrians cause severe noise and artifacts in the projected depth maps.

TUM RGB-D: The TUM RGB-D dataset [170] is comprised of 39 video sequences recorded with a Microsoft Kinect. We used a subset of sequence, *walking*, which contains moderate dynamic scenes that two people walk both in the background and the foreground. We train our method on these subsets and evaluate on the test set following MC [32].

NYUv2: Although the NYUv2 dataset [134] does not contain dynamic scenes, it is the most common public benchmark of depth estimation for indoor scenes. The training dataset consists of 268K images. We subsampled every 10 frames and used only 47k frames for training. The official 654 test scenes are used for evaluation.

Implementation Details: Our monocular depth estimation network is based on U-Net [171], an encoder-decoder architecture. For training, we apply FlowNet2.0 [172] to compute optical flow and Mask R-CNN [173] pre-trained on the COCO dataset [174] to detect human masks. For normal-guided scale constraint, we apply 20×20 boxes to compose the small patch B. The camera of the NAVER-LABS dataset is towards the upper side for $\theta = 10^{\circ}$ as in Fig. 3.3 (b); thus S_{th} is set to 15°. We investigate the sampling ratio for M' in Eq. 3.8 at 10%, 30%, and 50%, and we select 30% because it is slightly better than others. We use the following set of weights for each loss term in the loss function: λ_d = 0.001, λ_{ph} = 1, λ_s = 0.3, λ_f = 0.1, and λn = 0.001. In the supplementary material, we provide more implementation details of network architecture and training procedures.

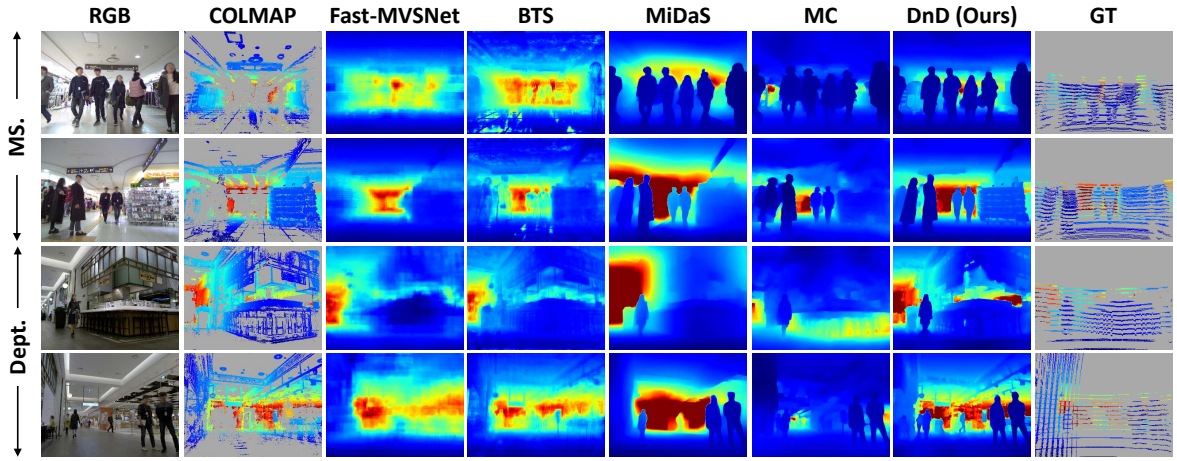


Figure 3.4: Qualitative comparisons of the Metro Station dataset (1-2 rows) and the Department Store dataset (3-4 rows). Our network takes an image (1st column) and a depth input projected from the 3D model (2nd column), and predicts dense depth maps (7th column). Ground truth (8th column) collected from 3D LiDAR is used to evaluate our method. Depth results (3-6 columns) are estimated by the existing depth estimation algorithms. Compared to MiDaS and MC, DnD generates detailed and high quality depth maps in non-human regions and also shows depth maps in human regions with sharp depth discontinuities. All color maps use the jet color map (low high; grey means empty depth values).

3.4.1 Experimental Results on NAVERLABS dataset

For quantitative evaluation, we compare our proposed method DnD with five different methods: 1) Fast-MVSNet [31] is a learning-based multiview stereo algorithm, 2) DepthComple is a learning-based depth completion algorithm that adopts an early-fusion encoder-decoder network from [26] combined with normalized convolution layers [27], 3) MiDaS [74] is a monocular depth estimation method trained on a diverse set of datasets including various dynamic scenes, 4) MC [32] is a monocular depth estimation method trained on a Mannequin Challenge dataset, and 5) BTS [33] is the state-of-the-art supervised monocular depth estimation method on the benchmark datasets [134]. To experiment with Fast-MVSNet, DepthComple, and BTS, we use their public codes and train the network from scratch with the NAVERLABS dataset. MiDaS and MC are implemented with pre-trained weights from their

Table 3.2: Contributions of our proposed modules to the evaluation results on the Metro Station (MS) dataset. F means the evaluation results on depth values in the human regions and F+B indicates the evaluation results on depth values over the entire scene. The median scaling is applied to DnD (L_{ph} only) for absolute scale depth prediction. Compared to DnD (L_{ph} only), we improve the RMSE by 26.4% in the entire scene and by 32.2% in the human regions.

Method	F+B / F (Lower is better)				F+B / F (Higher is better)		
	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta_{1.25}$	$\delta_{1.25^2}$	$\delta_{1.25^3}$
DnD (L_{ph} only)	0.326 / 0.596	1.38 / 4.16	2.39 / 3.60	0.107 / 0.177	0.711 / 0.545	0.840 / 0.705	0.898 / 0.795
DnD w/o FSC, NSC	0.209 / 0.304	0.24 / 0.25	1.80 / 2.56	0.095 / 0.172	0.767 / 0.540	0.851 / 0.690	0.899 / 0.787
DnD w/o NSC	0.204 / 0.299	0.23 / 0.24	1.78 / 2.51	0.094 / 0.170	0.772 / 0.552	0.856 / 0.701	0.901 / 0.790
DnD w/o FSC	0.192 / 0.252	**0.20** / 0.17	1.77 / 2.47	0.086 / 0.139	0.794 / 0.644	0.877 / 0.785	0.917 / 0.851
DnD (full)	**0.189 / 0.240**	**0.20 / 0.16**	**1.76 / 2.44**	**0.084 / 0.133**	**0.806 / 0.677**	**0.881 / 0.798**	**0.919 / 0.856**

public codes. To train the former methods, we exploit a depth map projected from the 3D model as groundtruth.

In our experiments, training video-based depth estimation methods [161, 162] fails in the video frames of NAVERLABS. Their methods based on static scene assumption cannot be applied for scenes with extreme object motion. Furthermore, self-supervised methods [26,76] show poor performance for depth evaluation. We observe that a joint training framework of pose and depth from a monocular video is extremely difficult because pose networks often fail to estimate proper camera ego-motion in complex and crowded indoor environments. Instead, we provide experimental results of DnD (L_{ph} only) trained only with the photometric consistency loss similar to self-supervised training settings, e.g., Monodepth2 [76]. DnD (L_{ph} only) only converts an RGB image to a dense depth map and thus suffers from the scale ambiguity issue. DnD, Fast-MVSNet, DepthComple, and BTS are able to estimate absolute depth values while MiDaS and MC only output relative depth up to an unknown scale factor. Thus, we apply a median scaling [76] to define a scale factor by comparing their depth predictions with the absolute depth from our groundtruth depths. In Table 3.1, we summarize the quantitative results on two indoor datasets. DnD outperforms other depth estimation methods on all evaluation metrics. In particular, we observe an improvement in performance at depths of both people (F) and the entire scene

(F+B). Although the models provided by MiDaS and MC were trained on different datasets, the methods focus on composing datasets for dynamic scenes and training their depth estimation network with supervised learning. Compared to such methods, our method can be scalable to general environments without high quality groundtruth depths.

A qualitative comparison is shown in Fig. 3.4. The visual results of DepthComple are added in the supplementary material because this method fails to generate reasonable depth maps. In general, the qualitative comparison corresponds well to our quantitative results in Table 3.1. In terms of depth in human regions, Fast-MVSNet and BTS fail to reconstruct the depth for moving people regions. BTS is not able to learn depths for moving people regions because it only relies on COLMAP results as ground truth. Fast-MVSNet is also not able to provide depths for moving people due to the limitation of multi-view matching. In contrast, MC and MiDaS adequately estimate the depths in human regions with sharp depth discontinuities. However, these methods show blurry and noisy depths for background regions. Compared to other methods, depth predictions over background regions from our method are sharper and less noisy.

Ablation Study In Table 3.2, we validate the influence of different loss terms within our proposed method. We note that all our proposed modules contribute to a significant improvement in Table 3.2. In particular, both NSC and FSC significantly improve depth estimation performance for all evaluation metrics. Figure 3.5 provides an ablation study using qualitative comparison to better understand the effectiveness of different loss terms in our method. In Fig. 3.5 (a), only using depth loss and photometric consistency loss for training is able to estimate dense depth maps over the entire scene. However, they show poor results especially on depth in human regions. Figure 3.5 (b) and (c) validate that our method improves the depth estimation performance with each proposed module .

Analysis We study the effectiveness of the number of sparse depth points and the crowd density. We

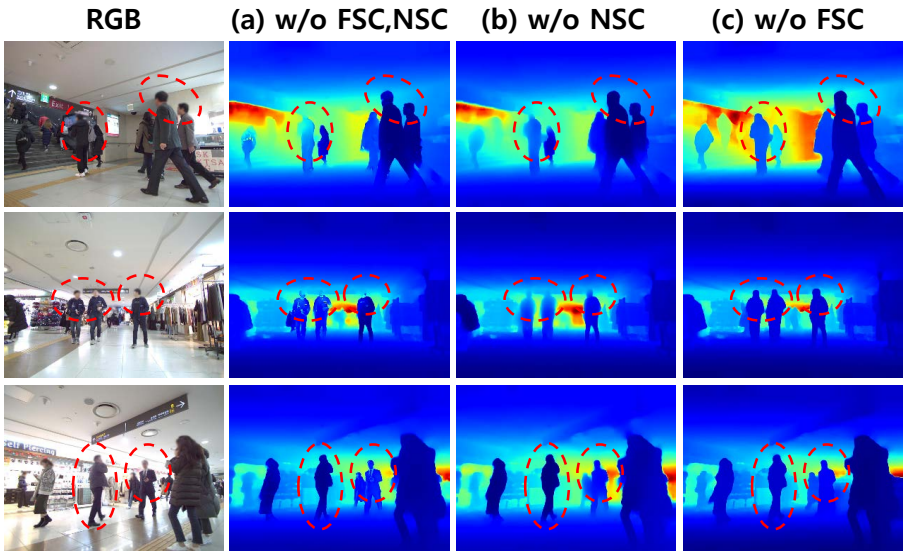


Figure 3.5: Qualitative comparisons of the proposed modules. (a) We note that a part of the human is cut off in the depth map. The visual artifacts that look similar to RGB image patterns often appear in human depth regions. (b) FSC encourages our network to capture sharper details, fill in missing parts of the human regions, and remove visual artifacts. (c) We observe that NSC estimates depth with sharp depth discontinuities and also recovers absolute scale depth values for human regions. All color maps use the jet color map (low high; grey means empty depth values).

use the accuracy score $\delta_{1.25}$ as an evaluation metric because other metrics do not show significant performance degradation in DnD. If the number of matched local features is not enough for reconstructing a plausible 3D model, the projected depth input for DnD might become a highly sparse map. In this case, the experimental results in Fig. 3.6 (a) show the relationship between the number of depth points and the performance of DnD. As the number of input depth points decreases, the performance degrades slightly as expected. However, this robustness to the sparsity proves that the RGB images serve as the primary source of our monocular depth estimation network. Furthermore, for the dense crowded scenes in indoor environments, we select some highly crowded samples in the original test set for the NAVERLABS dataset. These are composed of 12% or more human pixels out of the total number of the pixels in one image.

Note that the crowd density is a more significant factor than the number of people per scene in

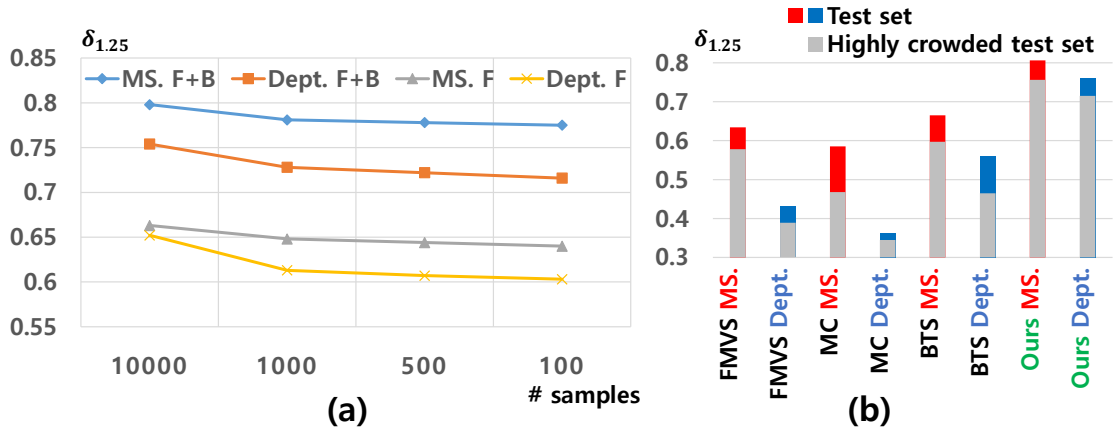


Figure 3.6: (a) shows $\delta_{1.25}$ metric for our method corresponding the number of uniform-sampled input depth points, and (b) indicates the performance degradation for scenes with high density crowds in MS and Dept: 8.8% and 9.7% for FMVS (Fast-MVSNet [31]), 20% and 5.2% for MC [32], 10.2% and 17.1% for BTS [33], 6.2% and 6% for ours.

our task. We focus on pedestrians whose apparent sizes are large (i.e., high crowd density) because they are close to the camera and occlude those who lie behind them. Despite the multiple non-rigid people causing difficulties in depth estimation, we notice that DnD robustly estimates depth for both humans and background regions as in Fig. 3.6 (b).

DnD with Localization For the complete system of DnD, there is a need for reprojected depth maps obtained from the 3D model using visual localization techniques. Thus, we exploit Kapture toolbox [18] to estimate the camera pose of given input images, and then project the 3D model to the sparse depth maps. We describe more details about the whole pipeline in the supplementary materials.

3.4.2 Results on NYUv2 and TUM RGB-D datasets

We train and evaluate our method on each NYUv2 and TUM RGB-D dataset. Since the 3D map for TUM and NYUv2 produced by COLMAP is only up to an unknown scale factor, we cannot use metric depth as input. Instead, we obtain the metric depth maps by sparsifying depth maps collected from Kinect in order to use standard error metrics for a fair comparison. To simulate sparse

Table 3.3: Quantitative comparisons of the NYUv2 and TUM RGB-D dataset. In both static and dynamic scenes, DnD shows improved performance compared to recent methods.

Method	Dataset	Lower is better		Higher is better		
		RMSE	Abs Rel	$\delta_{1.25}$	$\delta_{1.25^2}$	$\delta_{1.25^3}$
BTS [33] (Single view)	NYUv2	0.392	0.110	0.885	0.978	0.994
Yang et al. [95] (Single view)	NYUv2	0.569	0.171	-	-	-
MonoDepth2 [76] (Single view)	NYUv2	0.617	0.170	0.748	0.942	0.986
P2Net [176] (Multi view)	NYUv2	0.533	0.147	0.801	0.951	0.987
DeepV2D [161] (Multi view)	NYUv2	0.403	**0.061**	**0.956**	**0.988**	0.996
DnD (Single view)	NYUv2	**0.362**	0.098	0.910	**0.988**	**0.998**
MC [32] (Single view)	TUM	0.840	0.204	0.664	0.931	0.981
MiDaS [74] (Single view)	TUM	0.819	0.196	0.678	0.911	0.965
DnD (Single view)	TUM	**0.631**	**0.175**	**0.751**	**0.947**	**0.982**

patterns existing in COLMAP, we only maintain the depth values according to the location of SIFT features [175] in the corresponding images. Since the number of SIFT features varies in each frame, we set the maximum number of features to 200 samples. Table 3.3 reports the quantitative comparisons with several other depth estimation methods on both datasets. DnD outperforms other single view methods, demonstrating the benefit of our method in less crowded benchmark indoor datasets. In supplementary materials, we show experimental results on TUM when we consider normalized relative depth as projected depth maps obtained from a scale ambiguous 3D model.

3.5 Conclusion

In this paper, we present a learning-based method for estimating dense depth maps of dynamic scenes collected from a moving monocular camera. Given RGB images and sparse depth maps projected from a 3D model, our method is able to predict absolute scale depth for multiple pedestrians and complex backgrounds. In complex and crowded indoor environments, this is a practical method to use for various robotics and augmented reality applications. Once we initially have the 3D model, our

method enables any freely moving mobile devices with a single-view camera to estimate dense depth maps. Our method still has limitations that we plan to address. The sparse depth maps projected from the 3D model may be inaccurate or not aligned with the current image. Also, our training method relies on FlowNet2.0 [172] and Mask R-CNN [173]. In the future, we would like to extend our approach to outdoor scenes.

II. Mapping

Chapter 4: TMO: Textured Mesh Acquisition of Objects with a Mobile Device by using Differentiable Rendering

4.1 Introduction

Recovering the 3D geometry of objects and scenes is a longstanding challenge in computer vision and is essential to a broad range of applications. Depth sensing technologies range from highly specialized and expensive turn-table 3D scanners and structured-light scanners to commodity depth sensors. More recently, advances in mobile devices have developed a new method for 3D capture of real-world environments with high-resolution imaging and miniaturized LiDAR. Specifically, the modern smartphone such as iPhone 13 Pro are equipped with RGB camera, accelerometer, gyroscope, magnetometer, and LiDAR scanner. These various sensors can provide high-resolution images, low-resolution depth from the LiDAR scanner, and associated camera poses offered by off-the-shelf visual-inertial odometry (VIO) systems such as ARKit [177] and ARCore [178].

Today's smartphones offer low-resolution depth maps [87] and valid poses. However, depth maps are very noisy and suffer from the limited range of depth sensors. Although this depth sensor can build a simple 3D structure such as a wall or floor, it cannot reconstruct objects with complex and varying shapes. Thus, the RGBD scanning methods [36, 179–181] are not suitable for these objects. Instead of the depth sensor, multi-view stereo (MVS) algorithms [30, 34, 141] reconstruct high-quality 3D geometry by matching feature correspondences across different RGB images and optimizing photometric consistency. While this pipeline is very robust in real-world environments, it misses the surface of low-textured areas [34]. Additionally, the resulting mesh generated by post-processing like Poisson reconstruction [37] heavily depends on the quality of matching, and the accumulated errors in correspondence matching often cause severe artifacts and missing surfaces. Because of the cumulative error

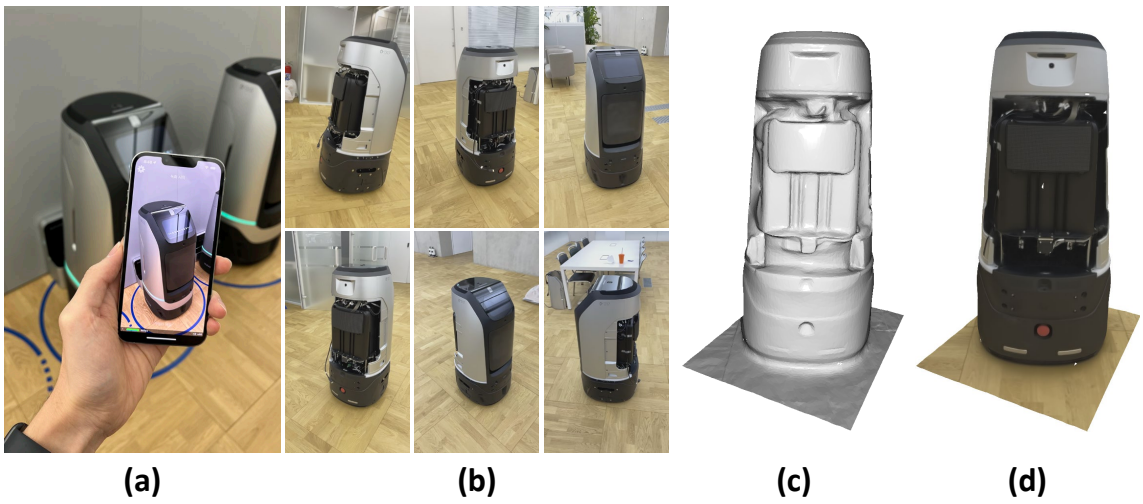


Figure 4.1: Example reconstruction results collected from a smartphone in the wild. (a) Data acquisition setup. (b) Images captured from a smartphone. (c) A reconstructed mesh. (d) A novel view of textured mesh. Our proposed method can reconstruct the high-quality geometric mesh with a visually realistic texture.

from the above pipeline, the texture mapping process [9, 38] leads to undesirable results. Reconstructing high-fidelity texture and 3D geometry of real-world 3D objects remains an open challenge.

Main Results: In this paper, we present a practical method to capture a high-quality textured mesh of a 3D object in the wild, shown in Fig. 4.1. We first develop a smartphone app based on ARKit [177] to collect images, LiDAR depths, and poses. Although the smartphone provides quite valid pose estimates, acquiring fine detail geometry and realistic texture requires highly accurate camera poses. Thus, we present an RGBD-aided Structure from Motion (SfM) which combines the advantages of both VIO [182] and SfM [29]. Since ARKit based on VIO is robust to the degradation of visual information such as low-textured surface, we perform incremental triangulation with initial poses obtained from ARKit. We also propose a depth filtering method to handle noisy depth from the smartphone. These filtered depth points are exploited as an additional depth factor for bundle adjustment. Our RGBD-aided SfM can estimate highly precise camera poses due to the good initial poses and additional constraints from the filtered depth.



Figure 4.2: Example objects collected by a smartphone in the wild

Our 3D geometry reconstruction process adopts a neural implicit representation [39, 183] for surface reconstruction with volumetric rendering. We observe that the neural implicit representation can perform more complete and smoother reconstruction than the classical methods which are known to be robust for 3D reconstruction in the wild. Furthermore, we introduce a new training method for neural implicit representations. In the early stage of training, we propose a regularization method that leverages the prior information from the classical MVS method. After obtaining the decent shape, we avoid using the prior information and generate sparse voxel octree to enable efficient sampling for training. Our training method can improve the performance of the neural implicit representations. Consequently, we show the generalization capabilities of our surface reconstruction method in real world objects collected by the smartphone.

Given a mesh extracted from the trained neural implicit representation, we can run classical texture mapping algorithms [9, 38] to generate texture maps that often exhibit blurring artifacts and color misalignment. We propose applying differential rendering [84] to fine-tune these texture maps obtained from classical texture mapping via a photometric loss. Compared to classical 3D reconstruction [30,34,36] and texture mapping [9,38] approaches, our method shows a better ability to reconstruct the 3D model and produce realistic textures. We evaluate our approach on the data collected by our smartphone application. The main contributions of this paper are summarized as follows:

- We present a unified framework to reconstruct the textured mesh using a smartphone.

- We propose a depth filtering scheme for noisy depths and refine initial poses from ARKit by using bundle adjustment with depth factor.
- Our pipeline builds on classical 3D reconstruction and texture mapping. We leverage a neural geometry representation to enable surface reconstruction of complex shapes and a differentiable rendering to generate high-fidelity texture maps.

4.2 Related Work

Classical 3D Reconstruction Traditional Structure from Motion (SfM) systems [29, 184] have been popular as a vision-based 3D reconstruction due to their robustness to various scenarios [142, 185, 186]. The SfM systems extract and match features [187] of adjacent images of the same scene and generate the camera poses and sparse 3D points via triangulation [97]. Then, Multi-view Stereo (MVS) algorithms [141, 188] compute the dense depth maps of each scene with multiple calibrated images. PatchMatch-based MVS [30, 34, 189, 190] methods are popular because they are proper for depth and normal optimization. After that, surface reconstruction methods [37, 191, 192] can be applied to reconstruct meshes from point clouds. However, all these methods often fail to reconstruct the 3D geometry of low-textured areas. RGB-D Reconstruction techniques fuse many overlapping depth maps into a single 3D model. Starting from the classical TSDF-fusion [193], several methods [36, 179–181] produce voxel-based TSDF with commodity-level depth cameras such as Kinect. However, these methods have difficulty handling outliers and thin geometry due to noisy depth measurements. Due to the noise and limited range of the LiDAR sensor in smartphone, RGB-D reconstruction methods fail to estimate the camera odometry.

3D Reconstruction with differentiable rendering Recently, NeRF [85] and its variants [194–197]

have used volumetric implicit representations for rendering and learn α-compositing of a radiance field along rays. These methods yield remarkable quality of novel view synthesis results and do not need ground truth masks. However, 3D geometry from volumetric representations is far from satisfactory due to their volumetric properties [197]. Some methods [198–200] reconstruct the surface by encoding the network as an implicit function such as a signed distance function (SDF) or occupancy function. Many hybrid approaches [35, 39, 201–203] unify surface and volumetric representations by regarding surface as defining volumes near the surface, enabling accurate surface reconstruction without masks. Their follow-up works [199, 204–206] import prior information from the learning-based network. We avoid exploiting prior hints obtained from learning-based models because they are expensive to acquire new training datasets and time-consuming to train other neural networks.

Classical Texture Optimization The matching between texture and geometry is a significant step for building realistic 3D models. Lempitsky et al. [207] apply the pairwise Markov Random Field to choose an optimal view for each face as texture. Waechter et al. [9] present a global color adjustment to mitigate the visual seams between face textures. Zhou et al. [38] optimize the camera poses using a color consistency measurement and geometric error via local image warping to obtain sharp texture. Bi et al. [208] employ a patch-based image synthesis to produce a texture per each face to optimize the camera pose and geometric error.

Texture Learning Recently, learning-based methods [209–212] have been proposed to generate textures for a 3D model. Huang et al. [211] introduce adversarial learning for texture generation in RGB-D scans. Thies et al. [212] store neural textures for a known mesh with given UV mapping and decode them to color for novel views. Some works [213, 214] learn neural texture with neural volumetric rendering. Differentiable rendering methods [40, 41, 215, 216] usually focus on estimating surface radiometric properties from images. These methods can learn to predict not only texture but also geometry

via image loss with ground truth masks. Unfortunately, differentiable rendering methods heavily rely on pixel-level accurate masks. Acquiring accurate ground-truth masks in the real-world environment is a difficult task [217]. Therefore, we apply the differentiable rendering to fine-tune texture maps with a fixed mesh.

4.3 Method

Given images, low-resolution depths and corresponding camera poses from ARKit, our goal is to reconstruct both the texture and geometry of real-world 3D objects. In Fig. 4.3, our pipeline consists of three stages: RGBD-aided Structure from Motion (SfM), Geometry Reconstruction, and Texture Optimization. Since both depth maps and camera poses provided by ARKit are very noisy, we construct a SfM system to filter out depth maps and refine poses. We then formalize our task as training a neural model for zero-level set of a signed distance function (SDF) guided by classical 3D reconstruction. After training, we extract the mesh with Marching Cubes [82] and apply classical texture reconstruction [9] to generate initial texture maps. Next, we exploit differentiable rendering [84] to finetune the texture maps with the supervision of multi-view images.

4.3.1 RGBD-aided Structure from Motion

The goal of this section is to refine initial poses from ARKit by performing a bundle adjustment with depth maps from the smartphone. Unlike simple structures like walls or floors, 3D reconstruction for capturing a high level of detail is required to refine inaccurate poses from ARKit.

Preprocessing We introduce two processes to obtain reliable depth estimates: single-view filtering and multi-view filtering. Single-view filtering utilizes a confidence map provided by ARKit, which has the

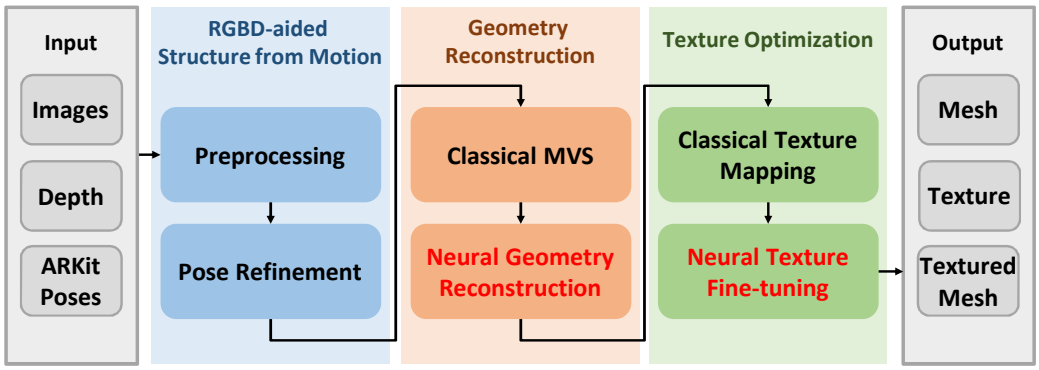


Figure 4.3: Overview of our pipeline. It is comprised of three steps: RGBD-aided Structure from Motion (Sec. 4.3.1), Geometry Reconstruction (Sec. 4.3.2), and Texture Optimization (Sec. 4.3.3). Our method builds on classical 3D reconstruction and texture mapping. To overcome the limitations of classical methods, we apply a neural geometry representation and neural texture fine-tuning to update 3D geometry and texture maps.

same resolution as the depth map. Each pixel in the confidence map has 0, 1, or 2 with the value 0 as least confident and 2 serving as most confident with respect to depth-value accuracy. We choose depth pixels corresponding to 1 or 2 in the confidence map. A main concept of the multi-view filtering is to remove outliers in terms of the reprojection error. Given the reference image I_{ref} with depth D_{ref}, the source image I_{src} with depth D_{src}, the depth map D'_{src} can be transformed from the source view to the reference view with accurate pixel-level correspondence. Then, the depth of pixel p in the reference image will be considered as an inlier if the difference δ_{depth} between D_{src} and D'_{src} is smaller than a small constant value ϵ. Then, assuming a set of neighboring N source images, we compute the number of inliers which fulfils $\delta_{depth} < \epsilon$ for each source image, and keep depth pixels if the number of inliers is greater than a certain threshold. Finally, we apply both single-view and multi-view filtering to remove all depth outliers.

Pose Refinement We consider camera poses obtained from ARKit as initial poses and exploit matching correspondences to perform incremental triangulation [29]. After combining filtered depth points from LiDAR and 3D points from incremental triangulation, we propose a depth factor and run bundle

adjustment with the depth factor and reprojection error. Let P_i denote the camera parameter of the i-th camera and X_j denote point parameters seen by the i-th camera. We minimize the proposed joint objective function E, which is formulated as follows:

$$E = \sum_i \sum_j \rho_i(\| \phi(P_i, X_j) - x_{ij} \|^2) + \frac{1}{\eta}|D_i - z_i|, \tag{4.1}$$

where Φ is the function that projects points to the image plane, and ρ denotes the Cauchy function as the robust loss function. x_{ij} is the feature point corresponding to the 3D point X_j seen by the i-th camera. z_i is the depth value of points projected from the 3D point X_j and D_i is the filtered depth of pixels corresponding to z_i. We define η as sensor measurement noise and use it to down-weight noisy depth. We notice that the depth factor has two benefits. First, additional information from the depth sensor is a useful constraint for accurate pose estimation and reliable triangulation. Furthermore, this depth factor guides the scale of the SfM to follow the metric scale of the depth sensor.

4.3.2 Geometry Reconstruction

Neural Geometry Representations Instead of classical 3D reconstruction, we adopt NeuS [39], which combines the advantages of both volume and surface rendering. In other words, it can extract accurate surfaces even without pixel-level masks. Our scanned object is represented by two Multi-layer Perceptrons (MLPs): geometry network f_{sdf}, which maps a 3D point $x \in \mathbb{R}^3$ to a signed distance, and color network f_{color} which converts a 3D point x and a viewing direction $v \in \mathbb{R}^3$ to a color c. The 3D geometry of an object is represented implicitly as an SDF with zero-level set $\{x \in \mathbb{R}^3 | f_{sdf}(x) = 0)\}$. These two parameters of geometry and color are learned by the volume rendering technique. For a ray $\{x(t) = o + tv | t \geq 0\}$ with camera center o and viewing direction v, the color is rendered with n

sampled points along the ray as follows:

$$\hat{C} = \sum_{i=1}^{n} T_i \alpha_i c(x(t), v), \tag{4.2}$$

where $T_i = \prod_{j=1}^{i-1}(1 - \alpha_j)$ is the accumulated trasnmittance, and α_i is the discrete opacity. Here, $\alpha_i = 1 - \exp(-\int_{t_i}^{t_{i+1}} \rho(t)\, dt)$, and $\rho(t)$ is opaque density function borrowed from the definition in NeuS [39].

Training Process RGBD-aided SfM already estimates both poses and sparse depths by matching feature correspondences. Given this prior information, we leverage an MVS algorithm [34] to estimate the depth and normal for all images with less cost. It is worth noting that we avoid using learning-based MVS algorithms because they are not robust to diverse real-world scenes and are hard to evaluate reliability. In this section, we modify the training process of NeuS to avoid losing fine details of complex shape. Our training process is divided into two stages. In the first stage, we train a neural radiance field with a prior depth and normal obtained from the MVS [34] algorithm to supervise the training process. Specifically, we sample K pixels p_k and their corresponding color values $I(p_k)$ in each iteration. Here, we propose a regularization method that relies on prior depth D and normal N from MVS algorithm. We unproject depth under the given pose to obtain 3D point $X(p_k)$. Then, we use this 3D point and its corresponding normal $N(p_k)$ from MVS to regularize the geometry network by minimizing

$$L_{reg} = \sum_{k} \; w_d |f_{sdf}(X(p_k))| \quad + w_n (1 - \langle \nabla_p f_{sdf}(X(p_k)), N(p_k) \rangle), \tag{4.3}$$

where w_d and w_n are weights for each term. This regularization will guide the SDF to learn explicit supervision from the MVS algorithm. Although this regularization is helpful for learning prior

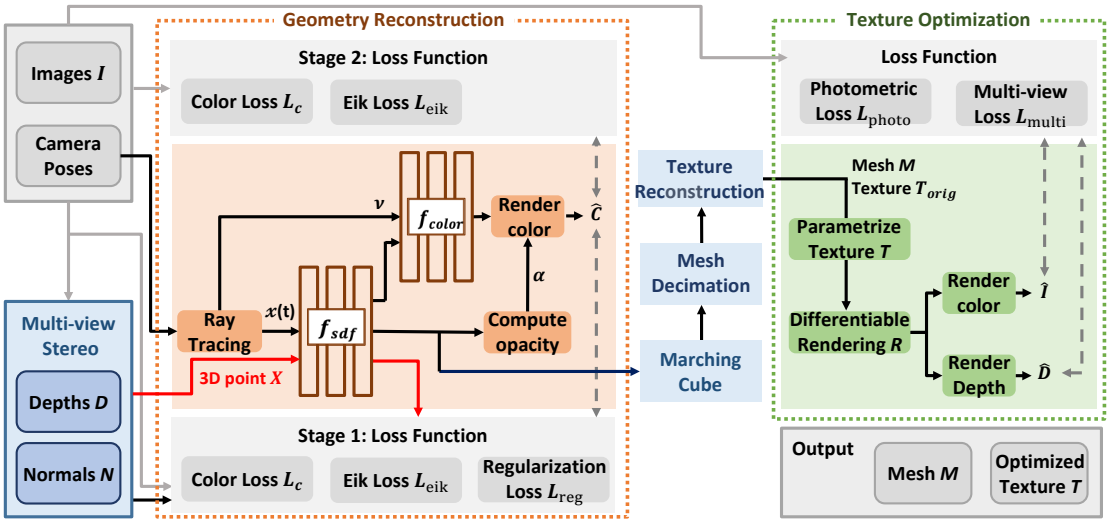


Figure 4.4: Overview of Geometry Reconstruction (Section 4.3.2) and Texture Optimization (Section 4.3.3). Our goal in geometry reconstruction is to train geometry network f_{sdf} and color network f_{color} via volumetric rendering with two training steps. In step 1, we leverage prior geometric information from the MVS algorithm [34]. Then, our method utilizes a sparse voxel octree to efficiently sample points near the surface. In texture optimization, we first utilize a classical texture reconstruction to generate texture maps for a given mesh. Then, our method applies a differentiable rendering to fine-tune texture maps.

geometry, it is susceptible to the noisy results derived from the MVS algorithm which tends to lose fine details and texture-less regions. Thus, we will only use this regularization in the first stage. The color of each pixel can be computed by E.q. 4.4. The color loss L_c is defined as

$$L_c = \sum_k |\hat{C}(p_k) - I(p_k)|. \tag{4.4}$$

The Eikonal loss [218] to regularize the SDF is formulated as

$$L_{eik} = \sum_{k,i} (\| \nabla f_{sdf}(p_{k,i}) \|_2 - 1)^2. \tag{4.5}$$

The overall loss function L in the first stage is defined as

$$L = L_c + L_{eik} + L_{reg}. \tag{4.6}$$

In the second stage, inspired by [183], we adopt a sparse voxel octree to guide the sampling process and capture fine details. After the first stage, we can extract a triangular mesh from the SDF via the Marching Cube [82] and define a sparse voxel volume based on this mesh. Then, based on the sparse voxel volume, we can avoid redundant point samples outside the sparse voxel and focus on point samples near the surface samples. Here, our total loss is $L = L_c + L_{eik}$, which is the same as E.q. 4.6 except for L_{reg}. To capture fine details, we will not use L_{reg} in E.q. 4.6.

Mesh Simplification A highly detailed 3D mesh from the SDF network with marching cube algorithms [82] leads to enormous memory requirements. To obtain a lightweight mesh, we explore popular quadric mesh simplification [99]. Although the mesh simplification generated small holes, the lightweight mesh enables us to reduce the computation complexity of the texture mapping process. We set the decimation ratio to 0.4%.

4.3.3 Texture Optimization

Classical Texture Reconstruction For a triangular mesh M provided by the previous section, we first utilize Waechter et al. [9] with poses and images across different views to construct a texture map T_{orig}. Compared to the color network from Section 4.3.2, this classical method [9] can acquire visually realistic texture images. However, it cannot avoid texture bleeding on the boundary of different views and blurring artifacts.

Texture Fine-tuning To tackle blurring and texture bleeding artifacts, we propose a fine-tuning step

by optimizing the texture map. Here, we parameterize a texture map T obtained from classical texture reconstruction. Given a 3D Mesh M with the 2D texture map T and multi-view images I_i with the corresponding camera poses P_i, we leverage a differentiable renderer R to generate an image $\hat{I}_i$ and depth $\hat{D}_i$, i.e. $\hat{I}_i, \hat{D}_i = R(M, P_i)$. A differentiable renderer R [84] performs mesh rasterization and then texture sampling to render an image $\hat{I}_i$. In other words, we project texture onto the image plane and compare it to the corresponding view of the image. Furthermore, we compare the rendered image with a different view image by minimizing multi-view photometric loss.

We use the combination of the L1 and SSIM [163] as the photometric loss L_{photo} in image space between the rendered image $\hat{I}_i$ and the reference image I_i as follows:

$$L_{photo} = \sum_i (1-\alpha) \parallel \hat{I}_i - I_i \parallel + \frac{\alpha}{2}(1 - SSIM(\hat{I}_i, I_i)), \tag{4.7}$$

where α is set to 0.85. Additionally, the differentiable render estimates depth $\hat{D}_i$ by interpolating z coordinates of each vertex. With the estimated depth $\hat{D}_i$ and intrinsic parameter K, we can generate a synthesized frame I'_i by reprojecting adjacent frames $I_{i'}$ to the current frame I_i with the camera intrinsic matrix K, the estimated depth $\hat{D}_i$, and the relative pose $P_{i \to i'}$. The multi-view photometric loss L_{multi} similar to Eq. 4.7 is formulated as:

$$\begin{aligned} I'_i(p) &= I_{i'} \langle \pi(K P_{i \to i'} \hat{D}_i(p) K^{-1} \tilde{p}) \rangle, \\ L_{multi} &= (1-\alpha) \parallel I'_i - I_i \parallel + \frac{\alpha}{2}(1 - SSIM(I'_i, I_i)), \end{aligned} \tag{4.8}$$

where $\tilde{p}$ is the homogeneous coordinate of p, π means projection from homogeneous to image coordi-

nates, and $\langle \cdot \rangle$ indicates the bilinear sampling function. We define our total loss L to be

$$L = L_{photo} + L_{multi}. \tag{4.9}$$

Finally, the differentiable rendering enables the loss gradients to back-propagate to update the textures parameters. Since our method does not require ground truth masks, we avoid considering optimization for the geometry. We observe that joint optimization with geometry and texture fails to converge without the ground truth mask.

4.4 Experiments

4.4.1 Experiment Settings

Data Collection: Our smartphone application, running on an iPhone 13 Pro using ARKit 6, can record various sensor outputs: synchronized images, poses, confidence maps, dense depth maps from the LiDAR scanner, and IMU sensor information. We built two types of datasets: ARKit-video and AR-capture. ARKit-video collects long video sequences similar to datasets constructed from RGBD-SLAM [219]. The second type is collected by our smartphone application based on Object Capture [1]. AR-capture is a multi-view dataset of objects consisting of high-quality 4K images with depth and gravity data. The ARKit-video collects thousands of frames sampled from the video with resolution of 1280x720. In contrast, AR-capture mostly contains less than fifty images at 4032x3096 pixels. We collect 11 objects: *robot arm 1*, *robot arm 2*, *plant*, *tree*, *sofa*, *bike*, *recliner chair*, *cafe stand*, *delivery robot*, *office chair*, and *camera stand*. We randomly select 70 % as the training images and utilize the

[1] https://developer.apple.com/videos/play/wwdc2021/10076/

30 % as test images for evaluating the novel view synthesis.

DTU Dataset: The DTU dataset [1] is a MVS dataset that provides reference point clouds acquired with laser sensor in the controlled lab. Since AR-capture and ARKit-video do not include 3D point clouds acquired with specialized hardware, we use the DTU dataset to evaluate the performance of geometry reconstruction. We follow the evaluation of NeuS [39] and exploit the chamfer distance as metric.

Implementation Details: We build the RGBD-aided Structure from Motion based on COLMAP [29], an existing SfM tool. Instead of SIFT [187], our system uses learning-based features [129,220,221] for feature extraction and matching to avoid failure in a featureless scene. ARKit offers an interface for accessing LiDAR data as a depth image. Following the work [39], the SDF network f_{sdf} is implemented as an 8-layer MLP a hidden dimension of 256, and geometric initialization [222] for the network parameters. We construct the color network f_{color} as MLPs with 4-layer MLP with hidden dimension of 256. We optimize networks in geometry reconstruction with Adam [137] and an initial learning rate of 5e-4. We train the first step for 60K iterations and the second step for 140K iterations. In terms of objects with complex shape, it takes 240K iterations for second step. In Texture Optimization, we utilize the public github code for classical method [9]. Then, for texture fine-tuning, the Adam optimizer is used with the learning rate for the texture parameter. This step takes 3000 epochs to fine-tune the texture parameter.

4.4.2 Experimental Results

In Fig. 4.5, we visualize the effectiveness of RGBD-aided Structure from Motion in Section 4.3.1 on our dataset. Figure 4.5 (a) represents the 3D point clouds by initial depth maps and poses obtained

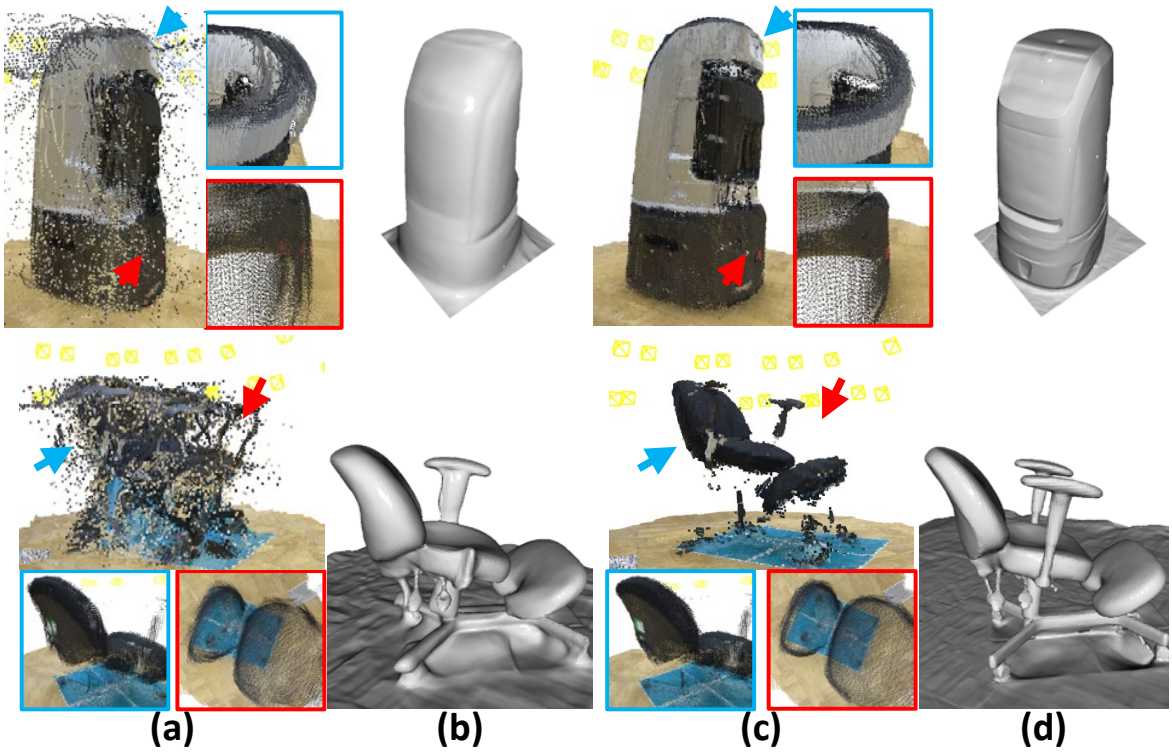


Figure 4.5: We show the benefits of RGBD-aided Structure from Motion. (a) illustrates the 3D point clouds obtained from initial depths with initial poses. In (c), we accumulate the 3D point clouds (by using filtered depths and refined poses) from RGBD-aided SfM. For a detailed description, we visualize the zoomed-in blue and red regions using filtered depth. Initial depth maps and poses from ARkit are input to the geometry reconstruction in Section 4.3.2 and its results are shown in (b). The results of our pipeline in (d) lead to higher quality reconstructions than (b) which does not apply RGBD-aided SfM.

from ARKit. Unfortunately, initial sensor data is too noisy to perform an accurate 3D reconstruction. After multi-view depth filtering as preprocessing, zoomed-in regions in (a) show the point clouds by reprojecting filtered depth maps with initial poses. Compared to initial data, multi-view depth filtering can handle noisy depth measurements. Figure 4.5 (c) provides the results of preprocessing and pose refinement. Pose refinement enables us to globally align 3D reconstruction without noticeable camera drift and with better local quality. For qualitative comparison, we take as input initial sensor data from ARKit and perform geometry reconstruction in Section 4.3.2 in Fig. 4.5 (b). Compared to our experimental results in Fig. 4.5 (d), it is difficult to achieve comparable quality and better reconstruc-



Figure 4.6: Reconstruction of our method, VolSDF [35], TSDF-Fusion [36], and a mesh from point clouds of ACMP [34] processed with screened Poisson surface reconstruction [37]. Note that the neural surface reconstruction is more perceptually convincing in the ARKit-video dataset.

tion. Consequently, our proposed RGBD-aided SfM significantly improves geometry reconstruction performance.

Figure 4.6 qualitatively compares our method to VolSDF [35], TSDF-fusion [36], and ACMP [34]. We apply screened Poisson surface reconstruction [37] to generate a mesh from point clouds of ACMP. Among all four algorithms, we apply the same pose estimation algorithm from Section 4.3.1 for fair comparison. TSDF-fusion, which requires sufficient scene overlap, exploits more frames than our method and ACMP. Due to the low-textured regions, the surface of ACMP is very noisy. TSDF-fusion utilizes depth maps obtained from low-resolution LiDAR. Here, since this low-resolution depth

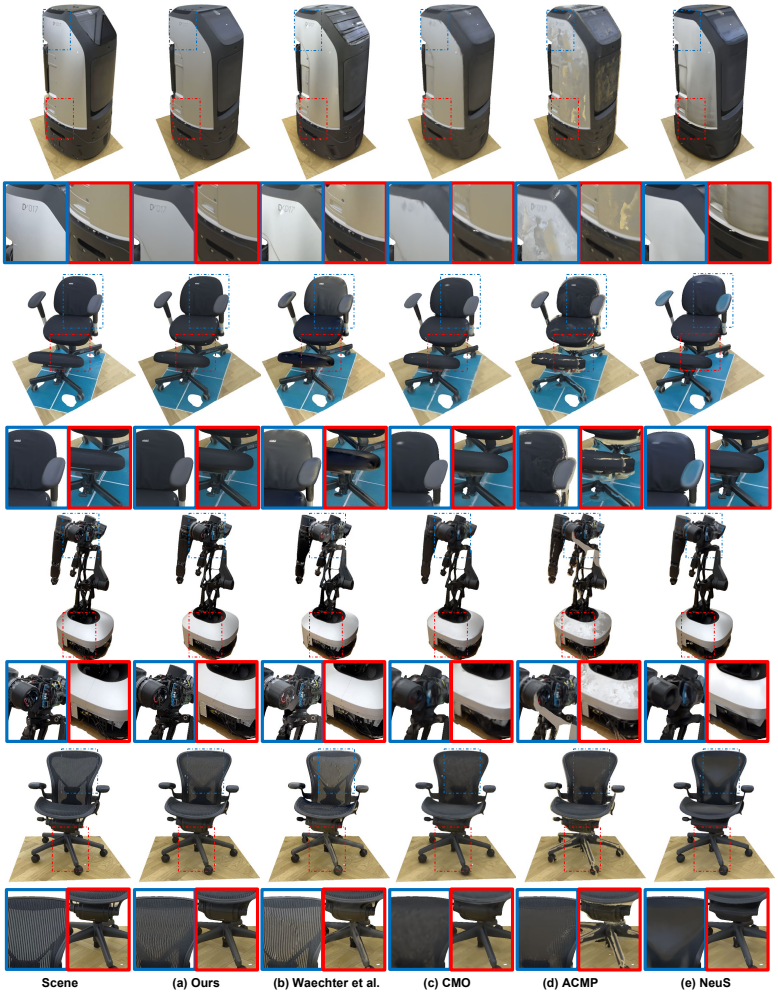


Figure 4.7: Qualitative comparison between (a) Ours, (b) Waechter et al. [9], (c) CMO [38], (d) ACMP [34], and (e) NeuS [39]. The result of our method is perceptually closer to the original scene.

sensor is very noisy, we only use filtered depth maps after preprocessing in Section 4.3.1. However, this model still cannot reconstruct either thin or challenging structures accurately (e.g., a chair armrest). Both implicit neural surface reconstruction approaches show comparable performance, and both are superior to classical 3D reconstruction. We observe that the difference between neural geometric representations (Ours and VolSDF) is negligible unlike pose quality. It is worth noting that using accurate poses as input significantly improves the 3D reconstruction quality as shown in Fig. 4.5.

Table 4.1: Quantitative comparison on our dataset. Ac and Av denote the AR-catpure dataset and ARKit-video dataset, respectively. We improve the LPIPS metric by 8.9% in AR-capture and by 3.8% in ARKit-video.

Method	Data	NeuS [39]			CMO [38]			Waechter et al. [9]			Ours		
		PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
robot arm 1	Ac	20.292	0.889	0.083	18.473	0.891	0.083	19.909	0.872	0.060	20.049	0.871	0.057
robot arm 2	Ac	20.741	0.882	0.086	21.107	0.883	0.092	20.190	0.862	0.067	20.652	0.858	0.062
plant	Ac	16.748	0.642	0.295	18.462	0.627	0.316	17.921	0.635	0.186	19.970	0.691	0.146
tree	Ac	18.409	0.869	0.117	18.448	0.870	0.127	17.518	0.858	0.096	18.331	0.858	0.102
sofa	Ac	19.910	0.872	0.139	21.170	0.840	0.136	19.759	0.829	0.113	21.608	0.894	0.109
bike	Ac	20.971	0.778	0.183	21.311	0.778	0.206	19.822	0.740	0.088	19.943	0.737	0.078
Mean	Ac	19.512	0.822	0.151	19.825	0.815	0.160	19.186	0.799	0.101	**20.095**	**0.818**	**0.092**
recliner chair	Av	24.920	0.893	0.099	24.761	0.876	0.112	23.692	0.864	0.054	25.045	0.896	0.056
cafe stand	Av	20.498	0.779	0.173	22.934	0.828	0.154	17.915	0.798	0.112	22.634	0.818	0.095
delivery robot	Av	22.412	0.897	0.126	23.549	0.908	0.128	25.246	0.929	0.073	23.451	0.906	0.086
office chair	Av	20.722	0.892	0.096	21.120	0.899	0.103	20.315	0.865	0.094	20.982	0.892	0.083
camera stand	Av	21.901	0.914	0.069	21.886	0.907	0.074	21.121	0.899	0.059	22.208	0.919	0.060
Mean	Av	22.092	0.875	0.113	22.850	0.884	0.115	21.658	0.871	0.079	**22.868**	**0.886**	**0.076**

Table 4.2: Quantitative comparison on DTU [1]. Chamfer distance is employed as the evaluation metric. COLMAP is superior to other neural geometry reconstructions. Our training process can improve the NeuS by 8 %.

Method	scan24	scan37	scan40	scan55	scan63	scan65	scan69	scan83	scan97	scan105	scan106	scan110	scan114	scan118	scan122	Mean
COLMAP [30] with trim=7	0.45	0.91	0.37	0.37	0.90	1.00	0.54	1.22	1.08	0.64	0.48	0.59	0.32	0.43	0.45	0.65
UNISURF [201]	1.32	1.36	1.72	0.44	1.35	0.79	0.80	1.49	1.37	0.89	0.59	1.47	0.46	0.59	0.62	1.02
NeuS [39]	1.37	**1.21**	0.73	**0.40**	1.20	0.70	0.72	1.01	1.16	0.82	0.66	1.69	**0.39**	0.61	0.51	0.87
VolSDF [35]	1.14	1.26	0.81	0.49	1.25	0.70	0.72	1.29	1.18	**0.70**	0.66	1.08	0.42	0.61	0.55	0.86
Ours without 2nd step	1.04	1.61	0.67	0.64	0.88	0.70	0.62	1.14	1.10	0.89	0.66	**1.02**	0.50	0.58	0.51	0.84
Ours	**0.96**	1.37	**0.67**	0.60	**0.88**	**0.70**	**0.45**	**1.14**	**1.10**	0.84	**0.66**	1.11	0.50	**0.58**	**0.50**	**0.80**

For qualitative comparison of textured meshes, we compare our proposed method with four different methods: 1) ACMP [34] is a state-of-the-art multi-view stereo algorithm: 2) Color Map Optimization (CMO) [38] is a texture mapping method for RGB-D scanning: 3) Waechter et al. [9] is a large-scale texture reconstruction based on global color adjustment, 4) NeuS [39] is the recent neural surface reconstruction based on volumetric rendering. We reconstruct a 3D mesh from the point cloud of ACMP with Screened Poisson Surface reconstruction [37]. In the case of CMO, since explicit depth maps from low-resolution LiDAR are too noisy, we utilize depth maps rendered by a triangular mesh from our geometry reconstruction step for a fair comparison. With these rendered depth maps and corresponding images, we run a color map optimization for mapping color images onto our reconstructed mesh. To train NeuS, we follow a training process similar to that in Sec. 4.3.2. We then extract a mesh



Figure 4.8: Qualitative comparisons of texture optimization algorithms based on differentiable rendering: (a) our method, (b) Lin et al. [40], and (c) Munkberg et al. [41].

from the SDF network in NeuS and apply the Blender Smart UV Project tool to obtain a per-vertex UV mapping. To fill a texture for this mesh, points densely sampled from triangle mesh with interpolated UV coordinates are fed to the color network, and eventually we can obtain texture maps for the given mesh.

In Fig. 4.7, we show the rec onstructed texture for novel views. The visual results of our approach are visually sharper and perceptually closer to the ground truth scene. Results of NeuS and CMO exhibit blurring artifacts and color misalignment. The reconstructed mesh from ACMP is very noisy due to incorrect geometry. Here, Waechter et al. (Fig. 4.7 (b)) is the result of classical texture reconstruction in Section 4.3.3. This method is more photorealistic and sharper than NeuS, CMO, and ACMP. Then, after texture fine-tuning, our method avoids the seams and texture misalignment often seen with Waechter et al. Table 4.1 reports the quantitative results on two datasets. Our method outperforms other texture optimization methods on all evaluation metrics. It indicates that our method satisfies both the high fidelity of the texture and its conformity to the ground truth.

In Fig. 4.8, we show the results of other texture optimization methods [40, 41] based on differentiable rendering [84]. Both methods aim to learn geometry, textures, and lighting parameters.

However, they heavily rely on ground truth masks which require additional annotation costs. Without ground truth masks, we note that joint optimization over geometry and texture is unstable and often has negative effects on geometry. For a fair comparison, we therefore provide the fixed mesh topology and train only texture and lighting parameters through differentiable rendering [84]. Since we capture objects in the wild under unknown environmental lighting conditions, it is difficult to disentangle materials and lighting via differentiable rendering. Neither method [40, 41] is able to generate realistic texture maps compared to either our method or classical texture reconstruction approaches.

Experimental Results on DTU dataset: Unlike our dataset, DTU [1] is captured by specialized hardware in the controlled lab environment. Thus, this dataset provides the ground truth poses and the RGBD-aided SfM is not necessary. As shown in Table 4.2, the surfaces obtained by COLMAP [30] with trimming value 7 achieve high-quality reconstruction. Our method exploits MVS [30] to supervise the training NeuS and outperforms NeuS. For the ablation study, when we only train our network with the first step (ours without 2nd step), its result is not better than our method in some scans such as scan37. In our experiment, sparse voxel octree sampling does not have a significant effect on geometry performance. Since our training process can be applied to any other 3D neural surface reconstruction methods. In the future, we will apply this training process to other recent 3D neural surface reconstruction algorithms [35, 203, 223], which shows better results on DTU.

4.5 Conclusion

In this paper, we have developed a practical framework to reconstruct high-quality textured meshes of 3D objects by using a smartphone. Given initial poses from ARKit, multi-view images, and low-resolution depth maps, our approach can refine initial poses and depth maps from ARKit. We

adopt neural geometry representation [39] for surface reconstruction and improve its performance by modifying the training process. Finally, a differentiable rendering is designed for fine-tuning the texture map acquired from the classical texture mapping method. In the future, our goal is to reduce the training time which can be comparable with classical methods. Unfortunately, most neural geometry reconstruction [35, 39, 203, 223] requires a long training time. We would like to reduce the training time in our framework for practical purposes.

Chapter 5: LTM: Lightweight Textured Mesh Extraction and Refinement of Large Unbounded Scenes for Efficient Storage and Real-time Rendering

5.1 Introduction

Photorealistic reconstruction and rendering of real-world objects and scenes is a longstanding problem of importance with several applications in computer vision, robotics and AR/VR. In recent years, Neural Radiance Fields (NeRFs) [85, 88, 196, 224] have been quite successful at novel view synthesis. However, the scene representation is learned as an implicit volumetric representation that is expensive to evaluate. This is a crucial limitation for many applications, such as Augmented and Virtual Reality (AR/VR), often requiring real-time rendering with low memory and compute power. Since modern graphics engines are highly optimized to rasterize triangle meshes, researchers have started exploring ways to extract explicit mesh representations from volumetric functions.

MobileNeRF [43] uses a polygon mesh with texture maps storing feature vectors and opacity. Although their rendering quality is good, the mesh quality is far from an ideal surface. While NeRF2Mesh [42] and NeRFMeshing [225], based on volumetric density fields, excel at generating highly detailed geometry, they often result in noisy and bumpy surfaces and numerous floaters, especially in planar regions, which are common in large-scale scenes. In contrast, neural Signed Distance Function (SDF) methods [35, 39, 201] typically excel in reconstructing accurate surface geometry. However, they may not be able to capture small or detailed geometry, and their rendering quality is inferior to volumetric density-field methods. BakedSDF [3] overcomes these limitations by applying mip-NeRF 360 techniques to train SDF representation, though the resulting mesh representation has a high memory overhead.

Previous neural SDF methods that rely on the Marching Cube algorithm [82] result in meshes

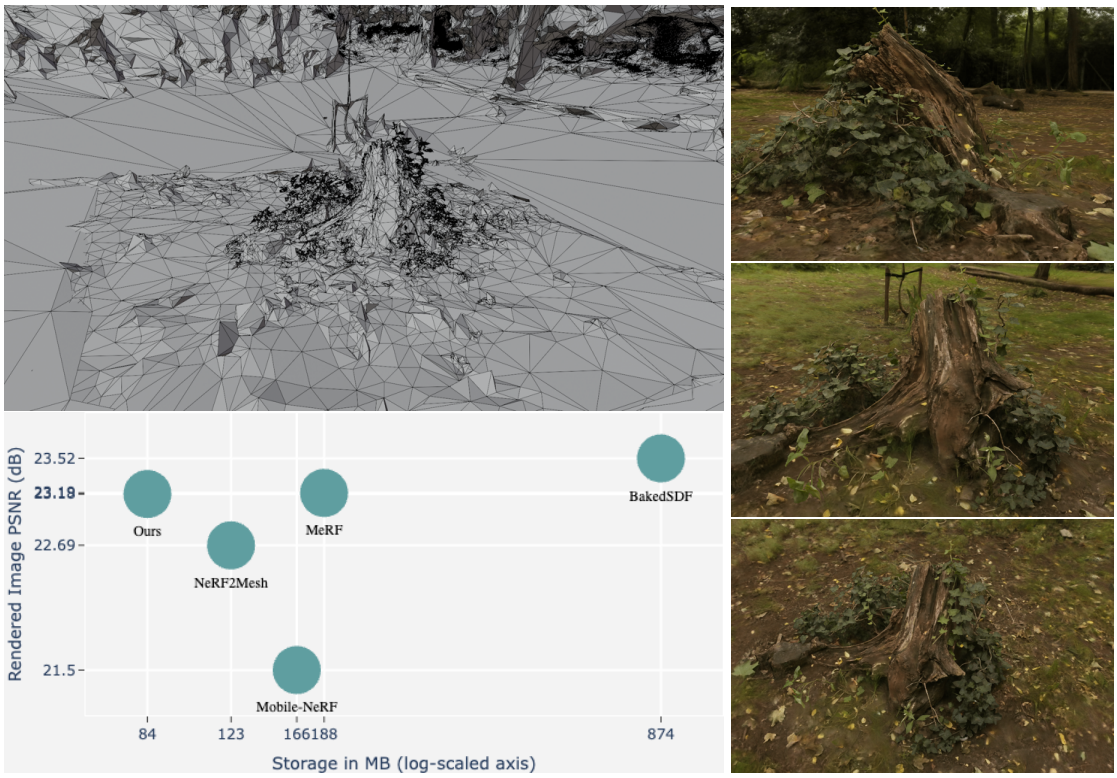


Figure 5.1: Example results using LTM showing mesh geometry (top-left) and novel-view synthesis (right). Bottom right plot compares performance of different methods on mip-NeRF360 outdoor dataset. Our method achieves comparable rendering quality to BakedSDF while reducing the mesh size considerably.

with high storage overhead. This results in a significant challenge for practical applications on mobile devices, where low disk storage and fast data transfer are essential. Instead of relying on the Marching Cubes, some methods [226–229] propose differentiable mesh reconstruction methods. These methods aim to extract an isosurface from an implicit function and can be seamlessly integrated with differentiable rendering approaches [230, 231]. However, most methods primarily concentrate on a single object with ideal implicit surface functions, unsuited for large unbounded real-world scenes [3, 88]. Reconstructing an ideal mesh for the unbounded large-scale scene is still an open problem, often resulting in non-manifold geometry or missing surfaces. Consequently, the renderings can result in visual artifacts due to the incompleteness of the mesh geometry.

Main Results: In this paper, our goal is to achieve a realistic rendering with a compact mesh representation that optimally utilizes disk storage for unbounded large-scale scenes, all while preserving geometric quality. Our method initially trains neural fields for geometry and appearance separately. For geometry reconstruction, we adopt the BakedSDF approach [3] to train an SDF in a contracted space [88]. Regarding the appearance field, we utilize hash-based encoding [196] with two shallow MLPs to represent diffuse and specular colors. The SDF and appearance fields are compact representations for reconstructing a high-quality mesh and achieving realistic rendering.

Given the initial geometry representations, we perform mesh extraction and decimation. As unbounded large-scale scenes occupy a large amount of disk storage, our method focuses on generating a lightweight mesh without compromising geometric details. We adaptively extract the mesh for the central and background regions in the contracted space, respectively. Subsequently, we modify a mesh decimation [232] designed for large-scale scenes. Our method eliminates redundant triangles corresponding to simple geometric features, such as planar surfaces, while preserving complex geometric features.

Since the appearance field is trained by volumetric rendering, the initial model is unsuitable for rasterization-based rendering. Moreover, the mesh from the previous step still fails to accurately represent areas with highly-detailed geometry (thin structures) or semi-transparent objects. These mesh defects hurt rendering quality. Thus, we use differentiable rendering [84] and utilize image-space supervision to refine the mesh surface by rectifying these mesh defects while preserving geometric accuracy. Due to the highly simplified mesh and various regularization techniques, our approach can deform the mesh topology without causing severe mesh distortion, a common issue in gradient-based mesh optimization [233, 234]. Compared to mesh-based neural rendering methods [3, 42, 43, 225], our approach results in lower storage overhead while providing comparable rendering quality. To sum up,

our contributions include:

- We effectively utilize classical mesh decimation techniques to extract compact geometry from neural SDFs of large unbounded scenes.
- We introduce a new approach for joint optimization of appearance and vertex deformation that preserves geometric details and improves rendering quality.
- We achieve a trifecta of precise geometry, efficient storage, and rendering quality comparable to the state of the art on diverse captured environments such as mip-NeRF 360, Tanks & Temples, and the Deep Blending dataset.

5.2 Related Work

Neural Rendering Neural Fields achieve remarkable photo-realistic rendering quality. One of the most successful approaches is Neural Radiance Fields (NeRFs) [85], which adopts volumetric implicit representations and trains a radiance field through ray marching. Mip-NeRF [88, 224] presents casting a conical frustum into the scene space and extends this integrated positional encoding to unbounded scenes by using a novel contraction function. Instead of learning volumetric density, neural surface reconstruction methods [21, 35, 39, 201] propose learning a Signed Distance Function (SDF) for surface reconstruction via volumetric rendering. Furthermore, several methods have been proposed to accelerate NeRFs by utilizing efficient data structure [196, 235, 236], although achieving real-time rendering for high-resolutions remains a challenge. Alternatively, to enhance rendering speed, "baking" based methods [3, 42, 43, 225, 237, 238] typically precompute and store a trained NeRF into a more efficient representation. SNeRG [238] adopts a deferred shading model [239] and bakes trained NeRFs

into sparse voxel grid representations with neural features. In particular, the mesh is widely used as geometry representation and bakes neural fields into its texture. MobileNeRF [43] builds on a mesh rasterization pipeline for efficient rendering on mobile architecture. NeRF2Mesh [42] introduces a two-step framework for NeRF training and textured mesh extraction. NeRFMeshing [225] proposes a novel method to produce a truncated SDF from NeRFs and extracts the accurate mesh. BakedSDF [3] enables us to reconstruct high-quality mesh and bake it into a small appearance model based on spherical Gaussians for fast rendering. DNMP [240] and NeuRaS [241] adopt mesh-based geometry and rasterization-based rendering for large driving scenes.

Mesh Processing Traditional graphics research has developed effective techniques for simplifying meshes, based on vertex decimation [242, 243], edge collapse [99, 232, 244–246] and appearance preservation [247]. Vertex decimation methods prioritize vertices based on shape heuristics and remove the least important ones. Edge collapse methods rank the edges based on the cost and subsequently contract pairs of vertices. QSlim [99, 245] introduces the concept of the Quadric Error Metric (QEM), which is defined as the sum of quadrics representing the distance of a point from a set of planes. Following the edge collapse, they utilize this QEM to identify which edges should be removed while preserving geometric topology. Lindstrom et al. [248, 249] use vertex clustering [243] on a uniform grid and accumulate a quadric error matrix to the occupied grid cell. Wang et al. [250] employ RGB-D sequences to reconstruct a lightweight mesh.

Differentiable Rendering and Mesh Optimization Recently, many differentiable rendering techniques [251] have emerged, aimed at facilitating end-to-end optimization of the rendering process by employing useful gradients. In particular, several works [144, 252] focus on approximating the rasterization stage of the rendering to compute meaningful gradients. Furthermore, modular differentiable renderers [84,253,254] exist, allowing users to customize and modify specific functions. Differentiable

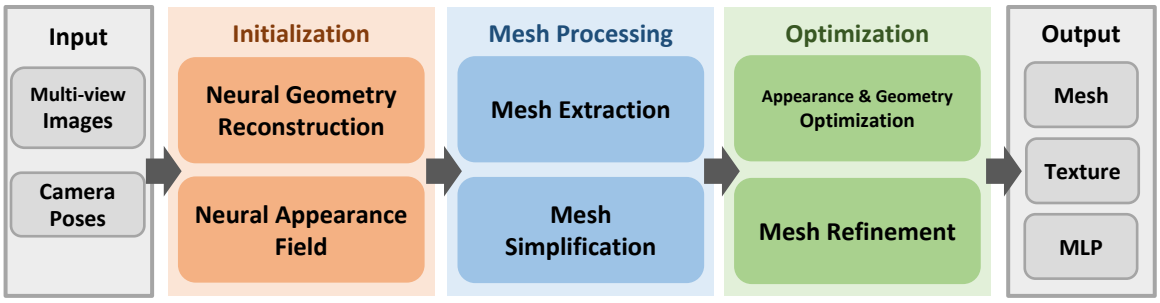


Figure 5.2: **Overview of Our Pipeline**. It consists of three steps: Initialization (Sec. 5.3.1), Mesh Processing (Sec. 5.3.2), and Optimization (Sec. 5.3.3). Our method focuses on baking the initial appearance and geometry representation, trained by volumetric rendering, into an explicit mesh and a memory-efficient texture conducive to real-time rendering. We employ classical mesh decimation to minimize storage requirements, introduce the optimization stage to facilitate rasterization-based rendering and refine the detailed mesh structure.

mesh reconstruction techniques [226–228] can build a mesh through gradient-based optimization. In particular, differentiable rendering [229–231] can offer 2D supervision to refine and enhance the geometric quality of the mesh. Some works [230, 231] adopt an inverse rendering approach, leveraging physics-based rendering to synthesize shapes. Nevertheless, gradient-based mesh optimization [233] is highly sensitive to initialization and prone to significant mesh distortion. Some methods [233,234,255] incorporate a novel ADAM optimizer and employ strong regularization techniques. However, this research is currently limited to generating single objects and is challenging to apply to large-scale scenes.

5.3 Method

Given multi-view images and corresponding camera poses, we aim to reconstruct the texture and geometry of unbounded large-scale scenes. Fig. 5.2 shows the three main stages of our pipeline: Initialization, Mesh Processing, and Optimization. In the first stage, we employ volumetric rendering to train neural implicit representation for geometry f_{sdf} and appearance f_{color}. Subsequently, we leverage the geometry representation f_{sdf} to extract a high-resolution mesh via Marching Cubes [82] and reduce

the number of triangles without causing geometry distortion. Next, we utilize differentiable rendering [84] to jointly optimize the explicit 3D mesh M and implicit appearance field f_{color} obtained from the first stage and refine the mesh surface.

5.3.1 Initialization of Geometry and Appearance

In this stage, we first learn neural implicit representations for both appearance and geometry. A neural radiance field [85] is an implicit representation of a 3D scene volume, a mapping from a 3D position x and a ray viewing direction $d \in \mathbb{R}^3$ to an RGB color c and volume density ρ. In practice, the networks modeled by MLPs are used to generate this volume density and view-dependent color:

$$\rho = f_{density}(x), \quad c = f_{color}(x, d). \tag{5.1}$$

To render the color of a pixel, we first compute the 3D points t_i along the viewing ray associated with each pixel p as $x(t) = o + td$. Then, estimated volume densities ρ_i and colors c_i are utilized to approximate a volume rendering equation [256] using numerical quadrature:

$$C = \sum_i T_i \alpha_i c_i, \quad T_i = \exp(-\sum_{j<i} \rho_i \delta_i), \tag{5.2}$$

where T_i is the accumulated transmittance and $\alpha_i = 1 - \exp(-\rho_i \delta_i)$ is the discrete opacity of the point. $\delta_i = t_i - t_{i-1}$ is the distance between two neighboring points along the ray. To deal with large unbounded scenes, mip-NeRF 360 [88] introduces a proposal MLP for efficient sampling and space contraction to represent an unbounded scene. The scene contraction function f (in Eq. 5.3) maps the

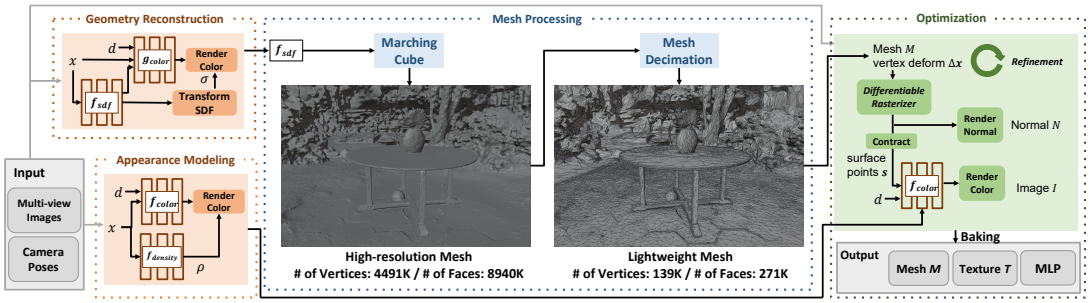


Figure 5.3: **Our Approach:** In Section 5.3.1, we start training both the appearance f_{color} and geometry f_{sdf} representations for a large-scale scene using volumetric rendering (shown in orange). Then, in Section 5.3.2, our method extracts the mesh from f_{sdf} and significantly simplifies its structure, resulting in the lightweight mesh M (shown in blue). In Section 5.3.3, we jointly train vertex deformation Δx and f_{color} computed from appearance modeling.

unbounded points into bounded space:

$$f(x) = \begin{cases} x & ||x|| \leq 1, \\ (2 - \frac{1}{||x||})\frac{x}{||x||} & ||x|| \geq 1. \end{cases} \tag{5.3}$$

Appearance Similar to previous works [42,43,238], we decompose an appearance field f_{color} into shallow MLPs to learn diffuse colors c_d and specular colors c_s separately. Particularly, the view-dependent effects are modeled by a small MLP that can be easily integrated into a fragment shader. In order to ensure the MLPs are able to model an unbounded scene well, we adopt strategies similar to the ones introduced in mip-NeRF360 work, specifically, scene contraction, proposal MLP, and regularization for the background. We train both MLP networks f_{color} and $f_{density}$ with a multi-resolution hash positional encoding [196] using the volume rendering objective in Eq. 5.2. The appearance field trained in this stage is further optimized to facilitate rasterization-based rendering, replacing the computationally expensive ray marching. This optimization is described later in Section 5.3.3.

Geometry Initial geometry modeling in our method is similar to BakedSDF [3]. The core mecha-

nism for neural surface reconstruction was introduced in VolSDF [35] for small objects. BakedSDF shows that by leveraging the efficient sampling and contraction ideas introduced in mip-NeRF360 [88], VolSDF [35] can be extended to large unbounded scenes. We follow this and jointly train two MLPs using the volumetric rendering objective, (i) f_{sdf} - representing geometry of a scene as a signed distance function, and (ii) g_{color} - a color prediction network, as shown in the *Geometry Reconstruction* block in Fig.5.3.

Following VolSDF, we define the volume density $\sigma(x)$ as a function of surface geometry f_{sdf}, namely, $\sigma(x) = \alpha\Psi_\beta(f_{sdf}(x))$, where Ψ_β is the Cumulative Distribution Function (CDF) of the Laplace distribution with zero mean and β is the learnable scaling parameter. The Eikonal loss [218] to regularize the f_{sdf} is defined as,

$$L_{eik} = (\parallel \nabla f_{sdf}(x) \parallel_2 -1)^2. \tag{5.4}$$

The color MLP g_{color}, in addition to the position and viewing direction, also takes the predicted surface normal and a 256-dimensional geometric feature as inputs. It is important to note that the sole purpose of color MLP g_{color} is geometry initialization and we discard it after this step. This is not to be confused with the shallow decomposed color MLP f_{color} trained specifically to model appearance that is also jointly refined with mesh vertices in the later stage.

5.3.2 Mesh Extraction and Decimation

In this stage, we first extract a mesh by using Marching Cubes (MC) [82] algorithm on learned SDF (f_{sdf} evaluated over a uniform grid) and simplify the extracted mesh using a geometry-aware quadric error based mesh decimation. While this approach works pretty well for extracting meshes

of well-captured small-scale objects and scenes, it presents unique challenges for large-scale scenes. Meshing neural SDFs of large-scale scenes with marching cubes can often result in spurious surfaces arising from (i) insufficient grid resolution to model complex geometry of nearby objects, and (ii) poorly learned geometry of distant objects due to insufficient view-points observing such distant regions. Furthermore, since our SDF f_{sdf} is trained in the contracted space, the resulting mesh is also defined in the contracted space. When we *uncontract*, i.e. warp the mesh vertices back into the original Euclidean space, it further exacerbates the geometry distortion in the background region.

We aim to preserve the detailed geometry in central (foreground) regions while achieving a coarser but less noisy reconstruction of distant (background) regions, within a small triangle budget. We achieve this by treating foreground and background regions distinctly during both extraction and simplification steps. During MC extraction, we set a finer grid resolution of 2048 and a spatial range of $[-1, 1]^3$ for the central region, and a coarser resolution of 1024 and a spatial range of $[-2, 2]^3$ for the background region. For decimation, we first partition the central and background regions and employ an aggressive mesh decimation in background regions. This helps us transform bumpy and noisy background areas into smooth surfaces while also reducing redundant triangles.

We first apply vertex clustering and then run quadric error based simplification algorithm - QSlim [99, 245] for all clusters. Then, we run edge contraction for each cluster to reach a threshold and repeat this process until convergence. The QSlim algorithm calculates a quadric metric for each vertex to decide whether the two vertices should be merged and aggregates these quadrics during each edge collapse operation. The basic quadric error in QSlim [99] is defined as the point-to-plane distance

from vertex $v \in \mathbb{R}^3$ to all planes $p \in \mathbb{R}^4$ of its adjacent faces $N(v)$:

$$E(v) = \sum_{p \in N(v)} (p^T v)^2 = v^T \Big(\sum_{p \in N(v)} pp^T \Big) v = v^T Q_v v. \tag{5.5}$$

The quadric Q_v is the symmetric 4x4 matrix, and each edge computes this quadric by adding the quadrics of the corresponding two vertices. The most commonly used quadric error is defined solely based on 3D vertex positions v.

Intuitively, the most effective strategy is to represent planar surfaces with a smaller number of large triangles while keeping a consistent number of triangles for complex shapes. Variants that incorporate additional surface attributes, such as normals or textures in calculating the quadric error are proposed in [245]. We experimentally observed that compared to the basic QSlim algorithm, using both the vertex positions $v \in \mathbb{R}^3$ and the corresponding vertex normal $n \in \mathbb{R}^3$ for quadric error is very effective at prioritizing triangle decimation on flat surfaces while retaining them on complex surfaces. The combination of vertex clustering and geometry-aware QSlim helps us in effectively reducing the number of triangles while preserving the original geometry.

5.3.3 Appearance and Geometry Optimization

In this stage, we optimize the appearance model jointly with mesh vertex deformation and refine the mesh topology using a geometric complexity based criterion.

Joint optimization of appearance and vertex deformation Given the simplified mesh M and the initial appearance field f_{color}, we employ the fast differentiable rasterizer introduced in Nvdiffrast [84] to jointly optimize the mesh geometry and the appearance field. The initial appearance field f_{color} trained using the volume rendering objective (Eq. 5.1) is agnostic of the surface geometry of the

extracted and simplified mesh. To finetune the model to fit the mesh geometry, we rasterize the mesh into each of the training camera poses and obtain surface points s and face normals N interpolated per pixel. Then we apply the contraction function in Eq. 5.3 to warp the surface points. This is because the appearance field defined in contracted space proves to be more effective in enhancing the rendering quality [88]. The appearance field f_{color} takes these surface points s and viewing direction d as input, rendering an image $\hat{I}_i$. Initially, rendering performance is degraded because the appearance field only utilizes the intersection point between the ray and the surface rather than accumulating all sampled points along the ray. We finetune this appearance field to be compatible with mesh rendering, akin to the functionality of a fragment shader.

Given the positions x of the mesh vertices, we train per-vertex deformation Δx using differentiable rendering [84]. We explore two approaches to optimize vertex deformation: a standard ADAM optimizer [137] and a rotation-invariant ADAM optimizer [234]. The ADAM optimizer is widely employed [42, 230] for deforming mesh topology to enhance rendering performance. While the ADAM optimizer often results in mesh distortions such as flipped triangles and self-intersections, our method mitigates mesh distortion, especially on smooth surfaces, due to the rigorous mesh decimation performed in Section 5.3.2.

Objective Function To jointly train appearance field and vertex deformation, we minimize the image-space L1 loss $L_{appearance}$ between the rendered image $\hat{I}_i$ of the deformed surface and the groundtruth image I_i for a camera view P_i:

$$L_{appearance} = \sum_i |\hat{I}_i - I_i|. \tag{5.6}$$

Naively training vertex deformation is highly likely to result in undesirable mesh topology char-

acterized by a disorganized arrangement of triangles, often referred to as a "polygon soup." To prevent this mesh distortion, we employ various regularization methods. We use the Laplacian mesh smoothness loss and the normal consistency loss inspired by [257].

$$L_{lap} = \sum \parallel LV \parallel_2 \tag{5.7}$$

where $V \in \mathbb{R}^{N\text{x}3}$ (N is the number of vertices) is the matrix of deformed vertex coordinates, indicating $x + \Delta x$, at each training step, and $L \in \mathbb{R}^{N\text{x}N}$ is the graph Laplacian of the mesh [258]. Minimizing the magnitude of a vertex's differential coordinates (LV) reduces its distance from its neighbors' average position. The normal consistency loss is

$$L_{nc} = \sum_{i,j \in F} (1 - n_i \cdot n_j)^2, \tag{5.8}$$

where n_i and n_j denote the surface normal of two adjacent faces and F is the set of faces sharing a common edge. Additionally, in Section 5.3.2, a high-resolution mesh is rasterized in the first pass before decimation. The output of this rasterization is used to interpolate face normals per pixel, resulting in a normal map $\hat{N}_i$ for every viewpoint. We calculate the L1 loss between a rendered normal N_i from current mesh M and the $\hat{N}_i$:

$$L_{norm} = \sum_i |\hat{N}_i - N_i|. \tag{5.9}$$

L_{norm} provides geometry supervision, particularly in planar regions, helping prevent severe mesh distortion when moving vertices. The total loss function L is defined as:

$$L = L_{color} + \lambda_{lap} L_{lap} + \lambda_{nc} L_{nc} + \lambda_{norm} L_{norm}, \tag{5.10}$$

where λ_* is the weight to balance between the loss terms.

Mesh Refinement Inspired by NeRF2Mesh [42], we adopt a classical mesh refinement approach based on mesh subdivision and decimation during training. The mesh refinement strategy is applied differently to the central and background regions. Instead of rendering error proposed by [42], which can arise from many factors including specularity, we employ a criterion: the vertex deformation values for subdividing triangle faces. The large vertex deformation implies a more substantial triangle budget to accurately capture the intricacies of highly detailed geometry associated with those vertices. We sort the absolute values of vertex deformation and determine a 10 % of faces to increase triangles. Then, we apply $\sqrt{3}$ subdivision [259], which adds a vertex at the barycenter of a triangle and connects it to all vertices of this triangle. This scheme enables us to augment the triangle budget for complex geometry and mitigate self-intersection during vertex deformation. Furthermore, the mesh decimation presented in Section 5.3.2 is applied after every mesh subdivision to remove redundant triangles on simple and flat geometry. For background regions, we examine all training views and check faces intersecting with ray casting. We utilize dilation for all invisible masks and employ mesh decimation to remove redundant faces.

Rotation-Invariant ADAM The rotation-invariant ADAM formulation [234] modifies the update rule in ADAM, resulting in the update step: $\Delta x = \zeta \cdot v \cdot l_{ref}$. Here, ζ is the learning rate, and v is the standard ADAM update step, which is the loss gradient with respect to x. l_{ref} is the coefficient initially set to the edge length incident to x, and it gets updated based on the average v.

5.4 Experiments

Datasets: To validate the effectiveness of our method, we mainly conduct experiments on seven unbounded, large-scale 360° indoor and outdoor scenes from the mip-NeRF 360 [88] dataset. Also, we validate the generalization ability of our approach on two different datasets, including large-scale scenes: two scenes from the Tanks & Temples [100] dataset and two scenes from the Deep Blending [6] dataset.

Implementation Details: For all datasets, we subsampled every eight frames for test sets following the mip-NeRF 360 paper [88]. For geometry reconstruction in Section 5.3.1, we use SDFstudio [260] to implement BakedSDF. In Section 5.3.2, we reduce 95% of the triangles in the central region and 99% of the triangles in the background regions through decimation. In Section 5.3.3, we run all experiments for 40k iterations, and a learning rate is annealed from 1e-2 to 1e-3. We set the loss weights to λ_{lap}=1e-3, λ_{nc}=1e-3, and λ_{norm}=1e-4. We apply mesh refinement at 8k, 16k, and 24k training steps. Subsequently, we compute diffuse color and specular features using f_{color} to export textures. All these color maps and features are 3-channel .png files. Additional details are provided in the supplementary materials.

5.4.1 Experimental Results

We compare our proposed method with three different methods: 1) MobileNeRF [43] is the state-of-the-art method, enabling NeRF to achieve real-time rendering of explicit 3D mesh and neural features. 2) NeRF2Mesh [42] employs a volumetric rendering to reconstruct the coarse mesh and refine the mesh by jointly optimizing appearance and geometry. 3) BakedSDF [3] is a recent method for optimizing a neural surface-volume representation for real-time rendering of large unbounded scenes.

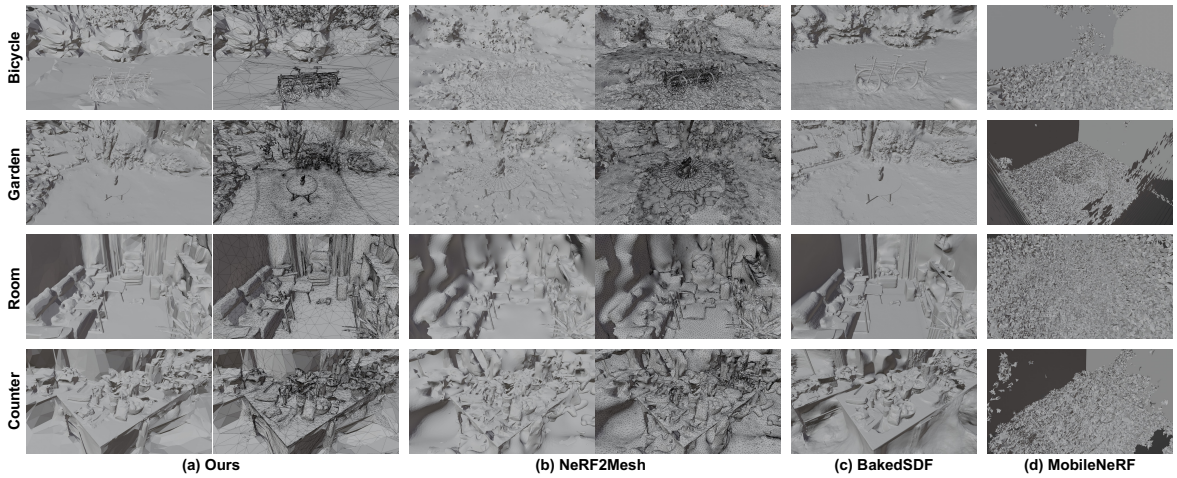


Figure 5.4: **Visual comparison of Our Method, NeRF2Mesh [42], BakedSDF [3], and MobileNeRF [43].** Our method and NeRF2Mesh show shading mesh and wireframe mesh extracted without texture, respectively.

Table 5.1: **Quantitative Comparison on mip-NeRF 360 Dataset.** "-" denotes a missing implementation. The best and second best-performing algorithms for each metric are bolded and underlineed.

Method	Mesh	Outdoor (PSNR ↑ / SSIM ↑ / LPIPS ↓)				Indoor (PSNR ↑ / SSIM ↑ / LPIPS ↓)				
		Bicycle	Garden	Stump	Mean	Room	Counter	Kitchen	Bonsai	Mean
Instant-NGP [196]	X	22.1 / 0.49 / 0.49	24.5 / 0.65 / 0.31	23.6 / 0.57 / 0.45	23.4 / 0.57 / 0.42	29.2 / 0.85 / 0.30	26.43 / 0.80 / 0.34	28.5 / 0.82 / 0.25	30.3 / 0.89 / 0.23	28.6 / 0.84 / 0.28
mip-NeRF 360 [88]	X	24.4 / 0.68 / 0.30	26.9 / 0.81 / 0.17	26.4 / 0.74 / 0.26	25.9 / 0.75 / 0.24	31.6 / 0.91 / 0.21	29.5 / 0.89 / 0.20	32.3 / 0.92 / 0.13	33.5 / 0.94 / 0.18	31.7 / 0.92 / 0.18
MobileNeRF [43]	O	21.7 / 0.43 / 0.51	18.8 / 0.59 / 0.36	**23.9** / 0.56 / 0.43	21.5 / 0.53 / 0.43	28.9 / 0.85 / 0.28	25.1 / 0.72 / 0.29	**26.8** / 0.79 / 0.79	23.8 / 0.71 / 0.72	26.2 / 0.77 / 0.52
NeRF2Mesh [42]	O	22.1 / 0.48 / 0.51	23.4 / 0.55 / 0.40	22.5 / 0.54 / 0.46	22.7 / 0.52 / 0.46	25.7 / 0.79 / 0.35	23.9 / 0.71 / 0.35	24.0 / 0.61 / 0.36	25.0 / 0.77 / 0.29	24.7 / 0.72 / 0.34
NeRFMeshing [225]	O	21.1 / - / -	22.9 / - / -	22.6 / - / -	22.2 / - / -	26.1 / - / -	20.0 / - / -	23.6 / - / -	25.6 / - / -	23.8 / - / -
BakedSDF [3]	O	22.0 / **0.57** / **0.37**	**24.9** / **0.75** / **0.21**	23.6 / **0.59** / **0.37**	**23.5** / **0.64** / **0.32**	28.7 / 0.87 / 0.25	**25.7** / **0.81** / 0.28	26.7 / **0.82** / 0.24	27.2 / **0.85** / 0.26	27.0 / **0.84** / 0.26
Ours	O	**22.4** / 0.52 / 0.44	23.5 / 0.64 / 0.30	23.7 / 0.57 / 0.42	23.2 / 0.58 / 0.39	**29.3** / **0.88** / **0.23**	25.1 / 0.77 / **0.27**	26.5 / 0.80 / **0.21**	**27.3** / 0.84 / **0.22**	**27.1** / 0.82 / **0.23**
Ours w/o Opt	O	16.1 / 0.30 / 0.53	18.8 / 0.45 / 0.39	19.4 / 0.39 / 0.50	18.1 / 0.38 / 0.48	17.2 / 0.54 / 0.46	16.2 / 0.47 / 0.47	18.1 / 0.50 / 0.36	16.4 / 0.54 / 0.43	17.0 / 0.51 / 0.43
Ours w/o GO	O	21.1 / 0.48 / 0.47	23.2 / 0.65 / 0.30	23.0 / 0.53 / 0.46	22.5 / 0.55 / 0.41	27.5 / 0.85 / 0.26	23.6 / 0.72 / 0.33	25.5 / 0.78 / 0.23	25.9 / 0.76 / 0.3	25.4 / 0.78 / 0.28
Ours w. RADAM	O	21.6 / 0.48 / 0.46	23.4 / 0.66 / 0.30	23.5 / 0.56 / 0.45	22.8 / 0.57 / 0.40	27.5 / 0.85 / 0.26	24.1 / 0.74 / 0.31	25.8 / 0.80 / 0.22	26.8 / 0.80 / 0.26	26.0 / 0.80 / 0.26

We also add NeRFMeshing [225], which distills the rendering NeRF into a neural field that represents TSDF (Truncated SDF) and reconstructs the 3D mesh. As BakedSDF and NeRFMeshing have not released public code, we adopt the metrics provided in their original papers. BakedSDF has made its GLB file available. We utilize this file for visualization.

Figure 5.4 qualitatively compares our method with NeRF2Mesh [42], BakedSDF [3], and MobileNeRF [43]. We visualize the shading meshes extracted without texture, which is the most effective way to visualize geometric differences. Furthermore, we include a wireframe mesh, distinguishing between our method and NeRF2Mesh. Visualizing the wireframe mesh from BakedSDF lacks

Table 5.2: **Disk Storage (MB), Number of Vertices (V) and Faces (F).** The unit for the number of vertices and faces is 10^3. "M" denotes the 10^6 unit. We save 73× the number of triangles and reduce 11× disk storage.

Method	Bicycle		Garden		Stump		Room		Counter		Kitchen		Bonsai		Mean	
	Disk ↓	V ↓ / F ↓	Disk ↓	V ↓ / F ↓	Disk ↓	V ↓ / F ↓	Disk ↓	V ↓ / F ↓	Disk ↓	V ↓ / F ↓	Disk ↓	V ↓ / F ↓	Disk ↓	V ↓ / F ↓	Disk ↓	V ↓ / F ↓
MobileNeRF [43]	201	1435 / 6638	135	723 / 340	164	1069 / 497	220	903 / 421	170	864 / 399	95	552/ 256	84	606 / 284	153	879 / 1262
NeRF2Mesh [42]	129	815 / 858	117	650 / 779	123	**415 / 820**	69	328 / 462	79	517 / 384	89	601 / 434	70	319 / 636	97	387 / 667
BakedSDF [3]	1087	111248 / 44384	513	10773M / 21072	943	19667 / 38467	385	7227 / 14110	585	12212 / 23892	659	13832 / 27056	641	13629 / 26679	687	1564M / 27951
Ours	**83**	**270 / 529**	**50**	**113 / 218**	**120**	425 / 847	**45**	**124 / 238**	**58**	**138 / 268**	**40**	**70 / 136**	**42**	**233 / 459**	**62**	**196 / 385**

meaningful interpretation due to its extremely high face density. MobileNeRF generates a "triangle soup" that inaccurately represents the scene's geometry. Also, compared to the mesh reconstructed by NeRF2Mesh, our method shows sharp details of complex geometry and smooth surfaces on the ground or wall. Moreover, in the wireframe mesh, our method employs more triangles in intricate shapes like the bicycle or the garden vase while allocating smaller triangles to planar regions or simpler structures. While the geometry of BakedSDF exhibits superior quality compared to our mesh reconstruction, achieving this high-quality mesh necessitates approximately 73× more triangles in Table 5.2.

Our method effectively achieves the optimal tradeoff between rendering quality and disk storage cost. For comparison, we report two tables for rendering quality and disk storage. Table 5.1 reports the rendering quality using the standard PSNR, SSIM [163], and LPIPS [261] on mip-NeRF 360 dataset. Instant-NGP [196], and mip-NeRF 360 [88] are volumetric rendering approaches that do not reconstruct explicit geometry, a topic that lies outside the primary focus of our study. In Table 5.2, we report the disk storage in megabytes and the number of vertices and faces. For a fair comparison, all the mesh files are saved in .obj and .mtl formats. Texture images are stored as .png files. Other assets, such as view-dependency MLP, are saved as .json files, except BakedSDF, which only provides gLTF format files. Our method demonstrates comparable rendering quality in outdoor scenes (with a 0.3 PSNR drop) and indoor scenes (with a 0.1 PSNR increase) while saving 73× triangles and 11× storage compared to the state-of-the-art BakedSDF. In comparison to NeRF2Mesh and MobileNeRF,

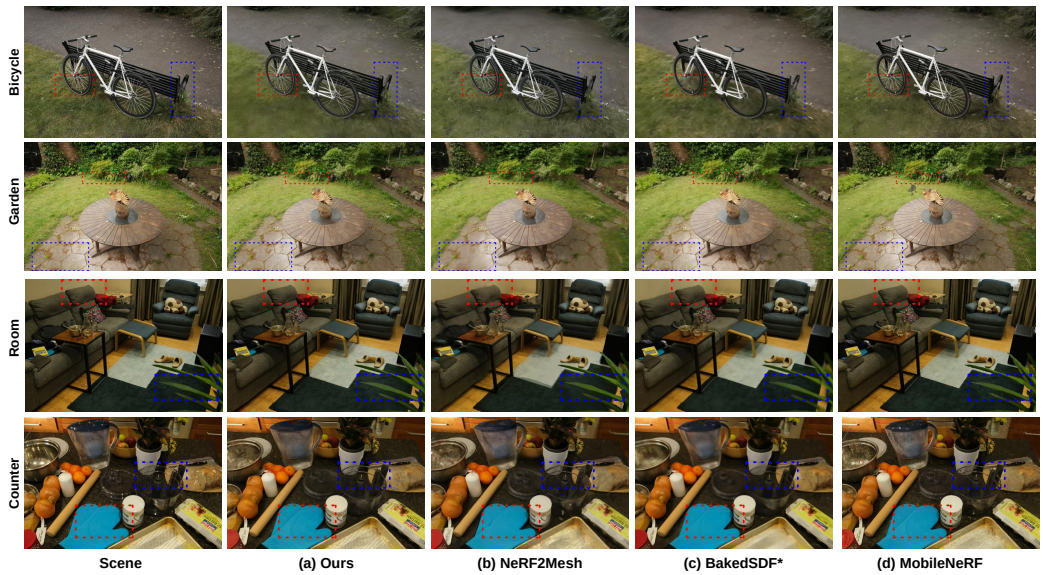


Figure 5.5: **Qualitative Comparisons with Existing Methods.** We visually compare our proposed technique with NeRF2Mesh [42], BakedSDF* [3], and MobileNeRF [43]. The visual results of BakedSDF* generated in its first stage through volumetric rendering. Red and blue dotted lines emphasize subtle differences in rendering quality.

our method shows better-rendering quality (Table 5.1), demands less disk storage, and fewer triangles (Table 5.2), and captures superior mesh quality (Fig. 5.4). In Table 5.5, we measure the rendering speed and GPU memory by running in-browser on a MacBookPro (2020, M1).

Figure 5.5 compares our method to the state-of-the-art baselines that focus on rendering quality: NeRF2Mesh [42], BakedSDF* [3], and MobileNeRF [43]. In their first stage, BakedSDF* represents the volumetric rendering results obtained from a surface-based volumetric representation. The visual results of our approach exhibit sharper details than other methods. In the Bicycle and Room scenes, the mesh surface of the armrest for a bench and tree leaves exhibits noticeable jiggling, resulting in distorted images. Unlike BakedSDF*, our method effectively manages the thin structure of the wheel.

Experimental Results on Deep Blending and Tank & Temples Datasets: We train and evaluate our method on four scenes: Playroom and DrJohnson from the Deep Blending dataset, and Barn and Courthouse from the Tanks & Temples dataset. The former includes bounded indoor scenes, while the

Table 5.3: **Quantitative Comparison.** We report PSNR ↑, SSIM ↑, and LPIPS ↓ on Deep Blending and Tanks & Temples datasets.

Method	Mesh	Deep Blending [6]		Tanks & Temples [100]	
		DrJohnson	Playroom	Barn	Courthouse
Instant-NGP [196]	X	27.7 / 0.84 / 0.38	19.5 / 0.78 / 0.46	23.4 / 0.72 / 0.43	20.0 / 0.68 / 0.53
mip-NeRF360 [88]	X	29.1 / 0.90 / 0.24	29.6 / 0.90 / 0.25	27.2 / 0.81 / 0.32	21.3 / 0.72 / 0.47
MobileNeRF [43]	O	25.9 / 0.78 / **0.31**	28.7 / 0.85 / **0.28**	24.4 / 0.77 / 0.24	17.7 / 0.54 0.36
NeRF2Mesh [42]	O	24.0 / 0.75 / 0.51	26.6 / 0.83 / 0.41	23.4 / 0.72 / 0.43	18.2 / 0.63 / 0.56
Ours	O	**26.5** / **0.82** / 0.40	**29.1** / **0.87** / 0.32	**26.5** / **0.83** / **0.24**	**20.2** / **0.73** / **0.34**

Table 5.4: **Disk Storage (MB), #Vertices (V) and #Faces (F).**

Method	DrJohnson		Playroom		Barn		Courthouse	
	Disk ↓	V ↓ / F ↓	Disk ↓	V ↓ / F ↓	Disk ↓	V ↓ / F ↓	Disk ↓	V ↓ / F ↓
MobileNeRF [43]	174.9	602 / 285	281.8	1097 / 521	289.2	1080 / 502	263.6	934 / 417
NeRF2Mesh [42]	108.6	414 / 815	118.7	401 / 793	131.9	397 / 793	145.7	451 / 843
Ours	**26.7**	**46 / 92**	**60.9**	**203 / 401**	**40**	**86 / 171**	**47.4**	**97 / 192**

latter comprises large unbounded outdoor scenes. Their capture methodology differs from that of the mip-NeRF 360 dataset, and they do not utilize 360-degree scenes. In Tanks & Temples, we exclude sky regions during training and evaluation, as mesh-based rendering tends to encounter challenges with far-away sky regions.

Ablation: In Fig. 5.6, we visualize the zoom-in shading mesh extracted without texture. "Ours w/o Opt" denotes we did not execute our optimization stage in Section 5.3.3 after mesh processing. The optimization stage enables us to represent the geometric details of garden vases and bicycle wheels. Our method can represent more detailed structures than BakedSDF and a smoother surface than NeRF2Mesh. Moreover, it is noteworthy that when employing rasterization-based rendering with the initial color model without the optimization stage, the rendering quality is poor, as shown in Table 5.1. In this table, "Ours w/o Geo" only trains appearance model f_{color} without training vertex deformation. Training vertex deformation yields a 1.7 gain in indoor scenes by refining the mesh surface. "Ours w. RADAM" denotes that we use the rotation-invariant ADAM (RADAM) for updating mesh vertices

Table 5.5: **FPS and GPU memory** on mip-NeRF 360 dataset

FPS / GPU (MB)	MobileNeRF	NeRF2Mesh	Ours
MacBookPro (1920x1080 Resolution)	59 / 324	205 / 130	212 / 115

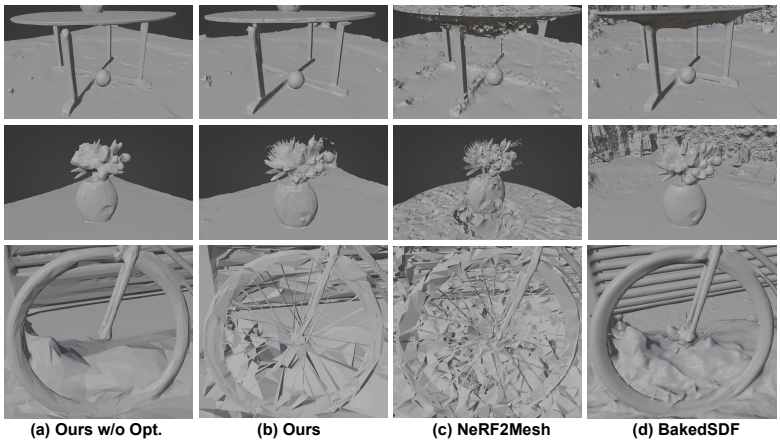


Figure 5.6: **Zoom-in Shading Mesh** to see more geometric details.

instead of a naive ADAM. However, it provides benefits for preventing mesh distortion.

5.5 Conclusion

This paper introduces a practical approach for reconstructing geometrically accurate meshes with photorealistic rendering quality. Our method achieves a compact mesh representation in unbounded large-scale scenes while preserving highly detailed geometry. Furthermore, the appearance field corresponding to this lightweight mesh is compatible with the classical rasterization pipeline, enabling efficient rendering. However, our method has limitations. BakedSDF employs a well-designed appearance model based on spherical Gaussians. The potential for enhancing rendering quality exists by incorporating a more sophisticated appearance model. Moreover, the gradient-based mesh optimization in the final stage may still result in mesh distortions, including self-intersections. We plan to employ a carefully designed optimizer to mitigate such distortions in future work.

Chapter 6: MeshGS: Adaptive Mesh-Aligned Gaussian Splatting for High-Quality Rendering

6.1 Introduction

Rendering realistic depictions of real-world objects and environment has long been a significant challenge with numerous practical applications in fields like computer vision, graphics, and AR/VR. Most graphics and game engines use polygonal mesh-based representations of the objects in the scene [262], including real-time rendering and animation. Recently, Neural Radiance Fields (NeRF) [85, 88, 196, 224] have demonstrated promising capabilities for 3D reconstruction and novel-view synthesis. However, NeRFs, which rely on volumetric rendering, suffer from slow rendering speeds and are not easily compatible with modern graphics engines [2, 42, 263]. Instead, researchers have been investigating methods based on neural implicit representation to facilitate mesh-based rendering. MobileNeRF [43] utilizes a classical rasterization pipeline, incorporating z-buffers and fragment shaders, to train neural fields based on polygonal mesh representations. Subsequently, various methods [3, 42, 225, 240, 241] first reconstruct a 3D mesh and then utilize this mesh to train appearance models employing high-speed rasterization [84, 253, 264]. These techniques facilitate real-time rendering and demonstrate potential for integration into graphics and game engines. However, all of these methods are constrained by the quality of the mesh surface, particularly exhibiting a loss in rendering quality when dealing with highly detailed structures, a well-known limitation of mesh-based approaches [37, 39]. Given the inherent difficulty in achieving perfect geometric reconstruction for real-world unbounded scenes, rendering algorithms must demonstrate robustness in handling mesh artifacts such as thin-level structures.

More recently, the 3D Gaussian Splatting method [10] has attracted considerable interest due

to its high-quality rendering and real-time rendering speed. Compared to mesh-based representation, 3D Gaussian splats excel in capturing intricate details of the scene with high fidelity due to their high degree of freedom in splat positions and shapes. However, 3D Gaussian splats are limited in their ability to represent the geometry of a scene and are not widely applicable to various applications. SuGaR [5] extracts mesh solely based on 3D Gaussian splats. Their method shows detailed structure at the object level and tightly binds the 3D Gaussian splats to the surface of the mesh. Other previous works adopt a similar method for tightly binding the 3D Gaussian splat to the mesh triangles [4, 265]. This is very useful for rigging, animation, or any deformation tasks. However, this assumes that the mesh can cover nearly all geometrical elements of the target object, such as the head or a human avatar. In unbounded scenes, since meshes frequently exhibit artifacts, we need better strategies to effectively bind 3D Gaussian splats to the mesh surface.

Main Results: In this paper, we present a new approach to integrate the strengths of mesh and 3D Gaussian splats in rendering and geometry for unbounded large-scale scenes. For geometry reconstruction, we first employ the BakedSDF [3] technique and extract a 3D mesh using Marching cube algorithm [82]. In unbounded large-scale scenes, meshes typically consist of a large number of triangles to represent complex geometric objects in the scene. Properly initializing Gaussian splats with millions of vertices and triangles becomes exceptionally challenging. Inspired by LTM [2], we employ mesh decimation and eliminate redundant triangles to extract a lightweight mesh with fewer triangles. Based on this lightweight mesh, we initialize the Gaussian splats for each mesh triangle. Subsequently, we introduce a mesh-based Gaussian splatting method that consists of two components. First, we conduct forward splatting considering mesh geometry and eliminate any Gaussian splats that are occluded by the mesh surface. This impacts our training process to align the Gaussian splats with the explicit mesh geometry and to remove redundant Gaussian splats positioned behind the mesh

surface considering all training viewpoints.

Secondly, we segregate tightly-bound Gaussian splats from loosely-bound Gaussian splats using the distance between each Guassian splat's center and their corresponding mesh triangles. As current mesh reconstruction methods cannot fully capture the 3D geometry of real-world scenes, we introduce loosely-bound Gaussian splats to depict regions where explicit mesh-based representation is not available, thus representing their appearance. We employ different training and densification strategies for each type of splat. The tightly-bound Gaussian splats are aligned with the mesh surface through various geometric regularization techniques, whereas the optimization of loosely-bound Gaussian splats relies solely on image supervision.

Our main contributions include:

- We present a novel approach for mesh-based Gaussian splatting. We integrate Gaussian splats with the triangle mesh and introduce a new method for initializing and training Gaussian splats based on the triangle mesh.

- We differentiate between two types of Gaussian splats based on their distance from the mesh surface. Subsequently, we introduce different training strategies for each type of Gaussian splat to accommodate different rendering requirements.

- Our method surpasses the rendering quality of the state-of-the-art mesh-based neural rendering algorithms [2,3] by 2dB PSNR on mip-NeRF360 dataset. Compared to the original 3D Gaussian splatting [10], we achieve comparable rendering quality while utilizing 30% fewer Gaussian splats for rendering.

6.2 Related Work

Neural Rendering. Neural Rendering has recently received considerable attention. Neural Radiance Fields (NeRF) [85] utilize a multi-layer perceptron (MLP) to encode a scene by applying differentiable volumetric rendering to represent RGB radiance and density. Various methods have been proposed to accelerate the training time and rendering speed by leveraging data structures [235,236,266], encoding techniques [88, 196, 224], and baking processes [238, 267]. Other researchers utilize neural implicit surface representation [218], employing MLP to map coordinates to a signed distance function (SDF) or occupancy grids. These approaches [3, 21, 35, 39, 201, 223, 268] involve converting volume density to SDF or occupancy and perform training with differentiable volumetric rendering.

Mesh-Based Rendering. Polygonal mesh-based rendering [264] is a classic problem in computer graphics to visualize 3D models on a 2D screen using rasterization. To address rendering speed issue, recent neural rendering methods also leverage explicit mesh and bake the neural appearance, enabling mesh-based neural rendering. SNeRG [238] builds on deferred shading [239] and bakes neural appearance into sparse voxel grid. MobileNeRF [43] directly optimizes the triangle faces and embeds opacities and features into texture maps through a baking process. Subsequently, other approaches involve baking the neural fields into textures and representing view-dependent effects using either a small neural network [42, 225, 240, 241] or spherical Gaussians [3]. VMesh [263] proposes hybrid representation combining mesh and volume rendering to achive high efficiency and expressiveness. Shells [269] extract the mesh and conduct volumetric rendering within a narrow band surrounding the mesh. LTM [2] employs significant mesh decimation to achieve efficient mesh representation for large-scale scenes while simultaneously training neural appearance and geometry through differentiable rendering [84]. Many existing methods encounter limitations due to mesh artifacts stemming

from 3D reconstruction techniques in real-world scenarios, posing challenges in achieving photorealistic rendering quality with neural fields. Our approach aims to address this limitation by leveraging Gaussian splatting for mesh-based rendering.

Point-Based Rendering and Gaussian Splatting Point-based rendering is a technique for rendering 3D scenes using point-based geometry representation instead of traditional polygon meshes. In this context, a variety of recent approaches have been proposed for the point-based rendering [270, 271], sphere rendering [272], and differentiable splatting [273]. The 3D Gaussian Splatting [10] employs 3D Gaussian splats [274], which are rendered through sorting and rasterization processes. SuGaR [5] utilizes Poisson reconstruction [191] to extract a mesh from 3D Gaussian splats and bind them to the triangle for rendering. GaussianAvatars [265] and GaMeS [4] focus on tightly binding 3D Gaussian splats to meshes and rendering dynamic scenes, including human avatars or single objects. However, these methods are mainly tailored for a single object or human avatars. Our goal is to develop a new mesh-based Gaussian splatting technique suitable for large-scale scenes, even in cases where the mesh only captures moderate levels of geometry. More recently, 2D Gaussian splatting [11] utilize the 2D Gaussian primitives instead of 3D Gaussian splats. In our paper, we focus solely on 3D Gaussian splat representation and the integration of 3D Gaussian splats with the mesh. We provide a more detailed analysis comparing 2DGS [11] with our method in the supplementary material.

6.3 Method

Given posed images, our goal is to reconstruct both the precise geometry and rendering high-fidelity images of unbounded large-scale scenes. An overview of our pipeline is illustrated in Fig. 6.1. First, we employ the BakedSDF [3] technique and extract a 3D mesh M via Marching Cubes [82].

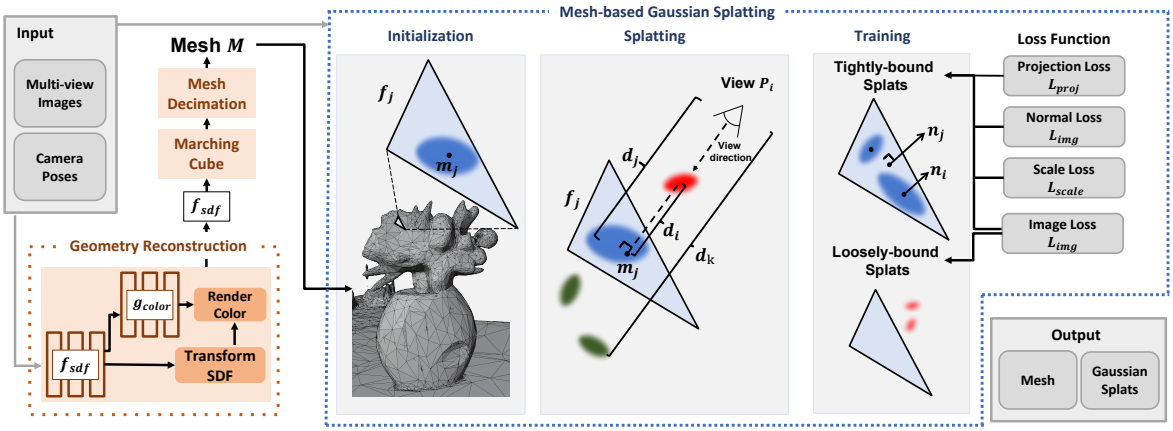


Figure 6.1: **Our Approach:** In Section 6.3.1, we jointly train the geometry f_{sdf} and appearance g_{color} using differentiable volumetric rendering (shown in orange). Then, we extract the lightweight mesh M which is important for the initialization of Gaussian splats. Next, in Section 6.3.2., we present the mesh-based Gaussian splatting for training, which involves the removal of Gaussian splats (green Gaussian splats) occluded by mesh surface. Given the distance d_i between each Gaussian splat and its corresponding triangle, we distinguish between tightly-bound Gaussian splats (blue splat) and loosely Gaussian splats (red splat). In Section 6.3.3, we introduce the training and densification strategy for both tightly-bound Gaussian splats and loosely Gaussian splats.

Following the method in [2], we significantly reduce the number of triangles in the mesh to facilitate the integration with Gaussian splats to generate high-quality rendering. Next, we bind the Gaussian splats G to the explicit mesh to represent their appearance.

6.3.1 Geometry Reconstruction

During this stage, we leverage the neural surface representation to extract a mesh and decimate it for integrating the Gaussian splats with the mesh. The original 3D Gaussian Splatting method (3DGS) [10] is specifically tailored to sparse point clouds generated by Structure-from-Motion (SfM) techniques [29]. Dense point clouds are not conducive to rendering and can even be detrimental to the rendering quality. Hence, it is imperative to meticulously design mesh reconstructions for the initialization of Gaussian splats. Following the approach of BakeSDF [3], we jointly train two multilayer

perceptrons (MLPs) to encode both the signed distance field f_{sdf} and the appearance field g_{color}, as shown in the *Geometry Reconstruction* block in Fig. 6.1. Both MLPs are optimized using differentiable volumetric rendering and Eikonal regularization techniques [218]. Details are described in [3]. Then, we employ Marching Cube algorithm [82] to obtain a mesh $M = \{V, F\}$ with vertices V and faces F extracted from f_{sdf}.

However, in large-scale scenes, the output mesh occupies a significant number of triangles (average 27 millions) [3]. Given occupying a large amount of triangles, they still cannot represent detailed geometry or reflective surfaces due to the strong smoothness regularization. Furthermore, a large number of triangles are allocated to distorted regions, especially in background regions. Several prior mesh-based Gaussian splatting methods [4, 5, 265] assume that at least one Gaussian splat is attached to the mesh surface to represent the appearance of its triangles. If we strictly follow their strategy, we allocate redundant Gaussian splats to distorted regions, resulting in a rapid increase in GPU memory usage after just a few training and densification steps. In Fig.6.2 (a), the mesh reconstructed by BakedSDF occupies 44 million triangles to represent geometry. The original 3DGS [10] necessitates 4 million Gaussian splats to achieve high-fidelity rendering quality, yet the very large number of Gaussian splats undermines rendering quality. SuGaR also imposes constraints on the vertex count (10K$\sim$15K); however, a significant number of vertices are needed to faithfully represent the geometry, particularly in expansive, unbounded scenes. We observe that SuGaR sacrifices geometric quality, particularly in background regions (see Fig.6.2 (b)), and then trains Gaussian splats attached to mesh faces. However, since we aim to achieve moderate geometry quality, adopt the methodology of LTM [2]. They utilize aggressive mesh decimation based on the Quadric Error Metric [99, 245] to efficiently represent moderate geometry with a fewer number of triangles (see Fig. 6.2 (c)). Using this mesh as a foundation, we can easily integrate Gaussian splatting with the explicit mesh.

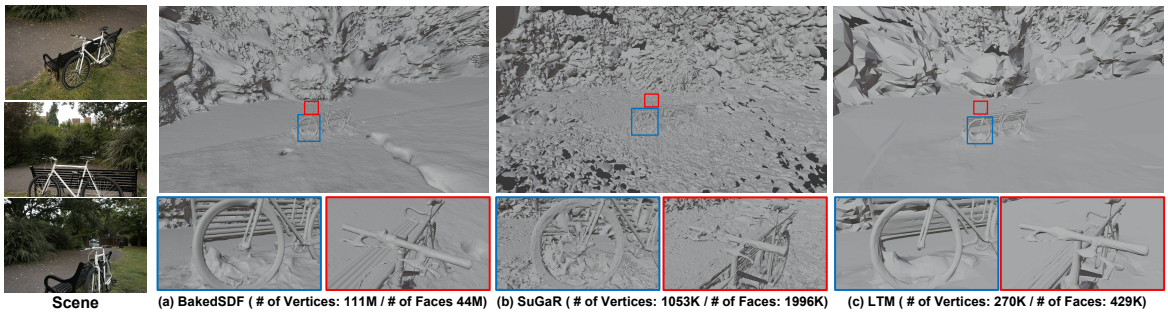


Figure 6.2: The examples of extracted mesh from BakedSDF [3], SuGaR [5], and LTM [2]. We employ LTM [2] to extract a lightweight mesh with moderate triangle count, using it for the initialization of Gaussian splats. The primary reason is that LTM utilizes a minimal number of triangles while maintaining geometric quality. Zoomed-in regions visualize that the mesh frequently lacks highly detailed geometry or complex structures. "K" and "M" denotes the 10^3 and 10^6 units, respectively.

6.3.2 Mesh-based Gaussian Splatting

3D Gaussian Splatting: 3D Gaussian Splatting (3D-GS) [10] employs millions of 3D Gaussians G to represent a 3D scene. These 3D Gaussian splats are constructed based on a sparse point cloud obtained from Structure from Motion [29], incorporating various attributes to model both the geometry and appearance of the scene. The geometry attributes of each 3D Gaussian (ellipsoids) G_i is defined by its center (position) $p_i \in \mathbb{R}^3$ and 3D covariance matrix $\Sigma_i \in \mathbb{R}^{3x3}$ in world space:

$$G_i(x) = e^{-\frac{1}{2}(x-p_i)^T \Sigma_i^{-1}(x-p_i)}. \tag{6.1}$$

To constrain valid covariance matrices, 3D-GS utilizes a semi-definite parameterization $\Sigma_i = R_i S_i S_i{}^T R_i{}^T$ with two learnable components: a scaling matrix $S_i \in \mathbb{R}^3$ and a rotation matrix $R_i \in \mathbb{R}^{3x3}$, encoded by a quaternion $q \in \mathbb{R}^4$. For rendering, given a viewpoint defined by rotation $R_{cw} \in \mathbb{R}^{3x3}$ and translation $t_{cw} \in \mathbb{R}^3$, the 3D Gaussians are projected into the camera space: $p_i^c = R_{cw} p_i + t_{cw}$ and $\Sigma_i^c = R_{cw} \Sigma_i R_{cw}{}^T$. Next, to transform camera space into ray space, 3D-GS utilize the affine transfor-

mation $J_i \in \mathbb{R}^{2x3}$ defined by p_i^c following [274], and the 2D covariance matrix in ray space can then be defined as $\Sigma_i' = J_i \Sigma_i^c {J_i}^T$. We can obtain the 2D Gaussian g_i defined by corresponding 2D covariance Σ_i' and 2D center p_i' as follows:

$$g_i(x) = e^{-\frac{1}{2}(x-p_i')^T {\Sigma_i'}^{-1}(x-p_i')}. \tag{6.2}$$

Additionally, each Gaussian includes an per-point opacity $o_i \in [0, 1]$ and the view-dependent color c_i represented by spherical harmonic (SH) [10]. 3D-GS orders all the Gaussians that contributes to a pixel and renders 2D image via alpha blending with the sorted Gaussians using the following equation:

$$c = \sum_{i=1}^{n} c_i \alpha_i \prod_{j=1}^{i-1}(1 - \alpha_j), \quad \alpha_i = o_i g_i(x). \tag{6.3}$$

The blending weight α_i is obtained by computing the 2D projection of the 3D Gaussian $g_i(x)$ multiplied by a per-point opacity o_i.

Distance-based Gaussian Splatting: In this section, we present our approach that builds the connection between large-scale mesh and Gaussian splats. Before training, we initialize a 3D Gaussian splats at the center m_j of each triangles f_j. Given mesh M, we can render a depth D_i from a given viewpoint P_i. When rendering for each viewpoint, we utilize the corresponding viewpoint depth map to mask out the Gaussian splats occluded by the surface. In light of our assumption that the mesh surface is entirely opaque, Gaussian splats occluded by it should be excluded for rendering purposes. In the *Splatting* block shown in Fig. 6.1, given that the depth d_k of the green Gaussian splats is greater than that of d_j, those Gaussian splats located behind the mesh surface do not contribute to rendering pixels. Furthermore, during the training process, we assess the visibility of Gaussian splats across all training views and identify those located behind the mesh surface hat are never observed by any of the training views. Then, we remove all redundant Gaussian splats situated behind the mesh surface at specific

training iterations.

Given that reconstructed meshes in real-world environments often exhibit geometric artifacts, our goal is to render not only well-represented regions but also those inaccuracies resulting from severe mesh artifacts. When we tightly bind all Gaussian splats exclusively to the mesh surface, rendering performance suffers, particularly when addressing severe mesh artifacts. Therefore, during the training step, We calculate the distance d_i between each Gaussian splat's center p_i and corresponding mesh faces f_j. We treat tightly-bound Gaussian splats G_i^t $(d_i < d_{th})$ and loosely-bound Gaussian splats G_i^l $(d_i > d_{th})$, distinctly. If the Gaussian splats are located at a distance greater than a certain threshold d_{th} from the mesh surface, we classify them as loosely-bound Gaussian splats. We aim to flatten G_i^t and align these splats with the mesh surface.

6.3.3 Training for Tightly-bound and Loosely-bound Gaussian Splats

Objective Function: Following 3D-GS [10], we employ the combination of L_1 loss and D-SSIM as the image loss L_{img} between the rendered image $\hat{I}_i$ and the ground truth image I_i expressed as follows:

$$L_{img} = (1 - \lambda_{img})L_1(\hat{I}_i, I_i) + \lambda_{img}SSIM(\hat{I}_i, I_i)). \tag{6.4}$$

We utilize the normal consistency loss L_{nc} to refine tightly-bound Gaussian splats G^t to align with the corresponding mesh surface. To estimate normals n_i for Gaussian splats, we first identify the minimum scale axis among the scale parameters S_i, and then apply the rotation matrix R_i to transform them into world coordinates. Then, we exploit the cosine similarity between the splat's normals n_i and

the corresponding face normals n_f as follows:

$$L_{nc} = \sum_{i \in G^t, f \in F} (1 - n_i \cdot n_f). \tag{6.5}$$

Furthermore, we apply the regularization to suppress the scale parameters $S_i = (s_1, s_2, s_3) \in \mathbb{R}^3$ of tightly-bound Gaussian splats G^t and align them with the mesh surface. We regularize the minimum scale and then the maximum scale with threshold value ρ.

$$L_{scale} = \sum_{i \in G^t} (\lambda_{min} |\text{min}(s_1, s_2, s_3)| + \lambda_{max} |\text{max}(s_1, s_2, s_3) - \rho|). \tag{6.6}$$

This ensures that the tightly-bound Gaussian splats do not expand excessively in comparison to the corresponding mesh face f_j, thus hindering the rendering process. Finally, given the mesh M consisting of face f_j and its face normal n_j, we project the Gaussian Splat's center P_i to the closest point s_i on the mesh surface.

$$s_i = P_i + ((P_i - v_i) \cdot f_j) \cdot f_j, \quad L_{proj} = \sum_{i \in G^t} \parallel s_i - P_i \parallel_2 . \tag{6.7}$$

This loss function ensures that the tightly-bound Gaussian splats remain close to the mesh surface, preventing them from moving too far from the surface. The total loss function L is defined as:

$$L = L_{img} + \lambda_{nc} L_{nc} + L_{scale} + \lambda_{proj} L_{proj}. \tag{6.8}$$

where λ denotes the weight used to balance between the different loss terms. **Densification Strategy** Here, we adopt two different densification strategies for loosely-bound Gaussian splats G^l

and tightly-bound Gaussian splats G^t. Regarding G^l, we follow the method outlined in the original 3DGS [10]. The original Gaussian densification strategy inherently overlooks the existing 3D geometry. In terms of G^t, we find the mesh triangle close to G^t and generate new Gaussian splats by employing random sampling from the corresponding mesh face. For both cloning and splitting, we conduct point sampling from the triangle and utilize this position to initialize new Gaussian splats. During cloning, additional attributes are copied to the new Gaussians, while during splitting, their scale is divided by a factor of 1.6.

6.4 Experiments

6.4.1 Experiment Settings

Datasets: For comparison, we assess the rendering quality of our method using the extensively utilized real-world large-scale scenes published by Mip-NeRF360 [88]. Furthermore, to assess the generalization capability, we employ a different large-scale dataset which are two scenes from Deep Blending [6].

Implementation Details: For geometry reconstruction section in Sec 6.3.1, we utilize the LTM [2] code for extracting mesh from neural field and decimating those mesh. In Sec 6.3.2, when applying distance-based Guassian splatting, we add the 0.01 to the depth D_i. We prune the Gaussian splats located behind the mesh at every 500 training iterations. To distinguish the tightly-bound and loosely-bound Gaussian splats, we set the distance threshold d_{th} to 0.01. For training Gaussian splatting, we generally follow the original 3DGS paper [10]. we run all experiments for 30k iterations. We set the loss weights to λ_{img}=0.001, λ_{nc}=0.1, λ_{min}=0.1, λ_{max}=10, and λ_{proj}=50. We set the threshold ρ to 0.1. Following the train/test split in Mip-NeRF360, we subsampled every eight frames for test sets. We evaluate all our method trained with 30K iterations.

6.4.2 Experimental Results

We compare our proposed method with three different approaches: LTM [2], which represents the state-of-the-art method for reconstructing lightweight meshes and training a neural field for the corresponding mesh. SuGaR [5], which builds upon the original 3DGS [10] by reconstructing the mesh and binding the Gaussian splats to the corresponding mesh. 3DGS [10], which relies on sparse point clouds instead of an explicit mesh representation. As BakedSDF [3] has not released public code, we include only metrics reported in their original paper.

In Table 6.1, we report the rendering quality using the standard PSNR, SSIM [163], and LPIPS [261] on outdoor scenes in mip-NeRF 360 dataset [88]. Generally, mesh-based Gaussian splatting methods, including our method, SuGaR [5], and GaMeS [4], exhibit superior performance (with around 2 PSNR increase) compared to state-of-the-art mesh-based neural rendering techniques such as LTM [2], BakedSDF [3], and others. Our method also demonstrates better quality compared to other recent mesh-based Gaussian splatting methods such as SuGaR [5] (with a 1.3 PSNR increase) and GaMeS [4] (with a 0.9 PSNR increase). Table 6.2 reports the rendering quality on indoor scenes in mip-NeRF 360 dataset [88]. Our method also exhibits higher rendering quality compared to GaMeS [4] (with a 2.0 PSNR increase) and SuGaR [5] (with a 0.2 PSNR increase). Overall, our method shows slightly lower rendering quality compared to the original 3DGS. Without considering geometry, 3D Gaussian splats are more flexible in representing the scene based solely on image loss. This is why most mesh-based splatting methods are not superior to the original 3DGS. Our goal is to find a balance between rendering quality and geometric alignment. Thus, we emphasize the performance by solely comparing mesh-based neural rendering and mesh-based Gaussian splatting.

Furthermore, we train and evaluate on Playroom and DrJohnson scenes from the Deep Blending

Table 6.1: **Quantitative Comparison on outdoor mip-NeRF 360 Dataset.** Splat-based rendering demonstrates superior performance compared to mesh-based neural rendering techniques such as LTM [2] and BakedSDF [3]. MeshGS* did not utilize any regularization techniques to tightly align Gaussian splats; instead, it solely applied image loss L_{img} for training. Compared to GaMeS [4] and SuGaR [5], our method enhances the PSNR by 0.9 dB and 1.3 dB, respectively.

Method	Mesh	Rendering	Outdoor (PSNR ↑ / SSIM ↑ / LPIPS ↓)			
			Bicycle	Garden	Stump	Mean
Instant-NGP [196]	X	Volume	22.1 / 0.49 / 0.49	24.5 / 0.65 / 0.31	23.6 / 0.57 / 0.45	23.4 / 0.57 / 0.42
mip-NeRF 360 [88]	X	Volume	24.4 / 0.68 / 0.30	26.9 / 0.81 / 0.17	26.4 / 0.74 / 0.26	25.9 / 0.75 / 0.24
3DGS [10]	X	Splat	25.2 / 0.77 / 0.20	27.4 / 0.86 / 0.10	26.5 / 0.77 / 0.21	26.4 / 0.70 / 0.17
MobileNeRF [43]	O	Mesh	21.7 / 0.43 / 0.51	18.8 / 0.59 / 0.36	23.9 / 0.56 / 0.43	21.5 / 0.53 / 0.43
NeRF2Mesh [42]	O	Mesh	22.1 / 0.48 / 0.51	23.4 / 0.55 / 0.40	22.5 / 0.54 / 0.46	22.7 / 0.52 / 0.46
BakedSDF [3]	O	Mesh	22.0 / 0.57 / 0.37	24.9 / 0.75 / 0.21	23.6 / 0.59 / 0.37	23.5 / 0.64 / 0.32
LTM [2]	O	Mesh	22.4 / 0.52 / 0.44	23.5 / 0.64 / 0.30	23.7 / 0.57 /0.42	23.2 / 0.58 / 0.39
SuGaR [5]	O	Splat	23.1 / 0.64 / 0.34	25.3 / 0.77 / 0.22	24.7 / 0.68 / 0.34	24.4 / 0.70 / 0.30
GaMeS [4]	O	Splat	23.4 / 0.67 / 0.33	26.3 / 0.83 / 0.14	24.6 / 0.66 / 0.35	24.8 / 0.72 / 0.39
MeshGS (Ours)	O	Splat	24.4 / 0.71 / 0.26	26.8 / 0.85 / 0.12	25.8 / 0.74 / 0.25	25.7 / 0.77 / 0.21
MeshGS* (Ours)	O	Splat	**24.9 / 0.75 / 0.25**	**26.8 / 0.86 / 0.12**	**26.3 / 0.76 / 0.22**	**26.0 / 0.79 / 0.20**

dataset [6]. These two scenes are collected from bounded indoor scenes and have many complex textures on Playroom scene and contain many textureless regions on DrJohnson scene. These two scenes were captured from bounded indoor environments, with the Playroom scene consisting intricate textures and the DrJohnson scene comprising several textureless regions. In Table 6.3, Our method demonstrates comparable rendering quality to both the original 3DGS [10] and SuGaR [5].

In Fig. 6.3, we visualize the rendering results for comparison. Our method builds on the mesh reconstructed by LTM [2]. Compared to LTM, the Gaussian splatting in our method exhibits sharper and more high-frequency details than the neural appearance field. As meshes struggle to represent highly detailed geometry such as dense foliage, tree leaves, and thin iron structures, LTM may not adequately capture these appearances for rendering (see the zoomed-in regions in Fig. 6.3). SuGaR is based on the original 3DGS and reconstruct the mesh solely based on Gaussian splats. However, compared to our geometry reconstruction, the mesh generated by SuGaR exhibits numerous geometric artifacts, particularly in the background region, where defects are prevalent due to weak smoothness regulariza-

Table 6.2: **Quantitative Comparison on indoor mip-NeRF 360 Dataset.** Splat-based rendering exhibits superior performance when compared to mesh-based neural rendering techniques. MeshGS* did not utilize any regularization techniques to tightly align Gaussian splats; instead, it solely applied image loss L_{img} for training. Compared to GaMeS [4] and SuGaR [5], our method enhances the PSNR by 2.0 dB and 0.2 dB, respectively.

Method	Mesh	Rendering	Indoor (PSNR ↑ / SSIM ↑ / LPIPS ↓)				
			Room	Counter	Kitchen	Bonsai	Mean
Instant-NGP [196]	X	Volume	29.2 / 0.85 / 0.30	26.43 / 0.80 / 0.34	28.5 / 0.82 / 0.25	30.3 / 0.89 / 0.23	28.6 / 0.84 / 0.28
mip-NeRF 360 [88]	X	Volume	31.6 / 0.91 / 0.21	29.5 / 0.89 / 0.20	32.3 / 0.92 / 0.13	33.5 / 0.94 / 0.18	31.7 / 0.92 / 0.18
3DGS [10]	X	Splat	30.6 / 0.91 / 0.26	28.7 / 0.90 / 0.29	30.3 / 0.92 / 0.22	31.9 / 0.94 / 0.29	30.4 / 0.92 / 0.19
MobileNeRF [43]	O	Mesh	28.9 / 0.85 / 0.28	25.1 / 0.72 / 0.29	26.8 / 0.79 / 0.79	23.8 / 0.71 / 0.72	26.2 / 0.77 / 0.52
NeRF2Mesh [42]	O	Mesh	25.7 / 0.79 / 0.35	23.9 / 0.71 / 0.35	24.0 / 0.61 / 0.36	25.0 / 0.77 / 0.29	24.7 / 0.72 / 0.34
BakedSDF [3]	O	Mesh	28.7 / 0.87 / 0.25	25.7 / 0.81 / 0.28	26.7 / 0.82 / 0.24	27.2 / 0.85 / 0.26	27.0 / 0.84 / 0.26
LTM [2]	O	Mesh	29.3 / 0.88 / 0.23	25.1 / 0.77 / 0.27	26.5 / 0.80 / 0.21	27.3 / 0.84 / 0.22	27.1 / 0.82 / 0.23
SuGaR [5]	O	Splat	30.0 / 0.91 / 0.25	27.6 / 0.89 / 0.23	29.6 / 0.91 / 0.16	30.5 / 0.93 / 0.22	29.4 / 0.91 / 0.22
GaMeS [4]	O	Splat	28.8 / 0.89 / 0.26	26.4 / 0.84 / 0.29	27.2 / 0.86 / 0.22	27.8 / 0.89 / 0.29	27.6 / 0.87 / 0.26
MeshGS (Ours)	O	Splat	30.3 / 0.90 / 0.24	**28.0** / 0.89 / **0.21**	**29.7** / **0.90** / **0.17**	**30.7** / **0.93** / 0.20	**29.6** / **0.91** / **0.20**
MeshGS* (Ours)	O	Splat	**30.9** / **0.93** / **0.23**	27.8 / **0.90** / 0.21	29.3 / 0.90 / 0.17	30.5 / 0.93 / **0.19**	29.6 / 0.91 / 0.20

Table 6.3: **Quantitative Comparison on Deep Blending Dataset.** We report PSNR ↑, SSIM ↑, and LPIPS ↓ on Deep Blending datasets [6]

Method	Mesh	Rendering	Deep Blending [6]	
			DrJohnson	Playroom
Instant-NGP [196]	X	Volume	27.7 / 0.84 / 0.38	19.5 / 0.78 / 0.46
mip-NeRF360 [88]	X	Volume	29.1 / 0.90 / 0.24	29.6 / 0.90 / 0.25
3DGS [10]	X	Splat	29.1 / 0.90 / 0.24	30.0 / 0.90 / 0.24
MobileNeRF [43]	O	Mesh	25.9 / 0.78 / 0.31	28.7 / 0.85 / 0.28
NeRF2Mesh [42]	O	Mesh	24.0 / 0.75 / 0.51	26.6 / 0.83 / 0.41
LTM [2]	O	Mesh	26.5 / 0.82 / 0.40	29.1 / 0.87 / 0.32
SuGaR [10]	O	Splat	28.6 / 0.88 / 0.27	30.3 / 0.89 / 0.26
MeshGS (Ours)	O	Splat	**28.8 / 0.90 / 0.24**	**30.5 / 0.90 / 0.25**

tion (see the background region in Fig. 6.2 and Fig. 6.4). Due to the geometric artifacts present in the background region, SuGaR often produces blurry rendering results, especially in unbounded outdoor scenes (see the background regions of Bicycle, Garden, and Stump in Fig. 6.3).

6.4.3 Ablation Study

In Table 6.1 and Table 6.2, MeshGS* (ours) denotes we only use image loss L_{img} to train our mesh-based Gaussian splatting in Section 6.3.3. Compared to MeshGS (ours), MeshGS* did not employ any regularization technique to tightly align Gaussian splats with the mesh surface. We observe



Figure 6.3: **Qualitative Comparisons with Existing Methods.** We visually compare our method with SuGaR [5], 3DGS [10], and LTM [2]. For a detailed description, we visualize the zoomed-in blue and red regions.

that MeshGS* shows slightly better rendering quality compared to our method. This is a trade-off between aligning splats geometrically and achieving high-quality rendering.

In Table 6.5, we show the proportion of tightly-bound Guassian splats and loosely-bound Gaussian splats. Compared to MeshGS*, our approach employs fewer Gaussian splats overall. Furthermore, our method utilizes Gaussian splats that are 6% less tightly bound compared to MeshGS*. It is noteworthy that the Stump scene generates a significantly higher proportion of loosely-bound Gaussian splats (42%) compared to other scenes (22%). In the Stump scene, we observe that the majority of background regions are covered by loosely-bound Gaussian splats. This is due to the inability of the mesh to accurately cover all background regions, causing Gaussian splats to often move far from the

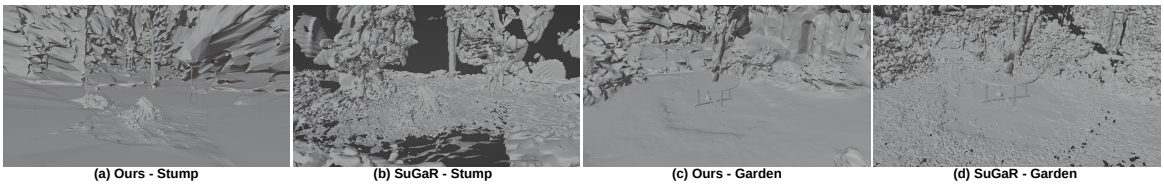
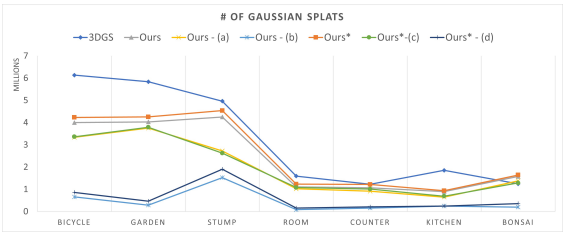


Figure 6.4: **Qualitative Comparisons of Our Method and SuGaR [5]**. We show shading mesh without texture.



# of Splats	Bicycle	Garden	Stump	Room	Counter	Kitchen	Bonsai	Mean
3DGS [10]	6132	5834	4961	1593	1222	1852	1244	3263
MeshGS	3792	3838	4252	1103	1067	893	1576	2360
MeshGS-(a) (Tightly-bound Splats)	3236	3651	2723	1024	912	647	1378	1939
MeshGS-(b) (Loosely-bound Splats)	556	187	1522	82	156	245	198	421
MeshGS*	4231	4255	4537	1230	1215	934	1640	2577
MeshGS*-(c) (Tightly-bound Splats)	3359	3788	2626	1077	1010	688	1283	1976
MeshGS*-(d) (Loosely-bound Splats)	859	467	1904	154	206	244	356	598

Figure 6.5: **The number of Gaussian Splats** The unit for the number of Gaussian splats is 10^3. Compared to the original 3DGS, we utilize 30% fewer Gaussian splats. MeshGS* did not utilize any regularization techniques to tightly align Gaussian splats; instead, it solely applied image loss L_{img} for training. MeshGS-(a) and MeshGS*-(c) shows the number of tightly-bound Gaussian splats. MeshGS-(b) and MeshGS*-(d) denotes the number of loosely-bound Gaussian splats.

mesh surface.

Figure 6.6 presents a visualization of both loosely-bound and tightly-bound Gaussian splats using the outdoor mip-NeRF 360 dataset. In Fig. 6.6-(a), tightly-bound Gaussian splats can cover the most regions for rendering. Nevertheless, due to limitations in mesh geometry, they fail to address all mesh artifacts. In Fig. 6.6, the red dotted box highlights regions where tightly-bound Gaussian splats cannot accurately represent details such as the bicycle wheel, thin trunk around the vase, and dense tall grass, which closely resemble mesh artifacts. Loosely-bound Gaussian splats can serve as a supplementary method to cover these artifacts. In the Stump scene, loosely-bound Gaussian splats demonstrate their ability to accurately depict the detailed thin grass and leaves covering the trunk, thereby facilitating high-fidelity rendering. Moreover, given that the mesh geometry shows more geometric defects in the background region, it necessitates the use of numerous loosely-bound Gaussian splats to



Figure 6.6: **Visualization results of both loosely-bound and tightly-bound Gaussian splats.** The second column shows wireframe mesh extracted without texture. The third column visualizes only the tightly-bound Gaussian splats, while the fourth column illustrates the loosely-bound Gaussian splats. The fifth column presents the final rendering results achieved by our method.

achieve high-quality rendering. Furthermore, in the mesh geometry, the process of mesh decimation significantly reduces the number of triangles in the background region. As a result, the initialization of Gaussian splats becomes more sparse, posing challenges in rendering images.

6.4.4 Compositional scenes

In Fig. 6.7, we visualize the practical application of our method for scene composition. We incorporate an artist-designed human model from Mixamo [275] into the scene reconstructed by our method. This human asset consists of high-polygon meshes with photorealistic materials and textures. We show the composition of a synthetic human into a different scene. The background region rendered by our method shows photorealistic image quality of compositional scenes. Thanks to the mesh, we can composite the object and scene. Furthermore, the composited scene can be rendered using Gaussian splatting without causing severe artifacts around the human's boundary.

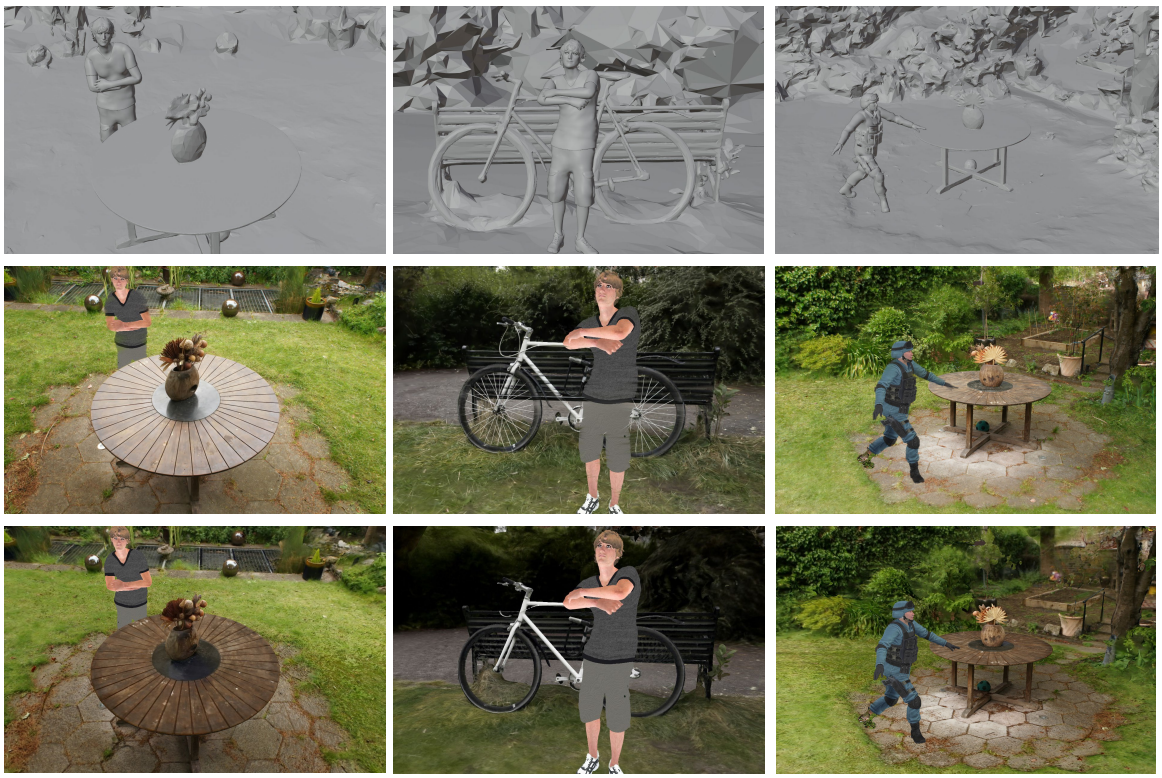


Figure 6.7: **Scene Composition**: Rendering obtained by embedding the synthetic human to a reconstructed scene. The first row visualizes the shading mesh without texture. The second row shows the composition of a human in the scene reconstructed by our method. The third row visualizes the composition of the same human in the scene generated by LTM [2].

6.5 Conclusion

This paper presents a practical approach for integrating mesh representation with 3D Gaussian splatting rendering. our method attains a high level of photorealistic rendering quality by employing Gaussian splats bound to the mesh surface and executing distance-based Gaussian splatting to ensure alignment with the mesh surface. Additionally, we differentiate between two types of Gaussian splats and present training strategies tailored to their respective roles. We present that a combination of mesh-based representation and Gaussian splatting is crucial for enhancing rendering quality and ensuring compatibility with game engines in practical applications. However, our method does have

limitations. Compared to textured mesh, our approach necessitates large disk storage and a high-end GPU for real-time rendering. Moreover, we require a more sophisticated method for assessing the quality of Gaussian splats and ensuring their tight binding to the surface mesh. Additionally, our method is constrained by the underlying mesh shape, particularly when dealing with excessively large faces that cannot appropriately initialize Gaussian splats. In the future, our goal is to develop surface extraction techniques from loosely-bound splats to address mesh defects and evaluate geometric quality. Furthermore, we should explore adaptive methods for distributing Gaussian splats to individual triangles, akin to mesh subdivision or remeshing techniques.

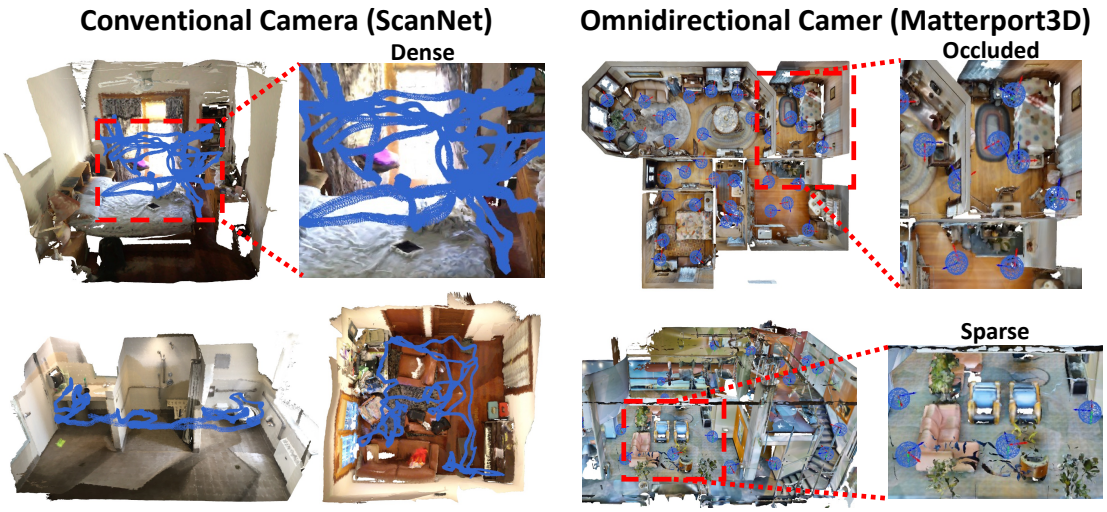
Chapter 7: IM360: Large-scale Indoor Mapping with 360° Cameras

7.1 Introduction

Indoor 3D mapping and photorealistic rendering are core technologies for various applications in computer vision, robotics, and graphics. High-fidelity digitization of the real world enables immersive experiences in AR/VR and helps bridge the sim-to-real gap in robotic applications. However, the conventional image acquisition process [44, 87, 88] is often labor-intensive and time-consuming due to the limited field-of-view and sparse scene coverage, as shown in Fig 7.1. Previous studies [276, 277] have demonstrated that omnidirectional cameras can significantly reduce acquisition time, which is crucial for capturing various indoor environments. Nevertheless, most recent research [9, 10, 29, 85, 278] relies on conventional cameras with limited fields of view, which results in inadequate scene coverage, susceptibility to motion blur, and substantial time requirements for capturing large-scale indoor environments.

Unlike conventional cameras, which benefit from well-established photogrammetric software such as COLMAP [29] and OpenMVG [278], research on omnidirectional cameras remains fragmented across various domains and lacks a robust pipeline for Structure-from-Motion (SfM) and photorealistic rendering. Notably, there has been limited academic exploration of sparse scanning scenarios aimed at significantly reducing image acquisition time. There are two key challenges in developing 3D mapping and rendering for omnidirectional cameras.

First, the SfM pipeline in sparsely scanned indoor environments suffer from the prevalence of large textureless regions, which degrade the accuracy of both feature detection and matching. Moreover, frequent occlusions in indoor environments, combined with sparse input views as shown in Fig. 7.1, make the feature matching process even more challenging. In such scenarios, conventional per-



Dataset	# Scenes	# Images	Floor Space (m^2)	View Density (images/m^2)
Stanford2D3D [8]	6	1,413 (Omnidirectional)	6020	**0.23**
Matterport3D [7]	90	10,800 (Omnidirectional)	46,561	**0.23**
ScanNet [44]	707	2,492,518 (Conventional)	34,453	72.35

Figure 7.1: While the wide field of view of the omnidirectional camera significantly reduces the data acquisition time, it introduces challenges such as sparse views and occlusions. According to the Table, Matterport3D [7] and Stanford2D3D [8] are captured with significantly lower view density compared to ScanNet [44]. Our approach provides an effective solution to handle issues in sparsely scanned indoor environments and enables efficient reconstruction and rendering using omnidirectional cameras.

spective cameras further worsen the problem due to their limited field of view, which reduces the overlap between views. This lack of overlap complicates the estimation of accurate camera poses, making the reconstruction task considerably more difficult. Recently, detector-free or dense matchers [279–284] have emerged as a promising alternative to detector-based approaches [129, 285, 286], particularly in handling repetitive or indiscriminate regions where keypoint detection performance tends to degrade.

Additionally, in sparsely scanned scenarios, recent neural rendering methods [10, 85] often face challenges in producing high-quality novel-view images. It is widely recognized that both neural radiance fields (NeRF) [85] and 3D Gaussian splatting (3DGS) [10] are optimized for densely captured

scenes with substantial overlap between viewpoints. Prior studies [48, 287–289] have demonstrated that neural rendering algorithms often exhibit overfitting to the training views and a poor rendering quality on novel-view images far from training views.

Main Results We present a novel indoor 3D mapping and rendering pipeline designed for omnidirectional images and address the challenges of sparsely scanned indoor environments. We tackle the problem of extensive textureless regions by employing detector-free matching methods [101]. To mitigate the frequent occlusions encountered in sparsely scanned images, we utilize an spherical camera model with a wide field of view, increasing the overlap between images and enhancing robustness in multi-view matching. While traditional methods, such as OpenMVG [278] and COLMAP [29], fail to register nearly half of the images in challenging datasets like Matterport3D [7] and Stanford2D3D [8], our method achieves unparalleled registration performance with high pose accuracy, thanks to the tight integration of a wide-view spherical model. To the best of our knowledge, ours is the first work to present a complete SfM pipeline dedicated to spherical camera models, where entire step—feature matching, two-view geometry estimation, track triangulation, and bundle adjustment—is conducted on a spherical manifold.

We extend our approach to mesh reconstruction and texturing and provide a comprehensive solution for indoor 3D mapping and rendering. For mesh reconstruction, we leverage neural surface reconstruction [290] by training a signed distance field and extracting a mesh. Classical texture mapping [9] is then combined with texture optimization through differentiable rendering. We optimize textures and train small multi-layer perceptrons (MLPs) to represent specular color. By jointly optimizing diffuse and specular textures along with the MLPs via differentiable rendering [84] under image supervision, our approach achieves superior rendering quality. Additionally, it demonstrates robustness in sparsely

scanned indoor environments, outperforming recent neural rendering methods. The proposed method demonstrates significant improvements over traditional approaches, particularly in challenging indoor environments with sparse and textureless scenes.

We evaluate our approach using the Matterport3D dataset [7], which offers a larger physical scale than typical indoor datasets like ScanNet, according to [291]. We also evaluate the performance on the Stanford2D3D datasets [8]. The novel contributions of our work include:

- We introduce a unified framework for reconstructing textured meshes using omnidirectional cameras in sparsely scanned, large-scale indoor scenes.
- We propose the first spherical Structure-from-Motion approach utilizing dense matching features specifically designed for omnidirectional cameras.
- We integrate classical texture mapping with neural texture fine-tuning through differentiable rendering, demonstrating improved rendering quality, in sparsely scanned, in-the-wild indoor environments.

We compare our method with SOTA and observe higher accuracy in terms of camera localization and rendering. In particular our approach can render higher frequency details and results in lower noise in the reconstructed models.

7.2 Related Work

7.2.1 Structure from Motion

Structure from Motion (SfM) is a fundamental problem in computer vision. Traditional SfM frameworks typically begin with the detection of keypoints and descriptors, which serve as a core

component, ranging from classical algorithms [292] to learning-based techniques [129,285,293]. Then, they match these keypoints by either nearest neighbor [29] or learning-based methods [293]. The matching pair is verified by recovering the two-view geometry [97] with RANSAC [294]. Based on these matching pairs, popular SfM methods [29, 278, 295] follow incremental approach, sequentially registering new images and reconstructing their 3D structures through triangulation [97] and bundle adjustment [296]. Alternatively, a recent work [297] employs a global approach, recovering the camera geometry for all input images simultaneously. The 2D correspondence matching significantly impact the overall performance of SfM pipeline. In textureless region of indoor environment, these methods suffer from poor feature detection [298]. Detector-free or dense matching methods [279, 283, 284] is proposed to solve this issue by estimating dense feature matches at pixel level. DetectorFreeSfM [298] builds these detector-free matches [279] and refines the tracks and geometry of the coarse SfM model by enforcing multi-view consistency.

Conversely, there are relatively few open-source software packages [278, 299] that support omnidirectional cameras. These frameworks typically employ an incremental SfM workflow based on a spherical camera model [299]. However, existing methods lack a comprehensive pipeline specifically designed for omnidirectional images. Recently, the first dense matching method for omnidirectional images, EDM [101], has been introduced. Building upon this feature matching and two-view geometry estimation [300], we propose a complete pipeline for spherical SfM to address the challenges inherent in indoor scene reconstruction.

7.2.2 Textured Mesh Reconstruction

Textured mesh is the basic component of photorealistic rendering. Traditional computer vision has developed effective techniques for mapping between texture and geometry. Classical texture mapping [9, 207, 301] employs a Markov Random Field to select the optimal color image for each mesh face and applies global color adjustment for consistency [9]. Recently, numerous neural rendering techniques have been developed for surface reconstruction [21, 39, 223, 290] and rendering [10, 49, 85, 88]. Notably, some approaches leverage differentiable rendering [41, 84, 216, 253, 254] to learn neural representations of texture, lighting, and geometry. Additionally, certain NeRF variants [2, 43, 238, 240, 241] employ differentiable volumetric rendering with mesh primitives, which is similar to rasterization-based rendering. Some neural rendering methods are tailored for omnidirectional cameras. EgoNeRF [302] utilizes a spherical feature grid to represent a distant environment for rendering. OmniSDF [45] introduces an adaptive spherical binoctree method for surface reconstruction. However, in sparsely scanned indoor environment, which contains lots of textureless regions and sparse viewpoints, most neural rendering methods struggle with severe artifacts. To address this limitation, we propose a hybrid approach that combines classical texture mapping with neural texture optimization.

7.3 Method

Given a set of equirectangular projection (ERP) images, our goal is to reconstruct the geometry and texture of real-world scenes. In Fig. 7.2, our pipeline is composed of three primary components: Spherical Structure from Motion (SfM), geometry reconstruction, and texture optimization. To address challenges associated with limited overlap and frequent occlusions in indoor environments, our method

leverages the wide field of view inherent in spherical imagery. We perform SfM utilizing a spherical camera model and dense features to estimate camera poses from sparsely scanned images in textureless regions. However, since a naive combination of SfM and dense feature matching is not directly feasible, we refine this approach to ensure compatibility. For 3D geometric reconstruction accurately, we converted ERP images into cubemaps and conducted training on the resulting perspective images. Then, we employ a neural model [290] that represents a signed distance function (SDF). After training the model, we extract the mesh using the Marching Cubes algorithm [82] and apply classical texture mapping [9] to initialize texture. Finally, we optimize the texture for diffuse, specular and small MLP components using differentiable rendering [84].

7.3.1 Dense Matching and Structure from Motion

Large-scale indoor scenes often pose significant challenges for camera localization using conventional SfM methods such as COLMAP [29] or OpenMVG [278], due to numerous occlusions and textureless regions. To effectively reconstruct sparsely captured indoor scenes, we integrate a spherical camera model into every step of SfM pipeline, combining spherical dense matching, two-view geometry estimation, and optimization methods into a unified, comprehensive 3D reconstruction framework.

We construct our pipeline based on the unit bearing vector $u \in \mathbb{S}^2$,

$$u = (\sin(\theta)\cos(\phi),\ \sin(\phi),\ \cos(\theta)\cos(\phi)) \tag{7.1}$$

where $\theta \in [-\pi, \pi]$ and $\phi \in [-\frac{\pi}{2}, \frac{\pi}{2}]$.

Spherical Dense Matching Performing spherical SfM requires feature matching as a preliminary step. Several feature matching methods specifically designed for 360° images have been proposed,

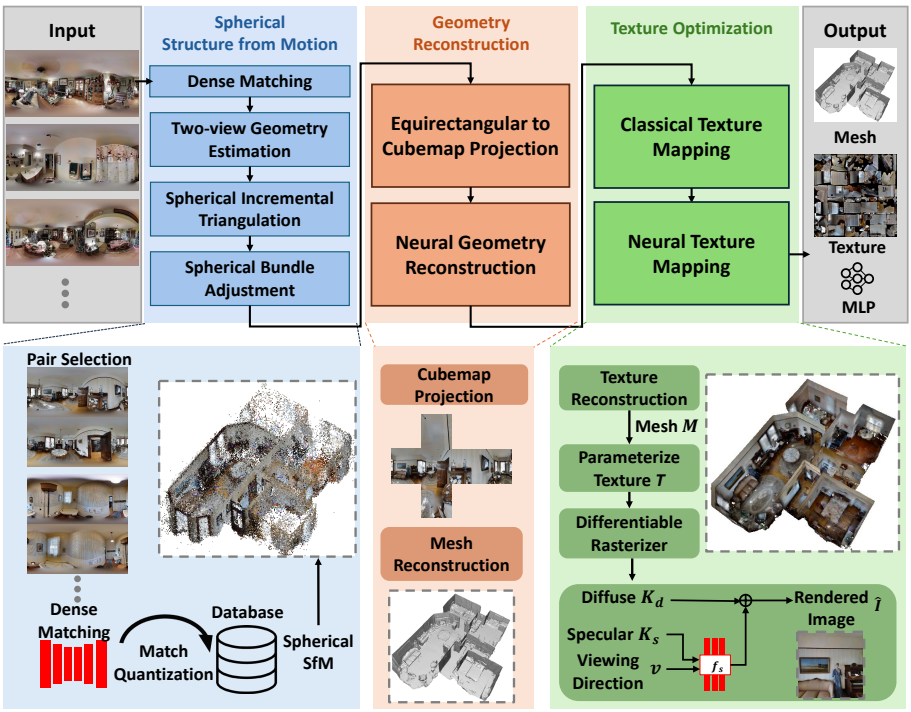


Figure 7.2: **Overview of Our Pipeline**. It is comprised of three steps: Spherical Structure from Motion (Sec. 7.3.1), Geometry Reconstruction (Sec. 7.3.2), and Texture Optimization (Sec. 7.3.3). In Section 7.3.1, spherical SfM is utilized to estimate the camera poses for omnidirectional images in challenging indoor environments. Then, in Section 7.3.2, our method apply cubemap projection and estimate monocular depth and normal to train geometry f_{sdf} representation using volumetric rendering. In Section 7.3.3, we initlize texture using classical texture mapping and jointly optimize diffuse, specular texture along with small MLP for modeling appearance.

such as SPHORB [303], SphereGlue [304], and EDM [101]. Alternatively, feature matching can also be performed independently on each face of a cubemap generated from a 360° image. Among these approaches, dense feature matching [101] recently demonstrated superior feature matching performance for 360° images, especially in sparse scenes with large viewpoint changes. However, since dense matching methods do not explicitly provide feature descriptors, constructing feature tracks of multiple views for SfM becomes challenging. Inspired by [298,305], we apply the additional strategies for match quantization by rounding [306] and merging [307] techniques, to ensure multiview match consistency.

Spherical Two-view Geometry Estimation It can be formally proven that the traditional two-view epipolar constratint for normalized coordinates extends to unit bearing vectors in spherical images [308],

$$u_1^T E u_2 = 0 \text{ with } E = R[t]_\times, \tag{7.2}$$

where $E \in \mathbb{R}^{3\times3}$ is the essential matrix, which can be decomposed into the camera rotation $R \in$ SO(3) and the translation $t \in \mathbb{R}^3$. Solarte *et al.* [300] employed the 8-point algorithm to solve spherical two-view geometry using the Direct Linear Transform (DLT) framework [97]. We adopt this two-view geometry estimation method with additional normalization [300] to enhance numerical stability during Singular Value Decomposition (SVD) computation. Using those initial pose with rotation R and translation t, we activate incremental triangulation and bundle adjustment under the sphere camera model.

Spherical Bundle Adjustment By representing each feature track observation as a bearing vector u, conventional Bundle Adjustment (BA) can be reformulated in a manner similar to the perspective case, as follows:

$$L = \sum_i \sum_j \rho(||\Pi(P_j; R_i, t_i) - u_{ij}||^2), \tag{7.3}$$

where ρ denotes the robust loss with a soft L1 function in local bundle adjustment, while global bundle adjustment is performed without this robustifier. $\Pi(\cdot)$ represents the spherical projection $\Pi : P \to u$, $\mathbb{R}^3 \mapsto \mathbb{S}^2$, which maps the 3D points X_j from triangulation to unit bearing vectors, given the camera poses $R_j \in$ SO(3) and $t_j \in \mathbb{R}^3$.

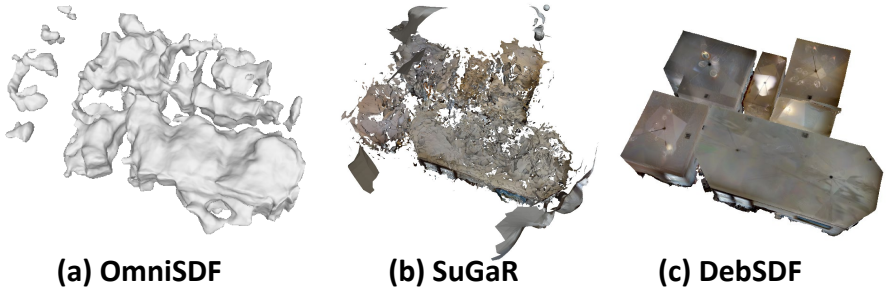


Figure 7.3: Our method, which applies cubemap projection and trains neural surface reconstruction using multiple perspective images, achieves highly accurate geometry reconstruction. In contrast, neural surface reconstruction designed for ERP images (OmniSDF [45]) and Gaussian Splatting-based surface reconstruction (SuGaR [46] and VCR-GauS [47]) fail to generate a clean mesh.

7.3.2 Geometric Reconstruction

To address visual localization and mapping in textureless and highly occluded sparsely scanned indoor scenes, we use neural rendering techniques [223, 290] for surface reconstruction instead of traditional 3D reconstruction methods such as multi-view stereo (MVS) [30]. Many methods leverage monocular geometric priors [223, 290] and demonstrate higher robustness in textureless and sparsely covered areas. However, in terms of ERP images there are no robust methods for zero-shot monocular depth and normal estimation. As shown in Fig. 7.3, we also observed that OmniSDF [45], which learns signed distance functions (SDF) from equirectangular projection (ERP) images, failed to converge when applied to large-scale indoor datasets. Despite incorporating geometric priors [309], Gaussian Splatting-based surface reconstruction [46, 47] still fails to produce reasonable geometry, resulting in a polygonal soup.

To address these limitations, we first project ERP images onto a cubemap representation and then apply a volumetric rendering approach to estimate the signed distance function (SDF). Since most existing mesh rendering techniques—except for volumetric methods leveraging monocular geometric

priors [223, 290]—struggle in sparse indoor scenes, we employ DebSDF [290] for geometric mesh reconstruction and integrate the Marching Cubes algorithm [82] to generate a high-fidelity mesh.

7.3.3 Texture Optimization

In this stage, we project textures onto the 3D mesh M generated in the preceding step. In comparison to recent neural rendering methods such as NeRF [85] and 3DGS [10], it is important to highlight that texture mapping provides a more robust solution for novel view synthesis in sparsely scanned indoor environments. Utilizing images obtained from cubemap projections along with their corresponding camera poses, we initially employ TexRecon [9] to construct the texture map T. However, this conventional texture mapping approach suffers from visible seams between texture patches and is prone to geometric inaccuracies.

Inspired by TMO [21], we parameterize the texture map and leverage a differentiable rasterizer $\mathcal{R}$ [84] to render an image from a given viewpoint P. While TMO [21] is constrained to representing only diffuse textures and fails to capture view-dependent effects, our approach surpasses these limitations by enabling a more comprehensive and physically accurate representation. The diffuse texture $K_d \in \mathbb{R}^3$ is directly converted into an RGB image. Additionally, we initialize a specular feature $K_s \in \mathbb{R}^3$ and employ a small MLP f_s as a fragment shader. This MLP takes the specular feature K_s and viewing direction v as inputs to compute the specular color as follows:

$$\hat{I}_d = \mathcal{R}(M, K_d, P),\ \hat{I}_s = f_s(\mathcal{R}(M, K_s, P), v) \tag{7.4}$$

The final rendered color $\hat{I} \in \mathbb{R}^3$ is obtained by combining the diffuse and specular components, $\hat{I} = \hat{I}_d + \hat{I}_s$. To optimize the diffuse, specular features, and an MLP, we employ a combination of

L1 and SSIM [131] as the photometric loss in image space. This loss is calculated between the final rendered image and the ground truth image from the corresponding viewpoint, as follows:

$$L_{photo} = (1 - \alpha) \parallel \hat{I} - I \parallel + \alpha \, (1 - SSIM(\hat{I}, I)) \tag{7.5}$$

where α is set to 0.2 and the weight to balance the losses.

7.4 Experiments

7.4.1 Experiment Settings

Matterport3D Dataset The Matterport3D dataset [7] consists of 90 indoor scenes encompassing a total of 10,800 panoramic images. Following the EDM approach [101], we define the ground truth camera poses of these images. From the validation and test sets of the official benchmark split, we selected three scenes each, ensuring that these scenes are entirely indoors or composed of levels no higher than the second floor. Since dense matching requires overlapped image pairs as input, we assume that during 360-degree camera scanning, the information about which room each camera belongs to is known. To utilize this information effectively, we leveraged the $house_segmentation$ labels available in the Matterport3D dataset. Images taken within the same room were selected as matching pairs, while for transitions between different rooms, only one image pair with the highest overlap was linked. For EDM [101], the ERP images were resized to 640×320, whereas images for other feature matching methods were resized to 1024×512.

Stanford2D3D Datatset The Stanford2D3D dataset [8] comprises scenes from six large-scale indoor environments collected from three different buildings. We utilize this dataset to demonstrate that

our spherical SfM method outperforms existing SfM approaches in terms of registration performance and accurate pose estimation. Specifically, we selected three scenes containing a moderate number of images. Assuming that overlapping image pairs were predefined during the scanning process, we determined covisibility and defined matching pairs using geometric meshes and ground truth poses. Additionally, we resized the equirectangular projection (ERP) images following the same methodology employed in the Matterport3D dataset.

Implementation Details We construct the Spherical Structure from Motion based on SphereSfM [299] and COLMAP [29]. To address challenges in textureless or occluded regions, we leverage the dense matching algorithm EDM [101] for enhanced feature extraction. Following the DebSDF [290], the SDF and color networks are implemented using an 8-layer MLP and a 2-layer MLP, respectively, each with a hidden dimension of 256. The networks are optimized for geometric reconstruction using the Adam optimizer [137], starting with an initial learning rate of 5×10^{-4} and applying exponential decay at each iteration. Input images are resized to 384×384, and geometric cues are provided by the pretrained Omnidata model [309]. In Texture Optimization, the specular features are initialized to zero values, matching the resolution of the diffuse texture image. The small MLP consists of 2 layers with 32 hidden dimensions. We employ the Adam optimizer [137] with a learning rate of 5×10^{-4} and utilize a learning rate scheduler. The model is trained for 7,000 iterations. We render images at a resolution of 512×512.

7.4.2 Experimental Results

Camera Pose Estimation Accurate geometric 3D reconstruction depends on precise camera pose estimation. We compare our proposed method, which leverages dense matching [101] and Spherical SfM,

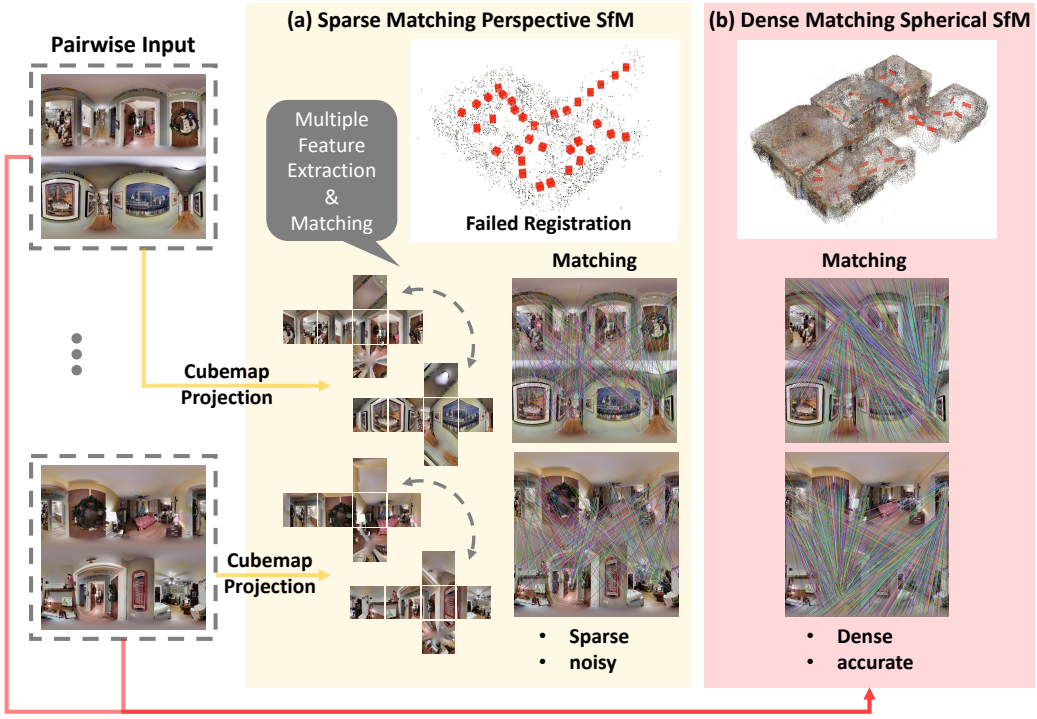


Figure 7.4: We provide a visual comparison between sparse matching perspective SfM and our proposed dense matching spherical SfM. In (a), ERP images are converted into a cubemap representation, after which feature matching is performed across all 36 possible image pairs, resulting in sparse and noisy correspondence matches. (b) demonstrates our approach, which directly finds dense and accurate correspondences on ERP images, thereby facilitating the construction of a detailed 3D structure.

against four different methods: 1) **OpenMVG:** An open-source SfM pipeline that supports spherical camera models [278]. 2) **SPSG COLMAP:** This approach employs SuperPoint [285] and SuperGlue [293] for feature detection and matching on perspective images. ERP images are transformed into a cubemap representation, yielding six perspective views per ERP image. Feature matching is performed in a brute-force manner across all 36 possible image pairs. Subsequently, incremental triangulation and bundle adjustment are performed under the rig constraints for each 360 camera node. 3) **DKM COLMAP:** This method leverages DKM [283] to establish dense correspondences without relying on explicit feature extraction. Since DKM is designed for perspective images, it also follows the cubemap projection and exhaustive pairwise matching, and rig-based optimization. 4) **SphereGlue**

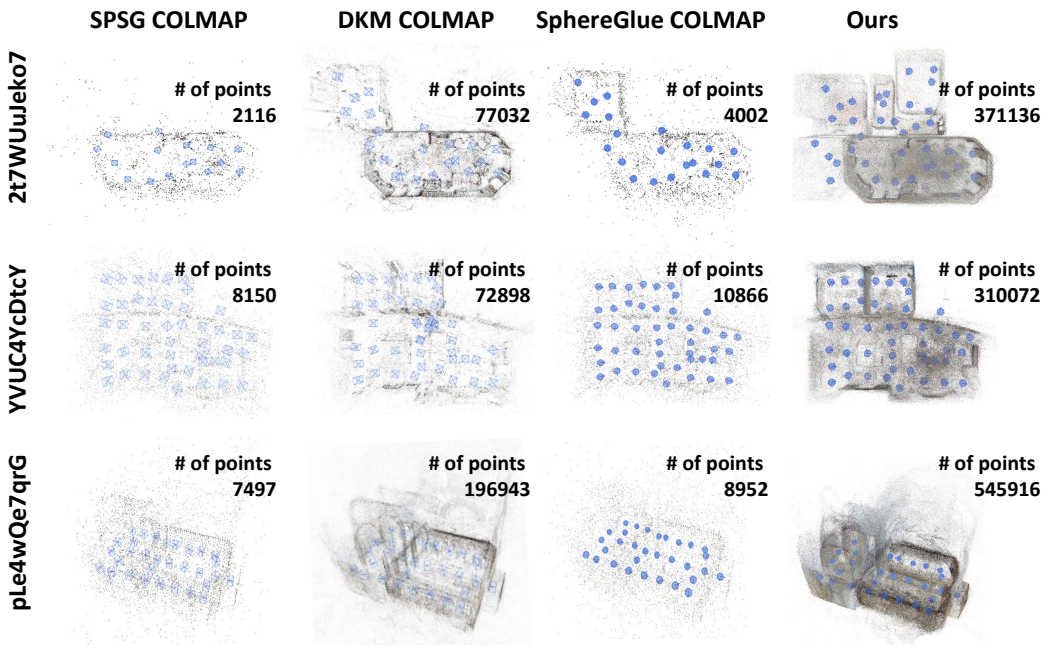


Figure 7.5: **Qualitative Comparison of SfM results on Matterport3D.** By leveraging dense features on equirectangular projection (ERP) images, our method effectively finds correspondences in textureless regions.

COLMAP: Building on COLMAP, SphereSfM [299] incorporates spherical camera models and spherical bundle adjustment to handle ERP images. For feature extraction and matching, SuperPoint [285] with a local planar approximation [310] and SphereGlue [304] are utilized to mitigate distortion in ERP images. To estimate initial poses for incremental triangulation, we follow the normalization strategy and non-linear optimization proposed by Solarte et al. [300].

Table 7.1 presents the quantitative results of camera pose estimation on the Matterport3D dataset. While previous methods also performed well in smaller scenes with significant image overlap, our proposed method demonstrated superior robustness across diverse scenes. By leveraging a wide field of view (FoV) for matching, our approach proved particularly effective on datasets with sparse input views. Notably, it achieved the most accurate registrations in scenes with frequent occlusions. Figure 7.5 visualizes the estimated camera poses and triangulated 3D points, demonstrating that our method estimated both the largest number of 3D points and the most camera poses.

Table 7.1: **Quantitative Evaluation of Camera Localization Performance** on the Matterport3D dataset [7]. Our method demonstrates superior results in terms of registration accuracy and achieves comparable performance in pose estimation across most scenes.

Method	2t7WUuJeko7				8194nk5LbLH				pLe4wQe7qrG			
	# Registered	AUC @3°	AUC @5°	AUC @10°	# Registered	AUC @3°	AUC @5°	AUC @10°	# Registered	AUC @3°	AUC @5°	AUC @10°
OpenMVG	5 / 37	0.82	1.10	1.30	2 / 20	0.27	0.37	0.45	25 / 31	45.11	52.82	58.67
SPSG COLMAP	16 / 37	14.30	15.79	16.90	12 / 20	26.64	29.88	32.31	**31 / 31**	75.02	85.01	92.51
DKM COLMAP	22 / 37	25.14	27.70	29.62	9 / 20	14.64	16.36	17.66	**31 / 31**	**75.94**	**85.56**	**92.78**
SphereGlue COLMAP	21 / 37	23.95	26.98	29.26	12 / 20	23.11	27.76	31.24	**31 / 31**	66.83	80.06	90.03
IM360 (Ours)	**37 / 37**	**49.16**	**69.05**	**84.53**	**20 / 20**	**34.91**	**44.16**	**66.87**	**31 / 31**	73.95	84.37	92.18
Method	RPmz2sHmrrY				YVUC4YcDtcY				zsNo4HB9uLZ			
	# Registered	AUC @3°	AUC @5°	AUC @10°	# Registered	AUC @3°	AUC @5°	AUC @10°	# Registered	AUC @3°	AUC @5°	AUC @10°
OpenMVG	22 / 59	9.86	11.31	12.41	8 / 46	2.22	2.41	2.56	2 / 53	–	–	–
SPSG COLMAP	34 / 59	22.93	26.87	29.83	**46 / 46**	**84.21**	**90.53**	**95.26**	19 / 53	7.84	9.22	10.55
DKM COLMAP	51 / 59	41.56	53.56	62.58	**46 / 46**	56.33	61.00	64.67	19 / 53	7.87	8.71	9.41
SphereGlue COLMAP	34 / 59	22.23	26.45	29.62	**46 / 46**	78.32	86.97	93.48	22 / 53	6.25	7.56	9.46
IM360 (Ours)	**59 / 59**	**53.44**	**71.87**	**85.93**	**46 / 46**	72.40	83.40	91.71	**53 / 53**	**52.35**	**71.14**	**85.49**

Table 7.2: **Quantitative evaluation of camera Localization Performance** on the Stanford2D3D dataset [8]. Our method outperforms all other approaches in terms of registration accuracy and pose estimation.

Method	area 3				area 4				area 5a			
	# Registered	AUC @3°	AUC @5°	AUC @10°	# Registered	AUC @3°	AUC @5°	AUC @10°	# Registered	AUC @3°	AUC @5°	AUC @10°
OpenMVG	6 / 85	0.35	0.38	0.40	17 / 258	0.34	0.37	0.39	8 / 143	0.23	0.25	0.26
SPSG COLMAP	28 / 85	6.71	7.47	8.29	73 / 258	4.75	5.77	6.55	54 / 143	8.84	10.93	12.51
DKM COLMAP	45 / 85	13.10	14.92	16.32	79 / 258	3.39	3.96	4.40	78 / 143	15.27	18.71	21.38
SphereGlue COLMAP	30 / 85	5.87	7.38	8.75	116 / 258	8.62	11.64	14.13	69 / 143	5.10	8.06	11.73
IM360 (Ours)	**85 / 85**	**32.20**	**45.12**	**58.23**	**258 / 258**	**28.70**	**50.50**	**71.29**	**138 / 143**	**19.52**	**32.54**	**51.72**

Surface Reconstruction Recently, neural surface reconstruction methods based on volumetric rendering techniques [39, 223, 290] have shown strong performance in dense 3D reconstruction. In particular, several approaches that leverage monocular geometry priors [223, 290] demonstrate robustness in handling large textureless regions and sparse input views. Therefore, we adopt DebSDF [290] to estimate the signed distance function and apply the marching cubes algorithm [82] to convert the neural implicit representation into triangle meshes. Figure 7.6 presents shading meshes extracted without texture, the most effective way to visualize geometric differences. This highlights the effectiveness of our SfM method in producing decent geometric meshes from omnidirectional data.

Texture Map Optimization We now evaluate the results of our view synthesis method by comparing it against four different approaches: 1) **ZipNeRF** [49], the state-of-the-art neural radiance field method;

Table 7.3: **Quantitative Comparison of Rendering Performance** on the Matterport3D Dataset. Our method shows higher rendering quality of our method, IM360,across all scenes, achieving a 3.5 PSNR over the TexRecon [9]. The best-performing algorithms for each metric are shown in boldface.

Method	Rendering	PSNR ↑ / SSIM ↑ / LPIPS ↓						
		2t7WUuJeko7	8194nk5LbLH	pLe4wQe7qrG	RPmz2sHmrrY	YVUC4YcDtcY	zsNo4HB9uLZ	Mean
ZipNeRF [49]	Volume	15.1 / 0.51 / 0.69	11.9 / 0.44 / 0.74	14.1 / 0.43 / 0.70	13.1 / 0.50 / 0.69	15.3 / 0.53 / 0.68	14.1 / 0.65 / 0.68	13.9 / 0.51 / 0.68
3DGS [10]	Splat	14.4 / 0.47 / 0.53	11.9 / 0.38 / 0.60	13.5 / 0.37 / 0.56	12.9 / 0.51 / 0.55	14.2 / 0.49 / 0.52	13.7 / 0.60 / 0.50	13.4 / 0.47 / 0.55
SparseGS [48]	Splat	15.4 / 0.48 / 0.50	12.8 / 0.36 / 0.58	13.8 / 0.37 / 0.52	13.6 / 0.46 / 0.57	15.5 / 0.50 / 0.49	14.4 / 0.58 / 0.55	14.3 / 0.46 / 0.53
TexRecon [9]	Mesh	17.1 / 0.51 / 0.43	14.2 / 0.47 / 0.49	15.9 / 0.45 / 0.46	16.7 / 0.60 / 0.38	15.0 / 0.59 / 0.42	17.0 / 0.51 / 0.42	15.9 / 0.54 / 0.43
IM360* (Ours)	Mesh	19.6 / 0.60 / 0.37	16.7 / 0.60 / 0.43	18.5 / 0.65 / 0.36	18.9 / 0.65 / 0.38	18.3 / 0.60 / 0.42	19.8 / 0.68 / 0.41	18.6 / 0.63 / 0.39
IM360 (Ours)	Mesh	**19.8 / 0.61 / 0.38**	**16.9 / 0.61 / 0.43**	**18.8 / 0.66 / 0.35**	**20.6 / 0.75 / 0.32**	**19.4 / 0.66 / 0.32**	**21.0 / 0.74 / 0.37**	**19.4 / 0.67 / 0.37**

2) **3DGS** [10], which employs 3D Gaussian splats rendered through rasterization; 3) **SparseGS** [48], designed to address sparse-view challenges in 3D Gaussian splatting; and 4) **TexRecon** [9], a classical texture mapping method for large-scale scenes. Table 7.3 reports the rendering quality with the PSNR, SSIM [131], and LPIPS [311] on Matterport3D dataset. We consistently observe that neural rendering methods, including NeRF [49] and 3DGS [10], perform inferior to traditional texture reconstruction techniques [9]. The significant discrepancy between training PSNR and novel view PSNR indicates that this gap arises due to the sparsity of the scanned images. To address this, we evaluated the state-of-the-art SparseGS [48] approach, specifically designed for sparse-view scenarios. While SparseGS [48] achieves a 1 PSNR improvement in rendering quality, this gain indicates that sparse scanning remains a significant challenge. Our method achieves improved rendering quality across all scenes, demonstrating a 3.5 PSNR increase over the TexRecon [9] method.

In Fig. 7.8, we visually compare our method to several recent rendering approaches: TexRecon [9], SparseGS [48], and ZipNeRF [49]. We visualize only SparseGS instead of 3DGS, as both methods employ similar rendering techniques, but SparseGS demonstrates superior rendering quality in sparsely scanned indoor environments. TexRecon shows the visual seams between face texture and color misalignment. ZipNeRF tends to generate ambiguous pixels, resulting in severe artifacts. SparseGS contains obvious floating artifacts in ground and floor regions. In contrast, our method pro-

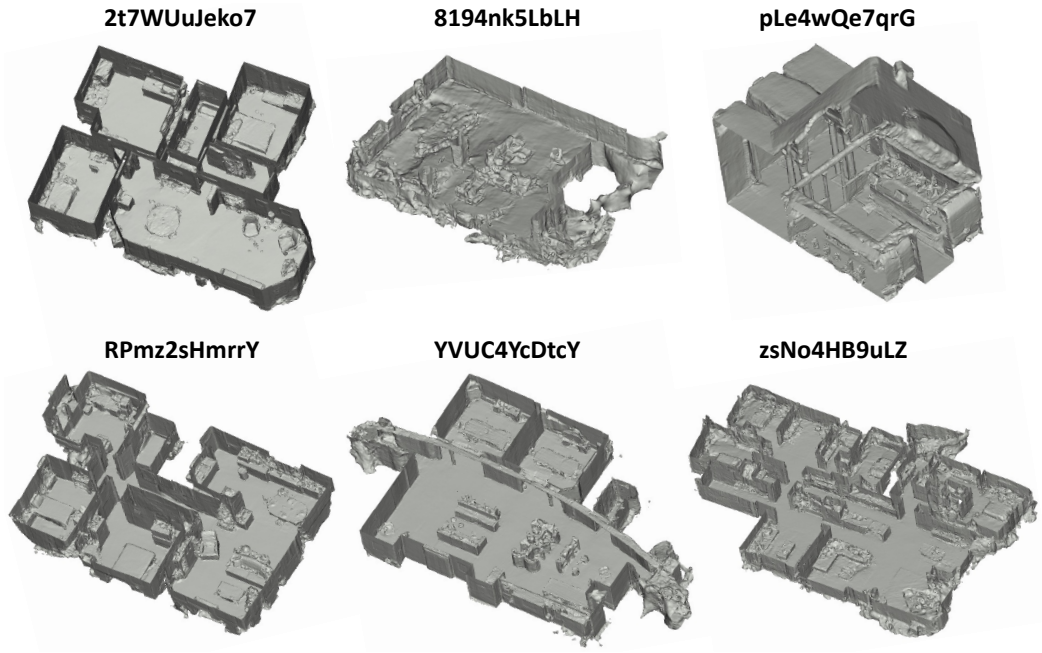


Figure 7.6: We visualize the shading meshes extracted without texture on Matterport3D. Leveraging the estimated poses from IM360, we achieve accurate surface reconstruction, demonstrating the effectiveness of our SfM method in generating decent geometric meshes from omnidirectional data.

duces much higher quality textures and consistently demonstrate improvement across various scenes.

7.4.3 Additional Experiments and Ablation Study

We observed that when initial poses are determined using two-view geometry without applying the method proposed by Solarte et al. [300], registration performance significantly deteriorates, resulting in fragmented SfM results. Furthermore, as shown in Table 7.1 and 7.2, by modifying the feature matching in COLMAP-based methods and comparing the registration and pose accuracy, we confirmed that our approach demonstrates superior robustness. We also evaluate our SfM method on multi-floor scenes, as illustrated in Fig. 7.7. Our spherical SfM exhibits robust performance in multi-floor scenes, highlighting the advantages of leveraging spherical cameras within our proposed pipeline for large-scale indoor environments.

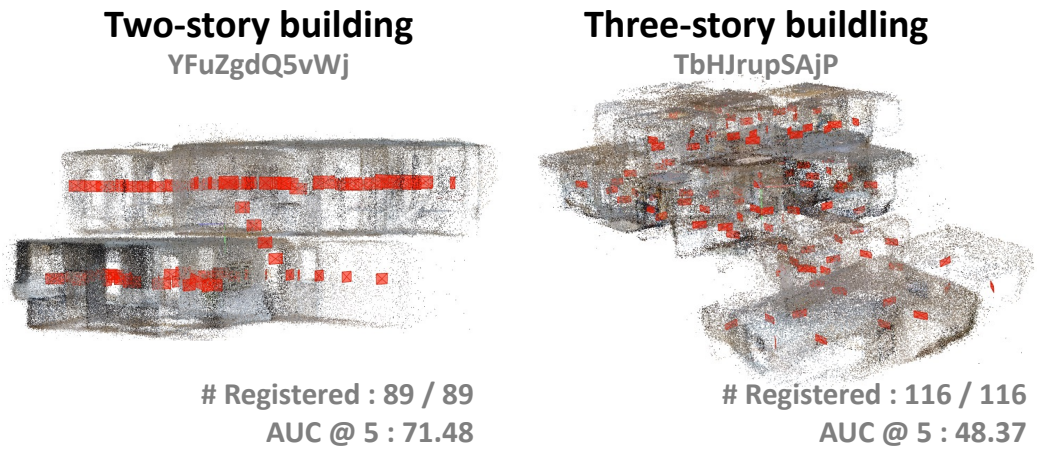


Figure 7.7: Localization results in multi-floor scenes of Matterport3D. Multi-floor scenes pose significant challenges for visual localization due to narrow corridors and staircases connecting different floors. Despite these difficulties, IM360 demonstrates the ability to estimate poses, ensuring robust localization in large-scale indoor environments.

In table 7.3, IM360* only finetunes diffuse texture without utilizing specular features. Without texture optimization stage outlined in Sec 7.3.3, our rendering quality is equivalent to that of TexRecon [9]. Although we apply texture optimization soley to the diffuse texture map, our method achieves a substantial improvement in rendering quality, with a 2.7 PSNR increase. Additionally, our final approach, which combines both diffuse and specular colors, provides a further 0.8 PSNR gain by accounting for view-dependent components.

7.5 Conclusion

We introduce a novel framework for textured mesh reconstruction for omnidirectional cameras in sparsely scanned large-scale indoor environments. To tackle the challenges posed by textureless regions and sparse viewpoints, we propose a spherical Structure-from-Motion (SfM) that incorporates dense feature matching and integrates spherical camera models. Additionally, we reconstruct texture maps and leverage differentiable rendering to enhance its quality, improving the rendering performance in sparse-view scenarios. One limitation of our approach is its reliance on human annotation



Figure 7.8: **Qualitative Comparisons with Existing Methods.** We visually compare our proposed approach with TexRecon [9], SparseGS [48], and ZipNeRF [49]. Our method can render high frequency details and results in lower noise.

for determining matching pairs. While 360° image scanning scenarios typically involve a small number of captures, making manual annotation feasible, integrating an automated image retrieval system would be a more scalable solution. For geometric reconstruction, we employ cubemap projection to transform 360° images into perspective views and utilize state-of-the-art volumetric rendering methods. However, this approach has limited capacity for reconstruction, as the quality deteriorates when the scene exceeds a certain scale. In future work, we aim to further refine geometric reconstruction and enhance the overall pipeline for large-scale 360° mapping.

III. Generation

Chapter 8: UAVTwin: Neural Digital Twins for UAVs using Gaussian Splatting

8.1 Introduction

Unmanned aerial vehicles (UAVs) have become indispensable for human recognition tasks, including person detection [53, 312] and action recognition [13, 313–315] in critical applications such as surveillance, disaster response, and security monitoring. Unlike autonomous driving and mobile robots—where extensive, large-scale datasets have propelled rapid progress—UAV-based human identification remains hindered by the scarcity of high-quality datasets. The challenges of dataset curation in UAV-based real-world scenarios are twofold: (1) UAVs are constrained by limited flight time, sensor diversity, and variability in camera viewpoints, making large-scale real-world data collection labor-intensive and expensive; and (2) human annotation at UAV altitudes is difficult and error-prone, particularly for fine-grained tasks such as pose estimation.

Given these constraints, synthetic data has emerged as a promising alternative for UAV-based perception [316–320] and human activity analysis [321, 322]. Recent synthetic dataset pipelines [321, 322] leverage virtual environments (e.g., Unreal Marketplace) and rendering engines [83, 323–325] to generate scene backgrounds, synthetic humans, and UAV camera trajectories. However, despite advances in realistic rendering, synthetic data suffers from a significant synthetic-to-real domain gap [312, 314, 326], largely due to lighting discrepancies, limited texture fidelity, and unrealistic human appearances.

Neural rendering [10, 85] has recently emerged as a promising technique to bridge this domain gap, offering photorealistic rendering and digital twin reconstruction of real-world environments. By leveraging 3D reconstruction techniques compatible with rendering engines [83, 323, 325], these methods present potential solution for building high-fidelity *digital twin* from real-world data with pixel-

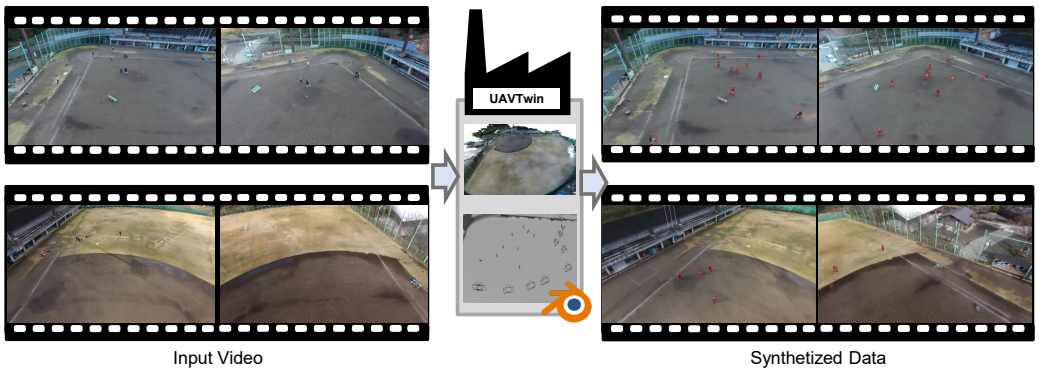


Figure 8.1: UAVTwin processes video captured by the UAV as input, enabling data generation for training UAV-based human recognition methods.

accurate annotations. However, *applying neural rendering for UAV-based datasets presents fundamental challenges*:

- Appearance Variation Across UAV Captures: UAV-based imagery often exhibits *significant inter-sequence appearance differences*, primarily due to *varying illumination, environmental conditions, and multi-UAV data collection* at different times. It is important to use videos captured across different sensors or times to achieve dense coverage of large areas. While previous methods [327–330] introduce per-image appearance embeddings, these embeddings fail to generalize across unseen UAV sequences and cannot directly transfer to new UAV trajectories. These methods inherently conflicts with our goal of freely generating data for any novel UAV trajectories.

- Dynamic Objects and Transient Artifacts: UAV-based scenes frequently contain *moving objects (e.g., vehicles, pedestrians) that occlude the background*, leading to noisy training data. While segmentation models like Segment Anything Model (SAM) [331] and its variants [50] have made progress in 2D scene understanding, they struggle with UAV footage where *dynamic objects are small and sparsely distributed*, occupying fewer pixels than in standard scenes. Additionally, prompt-based segmentation methods [50, 331] often fail to accurately capture all dy-

namic objects.

Main Results: We introduce *UAVTwin*, a new approach that can build a digital twin from real-world data captured by UAV and generate novel data for training UAV-based human recognition algorithms. First, we propose Multi-Sequence Gaussian Splatting (MsGS) in *UAVTwin* that incorporates per-sequence appearance embedding for training neural radiance fields for addressing large appearance variation. Additionally, it employs mask refinement to remove transient and dynamic objects. Our model can generate photo-realistic rendering from any novel-view camera trajectory without prior test images and then reconstruct a high-fidelity mesh for neural data generation. For seamless integration with Blender [325], a 3D mesh that is fully compatible with its framework is required.

Next, we explore a new approach to achieve data augmentation in UAV scenes by integrating MsGS and data generation using a graphics engine [325]. Our system can import synthetic humans from digital assets and environmental lighting, and generate camera poses that mimic UAV flight trajectory for realistic data generation. We can select various motions for synthetic humans and generate various scenarios for data augmentation. We further incorporate scene composition for enhancing realism of generated data. Thanks to scene backgrounds rendered by MsGS and composition technique, our system can minimize the domain gap between virtual environment and the real world.

- We propose a MsGS for training 3DGS-based neural radiance field, which utilizes per-seqeunce embedding to enable varying appearance modeling from multi-sequence image collections and refine semantic masks for addressing dynamic objects.

- We present a neural data generation approach that integrates the strengths of neural rendering and graphics engines to augment training data for UAV-based perception algorithms.

- We validate the background rendering quality, demonstrating that our method outperforms other Gaussian Splatting algorithms by 2.79 dB PSNR on the Okutama-Action dataset. Additionally, we assess the effectiveness of augmented data by training a person detection model, achieving a 2.5%–13.7% improvement in mAP.

8.2 Related Work

Neural Rendering. Neural rendering includes methods that utilize neural networks to replace or improve traditional rendering pipelines. Neural Radiance Fields (NeRF) [85] have demonstrated remarkable photorealistic rendering quality [88, 196] and enabled high-fidelity surface reconstruction [2,3,21,39]. More recently, 3D Gaussian splatting (3DGS) [10,332] has achieved a comparable rendering quality while offering significantly superior rendering speed compared to NeRF. However, 3DGS often lacks precise alignment with the underlying scene geometry, negatively affecting rendering quality and surface extraction. Some studies [22, 289] leverage off-the-shelf reconstruction methods and align Gaussian splats with this geometry. Several approaches [11, 12, 52] have explored the use of 3D Gaussian splats not only for rendering but also for surface extraction. In particular, 2DGS [11] demonstrates remarkable capability by leveraging surefels as a representation. PGSR [12] also exhibits clean geometric fidelity while high rendering quality. In this work, we present a method to train 3D Gaussian splats with UAV captured images in order to achieve both photorealistic rendering and surface reconstruction.

Gaussian Splatting for Simulation and Robotics. Recent innovations in neural rendering methods enable building digital twin from the real-world environment. Many robotics applications [62, 63] leverage this photorealistic simulator to learn the real-world and generate the data for specific ap-

plication. For UAV applications, previous studies [64, 65] have developed GS-based simulators using data collected in controlled laboratory environments, enabling end-to-end training for UAV navigation. However, to the best of our knowledge, our work is the first GS-based digital twin generation method designed for training UAV perception algorithms, with a particular focus on human-related tasks such as person detection.

8.3 Building Digital Twin

The *UAVTwin* system takes UAV-captured images as input and constructs a digital twin to generate data for training UAV perception algorithms. An overview of our pipeline is illustrated in Fig 8.2. The UAV collects N video sequences $\{\overline{V}_j\}_{j=1}^{N}$ from the real-world environments. These sequences are captured at different times, from varying UAV trajectories and altitudes, and under diverse lighting conditions. Each video sequence, denoted as $\overline{V_j} = \{I_i\}_{i=1}^{K_j}$, consists of K_j consecutive images, each exhibiting various characteristics. For all $\mathcal{T}$ images, we extract camera poses and initial point clouds using COLMAP [29]. We propose Multi-sequence Gaussian Splatting (MsGS) to reconstruct accurate 3D geometry for the background mesh and enable photorealistic rendering for the background region, which are basic components of rendering engine. To handle transient objects, two types of segmentation maps are extracted from all images using both the SAM [50, 331] and entity segmentation methods [333]. Both segmentation maps are used for training the 3D Gaussian model and refining mask quality.

8.3.1 Preliminaries: 3D Gaussian Splatting

3D Gaussian Splatting (3DGS) [10] represents a 3D scene using a set of anisotropic 3D Gaussians G. Each Gaussian incorporates multiple attributes to represent both the geometry and appearance of the scene. Each 3D Gaussian is parameterized using its mean position $\mu_i \in \mathbb{R}^3$, and 3D covariance matrix $\Sigma_i \in \mathbb{R}^{3x3}$ in world space. During optimization, the covariance matrix is reparametrized using a scaling matrix $S_i \in \mathbb{R}^3$ and a rotation matrix $R_i \in \mathbb{R}^{3x3}$, which is represented by a quaternion $q \in \mathbb{R}^4$, following the formulation $\Sigma_i = R_i S_i S_i{}^T R_i{}^T$. P_{CW} is the world-to-camera transformation matrix and W is the rotational component of P_{CW}. For rendering, the 3D Gaussians are transformed into the camera coordinates with W and projected to the image space using the affine transformation matrix J_i following [274]. The 2D Gaussians $g_i(\mu_i', \Sigma_i')$ on the image plane is represented as $\mu_i' = \pi(P_{CW} \cdot \mu_i)$ and $\Sigma_i' = J_i W \Sigma_i W^T J_i{}^T$. π is the projection operation and Σ' is the 2D covariance matrix in the image space. 3DGS sorts all the Gaussian involved in a pixel and renders a 2D image via α-blending using the following equation:

$$\bar{C} = \sum_{i=1}^{n} c_i \alpha_i \prod_{j=1}^{i-1}(1-\alpha_j), \quad \alpha_i = o_i g_i(x). \tag{8.1}$$

where c_i and α_i represent the color and opacity of this point multiplied by an optimizable per-point opacity and SH color coefficients. These 3D Gaussians are optimized with respect to the photometric loss L_{pho} using a combination of SSIM [163] and L_1 losses between the rendered color $\bar{C}$ and groundtruth colors C:

$$L_{pho} = (1-\lambda_{pho})\text{L1}(\bar{C}, C) + \lambda_{pho}\text{SSIM}(\bar{C}, C)). \tag{8.2}$$

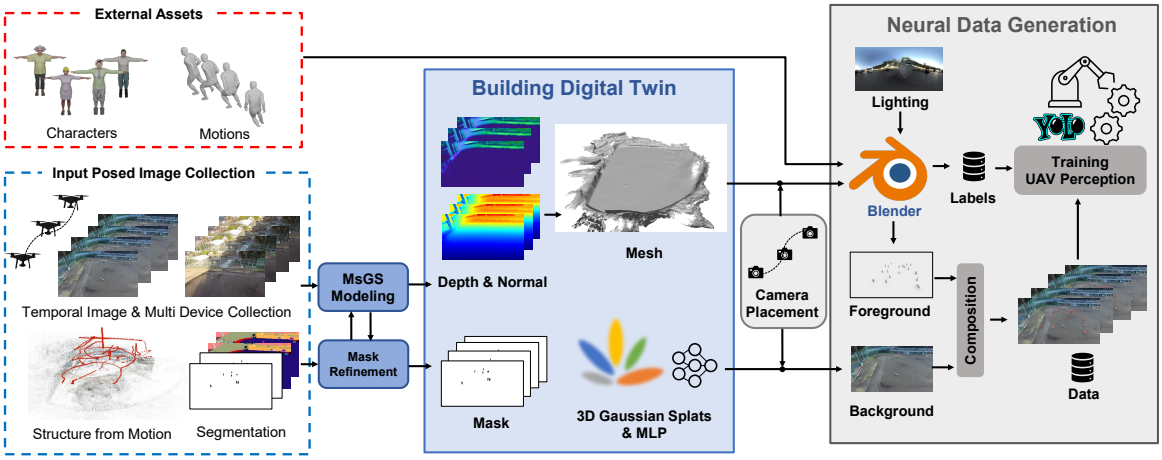


Figure 8.2: **UAVTwin framework.** In Section 8.3, our approach first constructs a digital twin using UAV-based images captured at different times. We introduce MsGS, a novel 3DGS method to analyze varying appearance images and reconstruct a clean mesh, Gaussian splats, and an MLP for novel-view synthesis. Then, in Section 8.4, our method generates data by compositing foreground humans rendered in Blender with backgrounds rendered using trained Gaussian splats.

8.3.2 Multi-sequence Gaussian Splatting (MsGS)

Previous research commonly employs either per-image embeddings [327, 329, 334] or Convolutional Neural Network (CNN)-based appearance encoders [328, 330] to address appearance variations. However, these approaches are not well-suited for our data generation pipeline, as it is difficult to accurately capture the appearance of unseen images produced by a random UAV trajectory. The proposed MsGS uses per-sequence embedding $\{q_i\}_{i=1}^{N}$ for each sequence. The original 3DGS [10] utilizes spherical harmonics (SH), which may not work well in terms of encoding large appearance variations due to their explicit color representations. Instead, we use 3-layer color MLP f of width 128. As shown in Fig. 8.3, we define a color MLP f that takes the viewing direction embedding d, per-sequence embedding q_i, base color $\hat{c}_i$, and a learnable embedding for each Gaussian h_i, as input. Following the previous works [12, 329, 334], the color MLP is designed to output the parameters of an affine transformation,

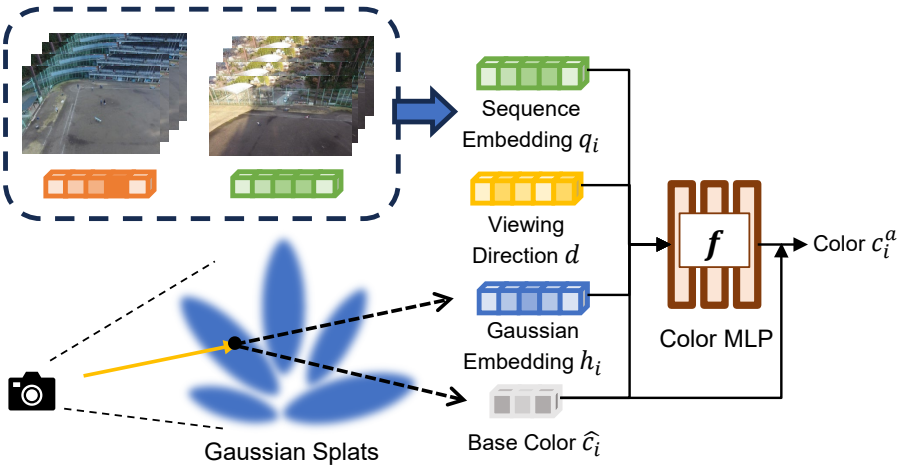


Figure 8.3: **The core components of MsGS.** From a video sequence captured at different times with varying appearances, we define a sequence embedding q_i. For novel-view image rendering, we select a specific sequence. Our color MLP f takes as input the sequence embedding, viewing direction embedding, per-Gaussian embedding, and base color. The output color is modulated to account for appearance variations.

α and β.

$$(\alpha_i, \beta_i) = f(h_i, q_i, d, \hat{c}_i), \quad c_i^a = \alpha_i \hat{c}_i + \beta_i, \tag{8.3}$$

where c_i^a represents the toned color corresponding to the assigned sequence. Then, the toned color for each Gaussian splat is passed through the 3DGS rasterization process to render the final image map C^a for each camera view. The benefit of per-sequence embedding is that we can freely generate data for novel-view poses without utilizing prior test-view images like previous research [329, 330]. Furthermore, we enforce consistent appearance within the same sequence and utilizes per-sequence embeddings to capture variations in appearance across different sequences. Additionally, incorporating a per-sequence embedding is effective in mitigating floaters, improving not only rendering quality but also surface reconstruction.

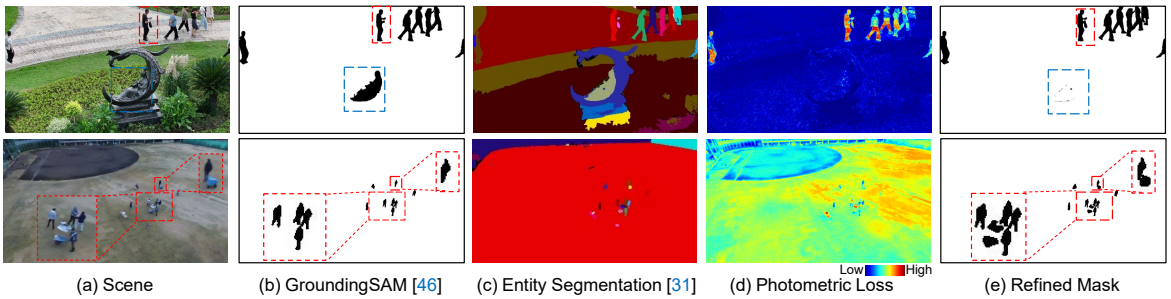


Figure 8.4: **Mask Refinement.** (a) is an example of training images with dynamic objects. (b) is its segmentation masks $\overline{M}$ using GroundingSAM [50]. (c) is the entity segmentation masks $\hat{M}$ [51]. Based on the SAM masks in (b), we add entity masks with high photometric loss (red dotted boxes) and remove those with low photometric loss (blue dotted boxes), resulting in the refined masks shown in (e).

8.3.3 Mask Refinement

To mitigate the influence of transient objects, our approach requires accurate masks for moving objects. Previous uncertainty models based on DINO [329] are ineffective due to the extremely small size of these moving objects. Instead, given $\mathcal{T}$ images, we use GroundingSAM [50] to extract image masks. To detect bounding boxes associated with specific text phrases, we provide a text caption (e.g., "human") as input for text embedding in Grounding DINO [335]. Using these box prompts, SAM [331] generates mask annotations $\overline{M} = \{M_i\}_{i=1}^{\mathcal{T}}$. However, GroundingSAM often produces inaccurate masks.

To address this, we introduce a novel mask refinement module to refine the accuracy of the SAM masks $\overline{M}$ in Fig.8.4-(a). Inspired by Entity-NeRF [51], we refine masks leveraging the entity masks $\{\hat{M}_i\}_{i=1}^{\mathcal{T}}$ in Fig.8.4-(b) extracted through entity segmentation [333]. We first measure error maps $\{R_i\}_{i=1}^{\mathcal{T}}$ using photometric loss L_{pho} in Fig.8.4-(c) in Eq. 8.2 for each training views. For a set of pixels $S(e_{id})$ corresponding to an entity ID E_{id}, we compute error map $R(e_{id})$ per each entity as

follows,

$$R(e_{id}) = \frac{\sum_{i \in S(e_{id})} L_{pho}(i)}{A(S(e_{id}))} \tag{8.4}$$

where $A(\cdot)$ represents the area of the corresponding pixels.

Here, we make three assumptions: (1) in UAV capture scenarios, most dynamic objects occupy a small number of pixels, (2) transient objects exhibit significant RGB loss errors, and (3) if a cluster of pixels in SAM masks has an RGB loss error below a certain threshold, it can be considered a static mask. Thus, we remove entity masks and SAM masks that exceed the area thresholds ρ_1 and ρ_2 respectively and select entity masks $S'(e_{id})$ where $A(S(e_{id})) < \rho_1$ and $A(\overline{M}) < \rho_2$. Next, we identify all entity masks that overlap with the dilated SAM masks $\overline{M}$ and gather these overlapped masks $M' \leftarrow Dilate(\overline{M}) \cup \hat{M}$. To determine whether an entity mask should be added to or removed from the original SAM masks $\overline{M}$, we apply an RGB loss threshold ρ_{rgb}. For each entity mask $\hat{M}(e_{id}) \in M'$, we include $\hat{M}(e_{id})$ to $\overline{M}$ if $R(e_{id})$ exceeds ρ_{pho}; otherwise, we remove $\hat{M}(e_{id})$ from $\overline{M}$. We outline the pseudo code for our mask refinement in the supplementary material.

8.3.4 Geometry Reconstruction

Following PGSR [12], we first render the normal maps N and distance map $\hat{D}$ of the plane and then convert them into depth maps D using camera intrinsic matrix K.

$$D(p) = \frac{\hat{D}}{N(p)K^{-1}\widetilde{p}}, \tag{8.5}$$

where $p \in \mathbb{R}^2$ is the pixel coordinate and $\hat{p}$ is the homogeneous coordinate of p. Furthermore, we apply the scale regularization to flatten the 3D Gaussian splats. Given the scale parameters $S_i = (s_1, s_2, s_3)$,

we minimize the minimum scale as follows:

$$L_{scale} = \sum_{i \in G} (\lambda_s |\min(s_1, s_2, s_3)|). \tag{8.6}$$

8.3.5 Training Objectives for MsGS

Our training process consists of two stages. In the first stage, we train 3D Gaussian attributes and MLP f using the original SAM masks $\overline{M}$, applying both scale regularization in Eq. 8.6 and masked photometric loss L_{Mpho}. The masked photometric loss is defined as follows:

$$L_{Mpho} = (1 - \lambda_{pho})\overline{M}\text{L1}(C^a, C) + \lambda_{pho}\overline{M}\text{SSIM}(C^a, C)), \tag{8.7}$$

where the mask $\overline{M}$ is a binary mask and is multiplied by the per-pixel loss from Eq. 8.2. C^a is the rendered color of the rasterized images.

In the second stage, we apply our mask refinement technique from Sec. 8.3.3 to enhance the original SAM masks $\overline{M}$, resulting in a high-quality mask $\widetilde{M}$. Additionally, we apply masked single-view normal loss L_{Msvgeo} and multi-view regularization L_{mvreg} from previous studies [12, 52, 336]. The details of these two losses are provided in the supplementary materials. The overall loss function L is defined as:

$$L = L_{Mpho} + L_{scale} + L_{Msvgeo} + L_{mvreg}. \tag{8.8}$$

8.3.6 Mesh Reconstruction

We first render the depth for each training view and then apply the TSDF Fusion algorithm [81] to construct the corresponding TSDF field. Subsequently, we extract the mesh B for the background

region. It is essential that Gaussian splats are precisely aligned with the actual 3D geometry represented by mesh B to ensure that the background image rendered by the Gaussian splats can be seamlessly composited with the foreground humans, which are positioned relative to the mesh.

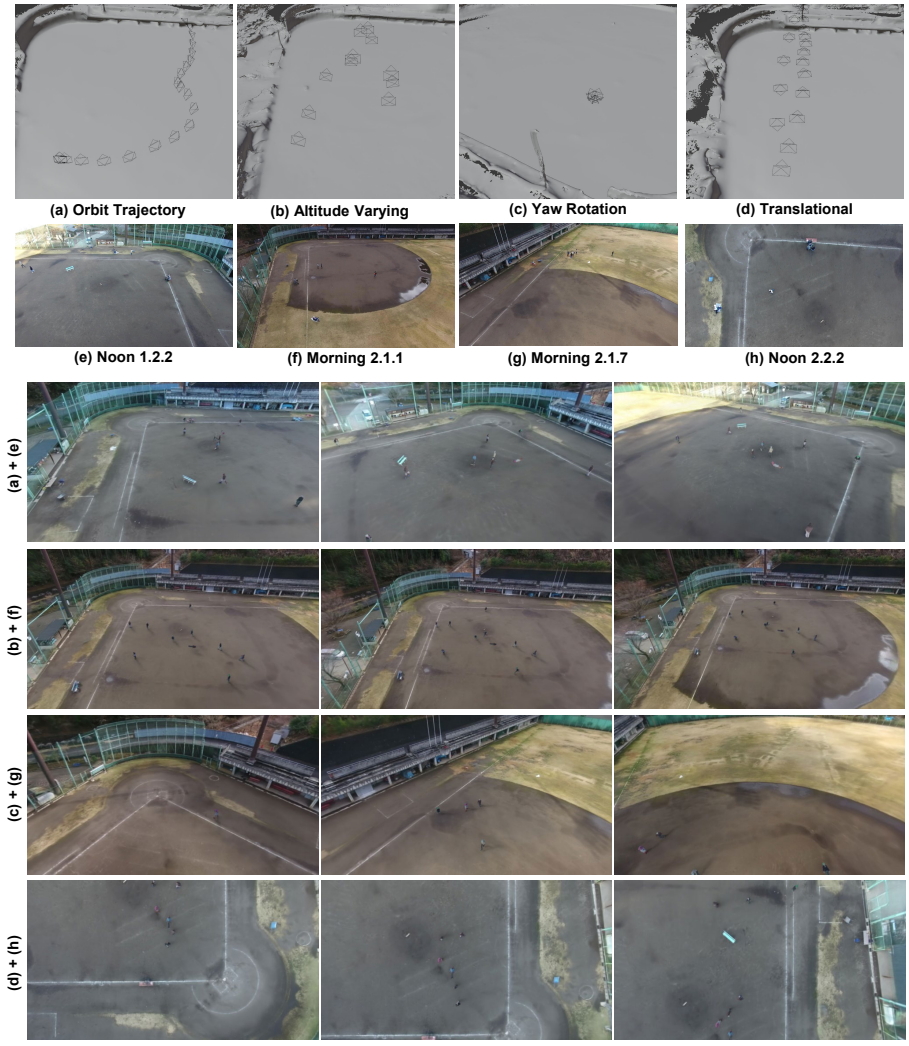


Figure 8.5: **Qualitative Results of Synthetized Data.** The first row illustrates the camera trajectory, while the second row presents a sample training image from the sequence used as a per-sequence embedding input for rendering our MsGS. Data is generated using different trajectories with sequence embeddings extracted from training data: (a)+(e) orbit (Noon 1.2.2), (b)+(f) altitude varying (Morning 2.1.1), (c)+(g) yaw rotation (Morning 2.1.7), and (d)+(h) translational (Noon 2.2.2).

8.4 Neural Data Generation

For data generation, our approach comprises synthetic human placement, camera trajectory generation, and scene composition. Our camera trajectory generation and synthetic human placement are implemented based on the reconstructed mesh B from Section 8.3.6. This ensures seamless composition of the foreground humans, rendered using Blender [325], and the background, rendered with our neural rendering method in Section 8.3.2, providing data for UAV perception.

8.4.1 Camera Trajectory Generation

The camera trajectory is a critical factor in generating UAV data. We consider the camera's pitch angle and various camera movement patterns for data collection. Following Blender convention, we define the camera's location t and rotation r to generate the camera trajectory. To simulate a real drone's movement, we introduce Gaussian random noise $\epsilon^r, \epsilon^t \in \mathbb{R}^3$ to each components, where the camera' location is represented as $(t_x + \epsilon_x^t, t_y + \epsilon_y^t, t_z + \epsilon_z^t)$ and its rotation as $(r_x + \epsilon_x^r, r_y + \epsilon_y^r, r_z + \epsilon_z^r)$.

Translational Trajectory: The UAV moves horizontally or vertically by increasing or decreasing t_x, t_y in a straight line while maintaining a fixed camera orientation, independent of the subject.

Stationary Yaw Rotation: The UAV remains in place but rotates horizontally $r_z \in [0, 2\pi]$ to capture a panoramic view of the environment.

Orbit Trajectory: The UAV moves in a circular path around the subject, maintaining a certain distance while keeping the camera focused on it. We compute the center points p^c of averaging all actors and define azimuthal angle ϕ and radius r_a. This camera is parameterized by $(r_a cos(\phi) + p_x^c, r_a sin(\phi) + p_y^c, t_z + p_z^c)$ using a human-centric spherical system.

Altitude-Varying Trajectory: The UAV dynamically adjusts its altitude by increasing or decreasing

t_z, ascending to expand the field of view for broader coverage and descending to capture finer details with focused observation.

8.4.2 Synthetic Human Placement

After generating the camera trajectory in the previous stage, we utilize SynBody [321], incorporating SMPL-XL models [337] to simulate human actors with diverse body shapes and clothing. Using the reconstructed mesh from Section 8.3.4, we identify the mesh faces visible from all camera views along the given trajectory. This allows us to determine the feasible range for synthetic human placement. Within this range, we randomly partition the area for actor placement and position N_a actors. For each human actor, we randomly select a single-person motion from AMASS [338], excluding interactive or non-ground motions.

Table 8.1: **Quantitative Comparison on Okutama-Action Dataset.** Except for 3DGS [10], all methods are designed for surface reconstruction. The "Mask" column indicates whether a mask is applied when measuring the training loss. Compared to 2DGS [11] and PGSR [12], our method enhances the PSNR by 1.23 dB and 2.79 dB, respectively. The best and second best-performing algorithms for each metric are bolded and underlined.

Method	Mesh	Mask	PSNR ↑ / SSIM ↑ / LPIPS ↓				
			Drone1-Morning	Drone2-Morning	Drone1-Noon	Drone2-Noon	Mean
3DGS [10]	X	O	26.33 / 0.901 / 0.222	27.32 / 0.891 / 0.216	31.47 / 0.917 / 0.177	34.10 / 0.942 / 0.175	29.90 / 0.913 / 0.198
GOF [52]	O	O	29.04 / 0.918 / 0.164	26.43 / 0.896 / 0.212	29.95 / 0.914 / 0.189	32.09 / 0.938 / 0.193	29.35 / 0.916 / 0.190
2DGS [11]	O	X	27.09 / 0.899 / 0.236	28.33 / 0.901 / 0.222	31.22 / 0.923 / 0.197	32.49 / 0.936 / 0.211	30.56 / 0.924 / 0.198
PGSR [12]	O	X	25.57 / 0.906 / 0.188	26.11 / 0.907 / 0.166	29.17 / 0.936 / 0.137	31.94 / **0.950** / **0.143**	28.25 / 0.925 / 0.155
PGSR [12]	O	O	27.71 / 0.920 / **0.142**	26.96 / 0.911 / **0.156**	29.47 / 0.937 / **0.137**	31.84 / 0.946 / 0.152	29.00 / 0.921 / **0.147**
MsGS (Ours)	O	O	**31.07** / **0.925** / 0.166	**29.89** / **0.918** / 0.161	**33.08** / **0.937** / 0.154	33.59 / 0.944 / 0.188	**31.79** / **0.932** / 0.162
MsGS w/o Refine	O	O	30.43 / 0.920 / 0.176	29.83 / 0.918 / 0.162	32.96 / 0.937 / 0.161	**33.60** / 0.941 / 0.178	31.75 / 0.929 / 0.166
MsGS w/o Mask	O	X	30.32 / 0.921 / 0.168	29.63 / 0.916 / 0.162	32.64 / 0.935 / 0.160	33.46 / 0.945 / 0.168	31.71 / 0.929 / 0.171

8.4.3 Scene Composition

To achieve high realism for scene composition, we require high dynamic range (HDR) lighting of the target scene to utilize as a virtual light source. We employ DiffusionLight [339], a diffusion-based

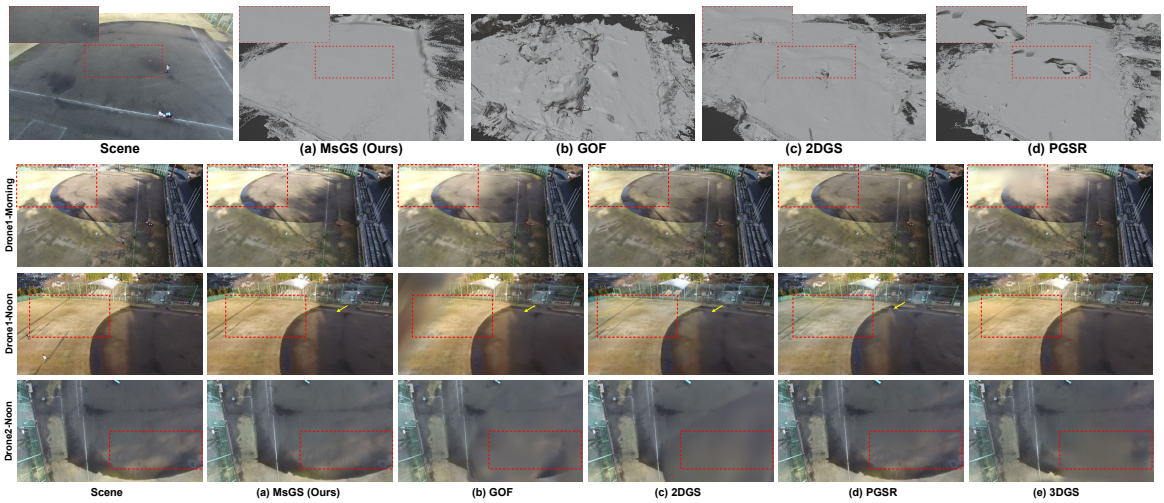


Figure 8.6: **Qualitative Comparison of MsGS (Our method), GOF [52], 2DGS [11], PGSR [12], and 3DGS [10].** In the first row, we show shading mesh without texture. The red dotted lines illustrates the superior geometric quality of our method. We visualize the rendering results (2-4 rows). Red dotted lines and yellow arrow emphasize difference in rendering quality.

lighting method, to generate a high dynamic range (HDR) map by inpainting a chrome ball into the image. The resulting environment map is then imported into the rendering engine as an environmental light source.

For foreground rendering, our approach utilizes the Blender's Cycles renderer [325] graphics engine to render foreground humans. We set the large plane under humans and use the shadow catcher module in Blender to create a shadow effect. We alpha-blend this foreground object I_{fg} with background rendered images I_{bg} from our neural renderer in Section 8.3.2. We also add motion blur $G(\cdot)$ with Gaussian kernel to rendered foreground humans.

$$I_{comp} = \alpha G(I_{fg}) + (1 - \alpha) I_{bg} \tag{8.9}$$

where α denotes the alpha map rendered by Blender. One key aspect we want to emphasize is that since the graphics engine can generate precise ground truth for segmentation masks, depth maps, and

action labels, we can effectively augment the dataset with highly accurate ground truth annotations.

8.5 Experiments

To evaluate the effectiveness of UAVTwin, we generate digital twins by reconstructing 3D scenes from UAV image collections. Our experiments are designed to evaluate two core components of our system: (1) the neural rendering module's ability to generate novel-view images and (2) the effectiveness of the synthesized dataset in enhancing UAV perception performance.

8.5.1 Datasets and Implementation Details

Okutama-Action: The Okutama-Action dataset [13] comprises UAV-captured video footage within a baseball stadium, depicting multiple human agents performing a range of single-agent and multi-agent actions. The dataset is collected using two UAVs, with altitudes ranging from 10 to 45 meters and camera angles set at either 45 or 90 degrees. They collected 25 scenarios under two different lighting conditions (sunny and cloudy) and at two time periods (morning and noon). Both time and lighting differences result in significant appearance variations. Additionally, each scenario covers only a limited area due to the constraints of a single sensor and limited power.

DroneSplat: The DroneSplat dataset [340] consists of six scenes captured by a single UAV, including dynamic elements such as humans, cars, and buses, commonly found in urban environments. For testing, they provide the novel-view scenes without dynamic objects.

NeRF-OSR: The NeRF-OSR dataset [341] is designed for outdoor scene relighting. Similar to the NeRF-MS [342], we select three sequences under different lighting conditions for each of the three scenes. For testing, we sampled every eighth frame.

Table 8.2: **Human Recognition Comparison on the Okutama-Action [13] of YOLOv8 family [14], R3D [15], Swin3D [16].** Compared to models trained with "Real" dataset, we improve the mAP_{50} metric by 2.5 % - 13.7 % and top-1 accuracy by 2% - 7%.

Train	Test	mAP_{50}			Video@1	
		yolov8n	yolov8s	yolov8m	R3D	Swin3D
Syn-N	Noon	40.8	39.7	44.2	40.2%	26.3%
R1.2.2	Noon	46.8	52.5	58.0	37.4%	38.7%
R1.2.2 + Syn-N	Noon	**50.2**	**53.4**	**61.5**	**42.4%**	**42.5%**
R2.2.2	Noon	40.7	43.1	51.6	36.7%	38.4%
R2.2.2 + Syn-N	Noon	**46.3**	**47.9**	**56.1**	**42.7%**	**40.2%**
Syn-M	Morning	19.7	16.8	31.3	33.5%	42.2%
R1.1.1	Morning	40.6	44.9	60.0	36.9%	42.7%
R1.1.1 + Syn-M	Morning	**43.1**	**48.1**	**61.5**	**42.6%**	**45.2%**
R2.1.1	Morning	30.6	33.6	35.2	28.5%	43.6%
R2.1.1 + Syn-M	Morning	**31.7**	**44.6**	**47.6**	**37.2%**	**51.1%**

Table 8.3: **Quantitative Comparison on NeRF-OSR Dataset.** We report PSNR ↑, SSIM ↑, and LPIPS ↓.

Method	stjohann	lwp	st	Mean
3DGS [10]	13.14/ 0.571 / 0.372	14.41 / 0.515 / 0.406	13.47 / 0.512 / 0.436	13.67 / 0.531 / 0.404
GOF [52]	11.67 / 0.542 / 0.411	13.90 / 0.527 / 0.420	13.18 / 0.528 / 0.428	12.91 / 0.528 / 0.428
2DGS [11]	10.77 / 0.521 / 0.415	14.31 / 0.530 / 0.411	14.31 / 0.569 / 0.420	13.13 / 0.54 / 0.415
PGSR [12]	11.52 / 0.544 / 0.407	14.26 / 0.494 / 0.420	14.17 / 0.504 / 0.407	13.31 / 0.514 / 0.411
MsGS (Ours)	**19.05 / 0.766 / 0.222**	**16.92 / 0.638 / 0.324**	**18.11 / 0.650 / 0.311**	**18.03 / 0.684 / 0.285**

8.5.2 Background Rendering

For evaluating the background rendering, we compare our method with four different methods. GOF [52], 2DGS [11], and PGSR [12] are state-of-the-art surface reconstruction methods built on 3D Gaussian splatting.

Comparison on Okutama-Action Dataset: In Table 8.1, we report the rendering quality using the standard PSNR, SSIM [163], and LPIPS [261] on UAV scenes in the Okutama-Action dataset. We calculate the average rendering metrics across 7 sequences for Drone1-Morning, 7 sequences for Drone2-

Morning, 6 sequences for Drone1-Noon, and 5 sequences for Drone2-Noon. Each sequence contains approximately 100 to 200 images. When applying masked loss during the training of 2DGS, we observe that the training process fails. Thus, except for 2DGS, we use the same refined mask for training for fair comparison. Since some test images include dynamic objects, we use masks for measuring metrics. Our method demonstrates better rendering quality compared to other recent methods such as 3DGS [10] (with a 1.89 PSNR increase), GOF [52] (with a 2.44 PSNR increase), 2DGS [11] (with a 1.23 PSNR increase), and PGSR [12] (with a 2.79 PSNR increase).

In Fig. 8.6, the first row presents the shading meshes, which provide the most effective way to highlight geometric differences. Our method produces a smooth surface on the ground, free from distortions commonly observed in other approaches. Notably, although both PGSR and our method incorporate multi-view geometric regularization from Sec. 8.3.4, sequence embedding plays a crucial role in eliminating floaters and geometric artifacts in the mesh. Rows 2–4 in Fig.8.6 present the rendering results. The red dotted boxes highlight regions where our method accurately captures the appearance, maintaining a color tone similar to the original scene. The yellow arrows indicate areas where strong mesh artifacts are prevalent in most algorithms, as well as rendering artifacts visible in rows 2 and 3 in Fig.8.6.

8.6 Conclusion

This paper introduces UAVTwin, a framework for data generation to train UAV-based human recognition algorithms by constructing a photorealistic digital twin using 3D Gaussian splatting. To promote photorealistic rendering, we propose MsGS for background rendering, which accommodates multi-sequence images with large appearance variations and dynamic objects. Additionally, we present

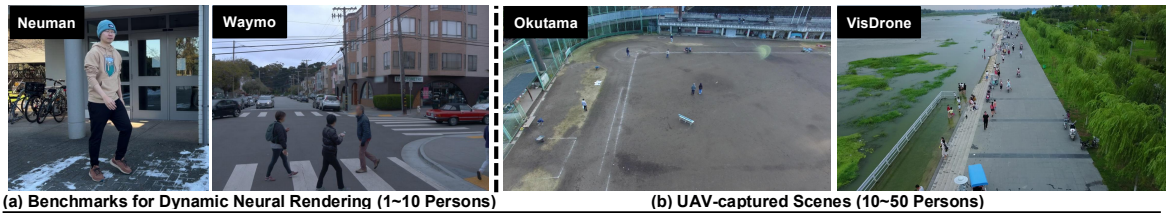
a neural data generation pipeline that integrates synthetic humans with background rendering, aiming to minimize the domain gap. However, our synthesized data still suffer from a domain gap, as synthetic humans visually differ from real humans, making it challenging to achieve significant improvements in perception performance. In the future, we will explore more carefully designed object insertion techniques or utilize realistic-looking human avatars reconstructed from real-human datasets to further reduce the domain gap.

Chapter 9: UAV4D: Dynamic Neural Rendering of Human-Centric UAV Imagery using Gaussian Splatting

9.1 Introduction

Unmanned Aerial Vehicles (UAVs) have become essential tools for capturing large-scale environments such as streets and urban areas, supporting applications like world mapping, human activity analysis, and security monitoring. Recent advances in neural rendering [10] have enabled photorealistic scene reconstructions from UAV data, driving progress in large-scale mapping. However, most of the existing works [104, 343, 344] primarily focus on static scenes, overlooking the more challenging dynamic scenes, even though real-world environments are inherently dynamic, involving human activities and complex background changes. Although dynamic neural rendering methods are rapidly evolving, most existing approaches [345] focus on phone-captured video scenes [105, 106, 346, 347], multi-view synchronized camera setups [348, 349], or driving scenario [350], as seen in Fig. 9.1. In particular, OmniRe [350] is designed for multi-sensor system scenarios, such as multi-camera rigs, a LiDAR sensor, and 3D bounding boxes. However, these setups are not accessible in UAV-based scenarios. In summary, dynamic neural rendering methods tailored for UAV-based scenarios remain underexplored. To the best of our knowledge, only a few works [104, 344] have attempted this, and they have some limitations. Some approaches use implicit neural representations [351] for volumetric rendering to generate novel-view images but face challenges in terms of maintaining high-fidelity outputs.

Applying dynamic neural rendering to UAV-based scenes presents two fundamental challenges. First, due to hardware constraints, UAVs typically lack access to a diverse range of sensors and cannot utilize multi-view synchronized camera systems or multi-sensor configurations. Consequently,



(a) Benchmarks for Dynamic Neural Rendering (1~10 Persons) (b) UAV-captured Scenes (10~50 Persons)

Method	Scene	Camera	RGB Only	Number of Humans	Human Size
4DGS [105]	Phone	Single	Yes	1	Large
DfGS [106]	Phone	Single	Yes	1	Large
OmniRe [350]	Driving	Multi	LiDAR	1-10	Medium
UAV4D (Ours)	UAV	Single	Yes	10-50	Small

Figure 9.1: (Top) Comparison between the Existing Benchmarks for Dynamic Neural Rendering and UAV captured Datasets. UAVs typically cover wide areas from a top-down perspective, capturing dynamic scenes with a distinct setup: they contain numerous human instances, each occupying only a tiny fraction of the image area. (Bottom) Comparison of various recent dynamic gaussian splatting methods. UAV4D is among the first to enable rendering of UAV scenes using only a single-view camera.

scene reconstruction must rely solely on monocular dynamic video, which presents substantial challenges [352] for both accurate geometry reconstruction and photorealistic rendering—particularly in dynamic environments. Furthermore, most dynamic objects in UAV-captured scenes are relatively small and multiple moving humans often appear simultaneously. These conditions pose significant challenges, making it difficult to apply the state-of-the-art approaches [346, 347, 353, 354] that rely on depth estimation [355], optical flow [356] and point tracking models [357]. Even recent 4D reconstruction methods [358, 359] powered by 3D foundation models [78] face difficulties when reconstructing small dynamic humans. This challenge arises primarily because existing large-scale datasets for dynamic scenes rarely include scenarios with small and moving human instances. OmniRe [350], tailored for driving scenes, leverages LiDAR and 3D bounding box information to freely localize dynamic foreground objects in world coordinates. However, in UAV-based scenarios where LiDAR is unavailable, human positions in world coordinates cannot be directly obtained. As a result, many of these previously mentioned methods are ineffective in UAV-based scenarios. Inspired by recent progress

in human-scene reconstruction [360], we focus specifically on reconstructing and rendering moving humans, which are a key component of dynamic scenes captured by UAVs.

Main Results: We present UAV4D, a novel neural rendering framework that reconstructs dynamic human instances and static backgrounds from UAV-captured real-world data, enabling temporally consistent and photorealistic novel-view synthesis. Our methods begin with reconstructing the joint geometry of small dynamic humans and static backgrounds from monocular UAV-captured videos by integrating SMPL-based human meshes with dense background geometry. We leverage the 3D foundation model [79] to estimate camera poses and perform dense background reconstruction, yielding consistent and robust background meshes across diverse scenarios. We observe that HMR2 [361] and SAM2 [362] exhibit strong performance in UAV contexts, and we adopt them to recover 4D human motion trajectories as temporally aligned SMPL body meshes [103] from monocular videos. However, the human and background meshes suffer from inconsistent scales, and the absolute global positioning of the human meshes remains unknown. To resolve these discrepancies, we propose a scale optimization technique that aligns the background geometry from the 3D foundation model to the metric human mesh, estimating an optimal global scale parameter. Using the optimized scale parameter, we unify the background and human meshes within a consistent world coordinate system over time. Finally, we localize the human mesh in the world space by unprojecting the 2D ground contact point into a 3D location on the reconstructed background mesh.

We then initialize separate 3D Gaussian representations for humans and background using their respective geometric reconstructions. We maintain separate Gaussian splats for humans and background, which are jointly optimized and composited to render photorealistic full frames. Our approach explicitly decomposes dynamic human instances and static backgrounds by independently optimizing their Gaussian splats. By initializing with a strong Gaussian prior from the human SMPL mesh, we

ensure that the optimization process properly learns the human Gaussian splats, rather than neglecting them due to their small pixel regions. The key contributions of this work include:

- We introduce UAV4D, the first neural rendering framework that reconstructs dynamic human motions and static backgrounds from UAV-captured monocular videos, enabling temporally consistent and photorealistic novel-view synthesis of full scenes.
- We propose a novel method for resolving the scale ambiguity issue in 3D foundation models and for aligning multiple human meshes with the background mesh. We also present a human placement method by identifying the ground contact point. By initializing both the human and background meshes in world coordinates, we decompose the human and background Gaussian splats to render the complete image.
- To the best of our knowledge, UAV4D is the first to enable dynamic Gaussian splatting for UAV-captured scenes with large number of small moving humans (e.g., up to 50). We evaluated its performance on three datasets with unique characteristics.

9.2 Related Work

3D Reconstruction for Static and Dynamic Scenes. Traditional 3D reconstruction typically consists of two components: Structure-from-Motion (SfM) and Multi-View Stereo (MVS) systems. SfM [29] extracts and matches features across multi-view images to estimate camera poses and sparse 3D points via triangulation [97]. MVS [141] then estimates dense depth maps for each image by performing an accurate and efficient global search for the optimal 3D plane hypotheses. Recently, learning-based SfM [363,364] and MVS [31,78,79,146,365] approaches have leveraged large-scale datasets to enable end-to-end differentiable SfM and dense reconstruction. Notably, DUSt3R [78] and MASt3R [366]

have emerged as breakthroughs by estimating globally consistent point maps from just two views without requiring any camera information. VGGT [79] also leverages point maps during training and modifies the network architecture by incorporating alternative attention mechanisms for improved global alignment. However, these methods are primarily designed for static scenes and struggle to handle dynamic components such as humans. Some works [20, 367] train monocular depth estimation networks using pre-computed optical flow, or jointly optimize depth and camera poses [368]. More recently, concurrent studies [359, 366, 369] have achieved remarkable ability in dynamic scenes by fine-tuning MASt3R [369] with large-scale dynamic datasets to estimate both camera poses and point maps. However, none of these previous methods perform well on UAV-captured datasets, as the humans in our data are relatively small compared to those in their training datasets. Therefore, we decompose the scene into static and dynamic components, leveraging the prediction results of VGGT for separate tasks within our pipeline.

Dynamic Neural Rendering. Earlier neural rendering methods were limited to static scenes, but researchers quickly extended them to handle dynamic scenes. One popular direction [105, 106, 346, 370] is deformation field-based methods, which often use a time-conditioned deformation network to warp 3D Gaussians into each time frame. Another line of work [371, 372] directly models 4D Gaussian primitives by integrating 3D Gaussians with the time dimension. Some approaches [346, 347, 353, 354] estimate 3D scene motion and point correspondences using pretrained models [355–357, 373, 374] to deform 3D Gaussians. HuGS [360] presents an approach similar to ours, using Gaussian splats to represent both humans and the scene. However, as their target scenes consist of phone-captured videos [375], they focus exclusively on a single, large person within the scene. Most of these methods are not designed for UAV-captured datasets, as they struggle to handle scenes with multiple pedestrians (more than 10 people) and very small human figures. UAV-Sim [344] and TK-planes [104] are

similar tasks to ours, as they use multi-resolution feature spaces in NeRF to represent small dynamic objects. However, we take a different approach by reconstructing the human and background meshes to initialize the Gaussian splats. We then render all the Gaussian splats to generate the full images.

9.3 Method

Our method takes as input monocular videos of human-centric scenes [13, 53, 54] captured by UAVs. Given this input, our goal is to enable dynamic neural rendering using 3D Gaussian splatting [10]. To achieve this, our approach involves initializing the human-scene from 2D images, reconstructing the 3D geometry of both the human and the scene, and optimizing the Gaussian splats for dynamic neural rendering.

9.3.1 Preliminaries

3D Gaussian Splatting (3DGS) [10] represents scenes as an unordered set of 3D Gaussian primitives, rendered through a differentiable rasterization process [274]. Each Gaussian component g_i is defined as its mean $\mu_i \in \mathbb{R}^3$ and 3D covariance $\Sigma_i \in \mathbb{R}^{3x3}$. The covariance matrix is decomposed into two learnable components, a scaling matrix $S_i \in \mathbb{R}^3$ and a rotation matrix $R_i \in \mathbb{R}^{3x3}$, as $\Sigma_i = R_i S_i {S_i}^T {R_i}^T$. Furthermore, each Gaussians store an opacity logit $o_i \in \mathbb{R}$ and color c_i defined by spherical harmonics (SH) coefficients. To render an image from a given view, the color of each pixel is computed by α-blending K ordered Gaussians using the following equation: $C = \sum_{i=1}^{K} c_i \alpha_i \prod_{j=1}^{i-1}(1-\alpha_j)$ where density α_i is computed by the multiplication of 2D Gaussians with covariance $\Sigma_i' \in \mathbb{R}^{2x2}$ and a learnable point-wise opacity o_i.

Skinned Multi-Person Linear (SMPL) Model [103] is a commonly used parametric human body

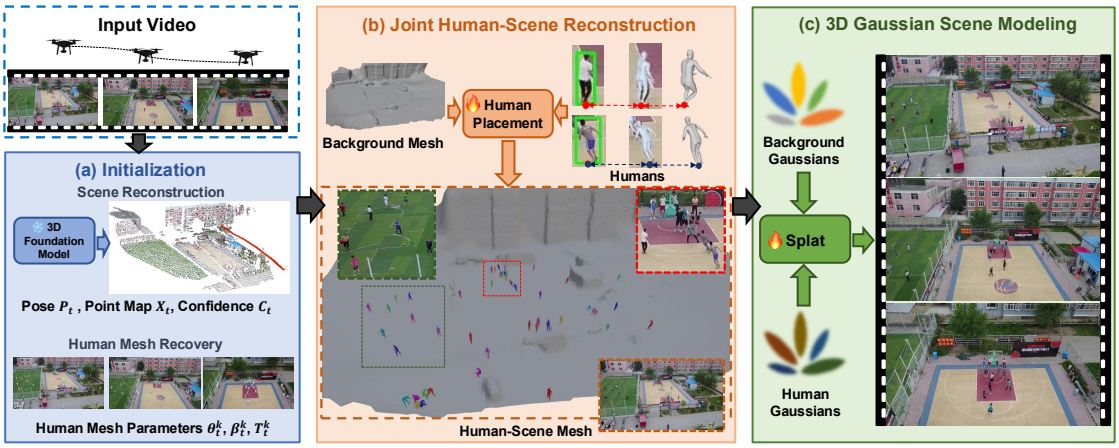


Figure 9.2: **Overview of Our Approach**: (a) We begin the initialization process for reconstructing the background scene and the human mesh. (b) We reconstruct the background mesh, determine the optimal scale for alignment, and position the human mesh by identifying the ground contact points. (c) We jointly optimize the human and background Gaussians for rendering.

model. The SMPL model exploits a template human mesh in the canonical rest pose, defined as $\mathcal{M}_h \in (\mathcal{V}_c, \mathcal{F})$ with vertices $\mathcal{V}_c \in \mathbb{R}^{6890\text{x}3}$. It takes body shape parameters θ, β as input and outputs a posed 3D mesh with deformed vertices, where $\theta \in \mathbb{R}^{24\text{x}3\text{x}3}$ and $\beta \in \mathbb{R}^{10}$ represent the pose and shape parameters, respectively. The pose parameters θ consist of the global rotation of the root joint (pelvis), and the 23 local rotations of other articulated joints relative to their parents along the kinematic chain. SMPL uses n_b predefined joints and Linear Blend Skinning (LBS) weights W, defined as $W(v_c) = \{w_1, ..., w_{n_b}\}$. The vertex v_c in the canonical template mesh can be deformed to the articulated space via the LBS, as $v = (\Sigma_{k=1}^{n_b} W_k(v_c) B_k) v_c$, where $B = \{B_1, ..., B_{n_b}\}$ is the target joint transformation.

9.3.2 Initialization

At this stage, we first utilize off-the-shelf models [79, 361, 362] to obtain initial estimates, such as camera poses, point maps, human masks, human meshes, for reconstructing the geometry of both the background scene and the human.

Scene. Given a video sequence $\left\{I_t \in \mathbb{R}^{H\times W\times 3}\right\}_{t=1}^{N}$, we first employ a 3D foundation model [79] to estimate camera poses $\{P_t\}_{t=1}^{N}$ including intrinsics K and extrinsics, rotation R_t and translation T_t, point map $X_t \in \mathbb{R}^{H\times W\times 3}$ and corresponding confidence $C_t \in \mathbb{R}^{H\times W\times 1}$.

Human. Due to the relatively small size of the human subjects, it is challenging to apply recent 3D tracking systems [361] trained on standard tracking tasks. Since our goal is precise reconstruction and rendering, we use bounding boxes $\left\{B_t^k\right\}_{k=1}^{K_t}$ as prompts for SAM2 [362] to generate accurate instance masks for K_t humans. These human segmentation masks $\left\{M_t^k\right\}_{k=1}^{K_t}$ are then fed into HMR2.0 [361] to predict camera-frame human meshes from each image I_t for K_t detected people. Thus, we can obtain the human mesh parameters for each human, defined as $\left\{\theta_t^k, \beta_t^k\right\}_{k=1}^{K_t}$ at timestep t.

Human Mesh Refinement. However, due to the challenging nature of UAV-based imagery, HMR2.0 [361] often produces noisy initializations. We introduce a refinement method to remove human mesh parameters with poor quality. Since the human meshes reside in the camera coordinate system, we project each individual human mesh into the image space and compute the bounding box $\left\{\bar{B}_t^k\right\}_{k=1}^{K_t}$ for each projected human. We then calculate the ratio between B_t^k and $\bar{B}_t^k$, and remove any human meshes for which $\bar{B}_t^k / B_t^k$ exceeds a threshold η_{box}. Furthermore, we measure the overlap between dilated human masks M_{t-1}^k, M_{t+1}^k from the previous and next time frames. If the current bounding box with missing human mesh parameters overlaps in both adjacent frames, we interpolate the missing parameters θ_t^k, β_t^k using the human mesh parameters from the previous frame $(\theta_{t-1}^k, \beta_{t-1}^k)$ and the next frame $(\theta_{t+1}^k, \beta_{t+1}^k)$ applying linear interpolation to both the quaternion pose parameters and the shape vectors.

9.3.3 Joint Human-Scene Reconstruction

Although our initialization provides a reasonable estimate of human poses, all predictions remain in the camera coordinate space. As a result, when the camera moves, the estimated human motion becomes implausible. To align the human and scene in the world coordinate, we reconstruct the background region and identify human-ground contact points to position the human mesh in the scene accurately. We observe that UAVs typically capture wide-area footage with the full human body visible, allowing us to leverage 2D bounding boxes to estimate the human-ground contact points. However, aligning the human mesh with the background remains challenging due to two key issues: (1) in complex environments, the reconstructed background geometry is often noisy and structurally intricate, making it difficult to reliably detect human-scene contact points; and (2) the inherent scale ambiguity of the 3D foundation model [79].

Background Geometry Reconstruction. To address the first issue, we first utilize all point maps $\{X_t\}_{t=1}^{N}$ and corresponding confidence map $\{C_t\}_{t=1}^{N}$ for background reconstruction. The point map represents a dense, pixel-aligned 3D location map for each corresponding image in the world coordinate frame. We stack all these point maps to represent the 3D geometry of the entire scene, although they include noisy points. Then, we compute the η_{conf} percentile of the pixel-aligned confidence values and use it to filter the point maps, removing noisy dynamic regions and intricate geometries with low confidence. The filtered point maps are then stacked, and Poisson surface reconstruction [191] is applied to generate the background mesh $\mathcal{M}_{bg}$. Thanks to this filtering, the background mesh is generated as a clean, flat ground surface, free from noisy dynamic human regions or complex structures that would otherwise make it difficult to identify human-scene contact points.

Scale Optimization. Furthermore, to address the scale ambiguity issue of the 3D foundation model

[79], we propose an alignment method between the background and the human mesh. Since the human mesh generated by SMPL [103] is defined in metric scale, we can align the background mesh relative to the metric human mesh. We introduce a scale parameter, σ, which adjusts the point maps X_t, background mesh $\mathcal{M}_{bg}$, and camera poses accordingly.

Our idea is to optimize the scale parameter from a *bone length* perspective. Inspired by [376], we leverage the correlation between observed 2D keypoints and 3D joints for this alignment. We first use ViTPose [377] to detect 2D keypoints j^k for each person k in the image. Then, we lift the 2D keypoints j_t^k to 3D joints using scaled point maps X_t, defined as $\hat{J}_t^k = \sigma X_t(i,j)$ where $(i,j) \in j_t^k$. Given the human mesh parameters θ_t^k, β_t^k from HMR2.0 [361], we can obtain the 3D location J_i of interest joint i from the mesh. Then we compute the 3D bone length $d_{(p,c)}^k$ for each joint pair (p,c) associated with the main body joints $\mathcal{J}_{body}$ of person k in the image at time t, as defined by:

$$\hat{d}_{(p,c),t}^k = J_p(\theta_t^k, \beta_t^k) - J_c(\theta_t^k, \beta_t^k), \quad d_{(p,c),t}^k(\sigma) = \sigma X_t(p) - \sigma X_t(c) \tag{9.1}$$

The loss function $L(\sigma)$ is computed as the L2 norm loss between the predicted 3D joint-based bone lengths $\hat{d}_{(p,c),t}^k$ and the corresponding ground-truth values $d_{(p,c),t}^k(\sigma)$. Specifically, the loss is calculated across all time steps t, individuals k, and joints (p,c) in the main body, where minimizing this loss ensures that the predicted bone lengths align closely with the ground truth. The scale parameter σ is optimized using the L-BFGS algorithm [378] applied to this loss. The loss function for optimizing the scale parameter σ is as follows:

$$L(\sigma) = \sum_t \sum_{k \in \mathcal{K}} \sum_{(p,c) \in \mathcal{J}_{body}} ||\hat{d}_{(p,c),t}^k - d_{(p,c),t}^k(\sigma)||_2. \tag{9.2}$$

The update rule is $\sigma_{n+1} = \sigma_n - \alpha \mathcal{H}_n \nabla L(\sigma_n)$ where α is the learning rate and, $\mathcal{H}_n$ is the approximation of the inverse Hessian. The optimization proceeds for T_{opt} steps, and the objective is minimized through gradient-based optimization.

Human Placement. After determining the optimal scale σ, we apply this scale to the background mesh M_{bg} and camera poses P_t to align them with the human mesh. In conjunction with the camera pose, it is necessary to initialize the root translation in the world coordinate system. Our approach takes advantage of the fact that UAV-captured images provide top-down views, which facilitate the identification of the ground contact point. This contact point serves as a plausible initial estimate for the root translation. Given the background mesh M_{bg}, we render the depth map D_t from the specified viewpoint. Using the 2D bounding box $B_t^k = (x_{\text{min}}, y_{\text{min}}, x_{\text{max}}, y_{\text{max}})$ projected from the human mesh, we detect the 2D contact point as $(x_c, y_c) = ((x_{\text{min}} + x_{\text{max}})/2,\ y_{\text{max}})$. Subsequently, we use the depth value from D_t to unproject this 2D contact point for each individual human into the world coordinate system as follows:

$$\psi_t = \sigma R^T [K^{-1} D_{x_c, y_c} [x_c, y_c, 1]^T] - \sigma R^T T. \tag{9.3}$$

ψ_t is the 3D ground contact point in the world coordinate. Given this, we can accurately position all individual human meshes within the background mesh, with all meshes defined in the world coordinate system, as seen in Fig. 9.3.

9.3.4 3D Gaussian Scene Modeling

We model the scene using distinct Gaussian representations for dynamic humans and the static background, ensuring both temporal coherence and rendering clarity. We employ standard 3D Gaussians to model the background Gaussians g_{bg} and the human Gaussians g_h. Both Gaussians g_h and

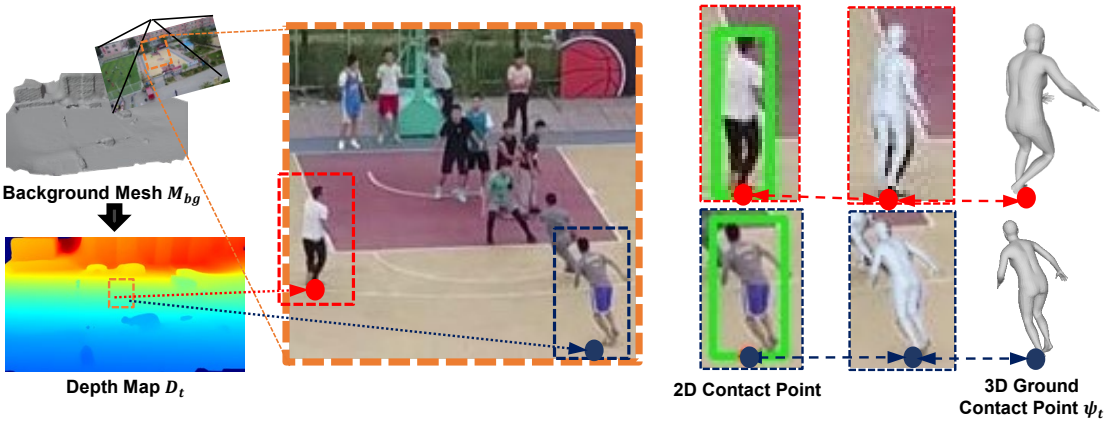


Figure 9.3: **Human Placement.** We identify the 2D contact point from the bounding box and use the depth from the mesh to unproject the 3D ground contact point.

g_{bg} contain learnable attributes (μ, r, s, c, o), which represent the 3D mean, 3D rotation, scaling factor, opacity factor, and color value, respectively. These are then merged into a unified human-scene primitive $g_{all} = g_{bg} + g_h$, which is subsequently fed into the Gaussian rasterizer [10] for rendering.

Background Gaussians. The background is represented by static Gaussians g_{bg} , which are initialized using stacked multi-view point maps X_t. For UAV scenes captured at high altitudes, we use all point maps without filtering. In contrast, for UAV scenes captured at low altitudes, we employ a confidence map to filter the point maps and use human masks M_t^k to remove point maps corresponding to human regions, stacking only the static background regions.

Human Gaussians. For human Gaussians, inspired by GART [379], we utilize the canonical template mesh $\mathcal{M}_h$ in the rest pose to represent the 3D Gaussian splats. The skinning transformation for each Gaussian, $A(t) = (R_A(t), T_A(t))$, is derived from the LBS weight $LBS(g_h, \theta_t)$, which transform the canonical Gaussian positions μ and rotations r to the world frame.

$$g_h(t) = (R_A(t)\mu + T_A(t) + \psi_t,\ R_A(t) \cdot r,\ s, c, o). \tag{9.4}$$

The changes in θ_t over time result in updates to the transformations of the key joints and linearly interpolates Gaussians to obtain the deformed position $\mu(t)$ and rotations $r(t)$ at different time steps.

Optimization. We optimize all Gaussian attributes g_{bg}, g_h, the human poses of all SMPL parameters θ_t, ψ_t for each frame t and the corresponding skimming weights.

The overall loss function for optimization is defined as:

$$L_{total} = (1 - \lambda_{pho})L_1 + \lambda_{pho}L_{ssim} + \lambda_o L_o + L_{smpl}, \tag{9.5}$$

where L_1 and L_{ssim} represent the photometric loss between the rendered full image and the ground truth image. L_{smpl} consists of various loss terms related to human Gaussian splats and human regions. We set λ_{pho} to 0.2 and λ_o to 0.05. Details are provided in the supplementary material.

9.4 Experiments

To measure the effectiveness of UAV4D, we evaluate our method on novel view synthesis tasks, using every 8th frame as the held-out test set. We report PSNR, SSIM, and LPIPS for full images and human-related regions, to assess dynamic reconstruction capabilities. Details of the datasets are provided in the supplementary materials. The human regions are obtained using a pretrained segmentation model which get a prompt from groundtruth bonding box.

Baselines: To evaluate rendering quality, we compare our method with four comparative methods: TK-Planes [104], 3DGS [10], DfGS [106], and 4DGS [105]. To the best of our knowledge, TK-Planes [104] is the only work that has specifically designed dynamic NeRFs for UAV scenes, while 4DGS [105] and DfGS [106] represent the state-of-the-art methods in dynamic Gaussian-splatting.

Table 9.1: **Quantitative Comparison** of our method with recent comparative works on the three datasets. Our method achieves the best performance across every category except for PSNR on the Manipal-UAV dataset. Please see the Appendix for detailed comparisons on the four scenes in each dataset. Red, orange, and yellow indicate the first, second, and third best performing algorithms for each metric, respectively.

Method	Okutama-Action			Manipal-UAV			VisDrone		
	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓	PSNR ↑	SSIM ↑	LPIPS ↓
TK-Planes [104]	28.05	0.796	0.417	27.98	0.718	0.396	23.98	0.644	0.450
3DGS [10]	29.90	0.867	0.228	29.46	0.856	0.150	24.59	0.782	0.220
4DGS [105]	25.90	0.780	0.410	30.87	0.843	0.253	22.58	0.628	0.440
DfGS [106]	22.64	0.748	0.443	31.22	0.894	0.114	15.46	0.522	0.614
UAV4D (Ours)	30.36	0.875	0.184	30.94	0.897	0.084	26.03	0.800	0.147

Table 9.2: **Quantitative Results.** (a) shows PSNR and SSIM across datasets, focusing on **human-only** regions cropped using precise human masks. Due to the small size of the human regions, it is not possible to compute the LPIPS metric. Consequently, we report only the PSNR and SSIM metrics. (b) presents ablation on VisDrone dataset. See Appendix for details.

Method	Okutama-Action		Manipal-UAV		VisDrone	
	PSNR ↑	SSIM ↑	PSNR ↑	SSIM ↑	PSNR ↑	SSIM ↑
TK-Planes [104]	19.14	0.631	18.38	0.752	17.65	0.544
3DGS [10]	19.07	0.656	17.99	0.748	16.51	0.548
4DGS [105]	18.18	0.602	19.23	0.770	16.25	0.510
DfGS [106]	16.33	0.571	18.65	0.762	12.59	0.354
UAV4D (Ours)	19.49	0.664	19.38	0.784	17.98	0.604

(a) Quantitative Comparison for human-only regions across datasets.

Method	PSNR ↑	SSIM ↑	LPIPS ↓
Ours	26.03	0.800	0.147
Ours wo SMPL	24.33	0.763	0.212
Ours wo Scale	23.02	0.651	0.25
Ours wo Refine	25.79	0.772	0.165

(b) Ablation Study on VisDrone dataset only.

9.4.1 Quantitative Comparison

In Table 9.1, we present the rendering quality of novel-view synthesis across three datasets. We compute the average rendering metrics across 4 scenes for VisDrone [53], 4 scenes for Manipal-UAV [54], and 4 scenes for Okutama-Action [13]. Due to the small size of the human subjects in UAV datasets, the rendering metrics for the full image do not reflect the dynamic rendering quality of the human regions directly. Moreover, since the Manipal-UAV dataset was captured at an exceptionally high altitude, the rendering metrics of the full image are less informative regarding the human regions and instead provide a better indication of the background region's rendering quality. In con-



Figure 9.4: **Qualitative Comparison of VisDrone [53] (top row) and Okutama-Action Dataset [13] (bottom row).** We visualize the zoomed-in blue and red regions, which emphasize the dynamic humans. Our method demonstrates superior capability in reconstructing small, moving humans compared to other existing approaches.

trast, the VisDrone dataset contains multiple humans, with relatively larger pixel regions compared to other datasets, allowing the full image rendering quality to be more directly correlated with the holistic scene rendering quality. Furthermore, certain scenes in VisDrone feature a significant number of moving pedestrians (approximately 30–50 individuals), presenting a particularly challenging scenario for dynamic reconstruction. Our method demonstrates superior rendering quality compared to other recent approaches, underscoring its robustness. In the Okutama-Action dataset, our method also exhibits better rendering quality than alternative methods. The Okutama-Action dataset is characterized by substantial camera variation and distinct viewing angles, which can destabilize deformation-field-based methods. However, our method shows strong resilience in handling these dynamic scenes, which involve substantial pose variation and changes in viewing angle.

In the VisDrone and Okutama-Action datasets, we observe that the training of 4DGS [105] and DfGS [106] is unstable, with the training of deformation-based fields often failing. We attribute this instability to the presence of multiple small moving objects within the scenes, which indicates that a single deformation field is insufficient to capture the required deformations accurately. As a result, the training process for these models becomes highly unstable. In contrast, TK-planes [104] demonstrates

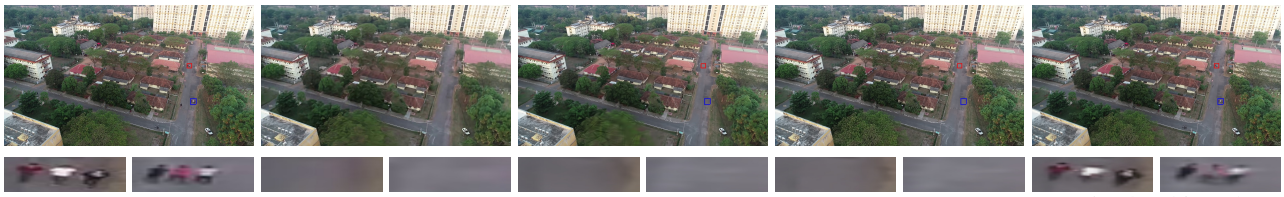


Figure 9.5: **Qualitative Comparison of Manipal-UAV Dataset [54].** We visualize the zoomed-in blue and red regions, which emphasize the dynamic humans. Our method demonstrates superior capability in reconstructing small, moving humans compared to other existing approaches.

greater robustness across various dynamic scenes, yielding superior performance. For the Manipal-UAV dataset, our method outperforms others in most cases, except for the PSNR values for DfGS [106]. We observe that DfGS [106] delivers superior rendering performance in regions with dense foliage and moving vegetation (e.g., swaying leaves). Our method currently struggles to represent such deformations, particularly in swaying trees, which cover large areas in UAV-captured datasets.

Table 9.2-(a) reports the rendering quality across three datasets, focusing on human-only regions cropped using precise human masks. Due to the small pixel size of the human regions, we are unable to use the LPIPS [261] metric, which relies on pretrained neural networks to measure metrics. Across all datasets, our method demonstrates superior rendering quality for human regions compared to other approaches. This indicates that our strategy effectively reconstructs and renders dynamic humans.

Ablation Study Table 9.2-(b) presents the results of an ablation study, averaged across four scenes in the Visdeon dataset. "Ours wo SMPL" indicates the version where we did not use human Gaussian splats, thus reconstructing only the background splats, which limits the model to static humans and removes moving people. "Ours wo Scale" refers to the version where the human mesh is placed without scale optimization. In this case, we set the initial scale σ to 40 and place the human mesh without any scale adjustment, leading to misalignment between the human regions and the human mesh. "Ours wo Refine" follows the full pipeline but omits the human mesh refinement process described in the the



Figure 9.6: **Scene Editing Samples**: Due to the nature of scene decomposition, we can move or remove the humans within the black-dotted boxes. The fourth image illustrates the decomposition of the background and humans. For additional visual results, please refer to the supplementary materials.

Initialization section. This version shows a slight improvement in artifacts caused by inaccurate in the human mesh.

9.4.2 Qualitative Comparison

Figure 9.4 presents visual examples from the VisDrone dataset [53]. Overall, our method achieves the highest rendering quality, producing a photorealistic background and sharp human regions, where the detailed shapes of the humans are more accurately recovered compared to other methods. TK-planes [104] generates blurred renderings, while 3DGS [10] is unable to represent moving individuals, although it successfully reconstructs clear background regions. The training of 4DGS [105] occasionally fails. A similar trend is observed in the Okutama-Action dataset [13], as shown in Figure 9.4. Figure 9.5 displays examples from the Manipal-UAV dataset [54]. Our method can reconstructs extremely small humans, occupying only around 10x10 pixels, outperforming all other methods. However, DfGS [106] shows better performance in rendering swaying leaves and trees, which are frequently observed in this dataset.

Scene Editing Since our Gaussian scene is composed of separate human and background Gaussians, our system enables scene editing. As demonstrated in Fig. 9.6, we can easily remove, and translate humans, thanks to the decomposition of the foreground and background. This capability can be extended

to downstream applications, such as simulating dynamic scenes captured by UAV that are expensive to capture in the real world.

9.5 Conclusion

We have introduced UAV4D, a method for dynamic Gaussian splatting in UAV-captured environments. We propose a scale optimization and human placement technique to reconstruct human-scene geometry in the world coordinate system. This human-scene geometry is then used to initialize 3D Gaussian splats for dynamic scenes. It is important to note that the explicit representation of human geometry in our method is crucial for accurately modeling small, moving humans, ensuring they are not neglected during the optimization of Gaussian splats. Our joint optimization approach demonstrates improved performance in novel view synthesis, both for background and human regions.

In future work, we plan to modify our human mesh refinement algorithm to better align with 2D images and incorporate segmentation information to represent other dynamic components using dynamic Gaussian splats. Our approach can be combined with downstream UAV applications such as person detection [344] and human action recognition [13].

Chapter 10: Conclusion, Limitations, Future Works

10.1 Summary

In this thesis, we have presented a unified approach to photorealistic 3D mapping that advances sensing, mapping, and generation as tightly coupled components of a real-to-simulation pipeline. By addressing key limitations in depth sensing, geometric reconstruction, rendering fidelity, and scalable data generation, our work contributes practical methods for building digital twins that support a wide range of downstream applications in UAV perception systems.

In sensing and perception, we demonstrated that accurate metric depth estimation can be achieved without dense ground-truth supervision by leveraging self-supervised learning and sparse geometric priors. Through SelfDeco and DnD, we showed that monocular depth estimation can be made robust to challenging conditions, including dynamic scenes, complex ego-motion, and non-rigid human motion, while consistently outperforming prior approaches. These results highlight the growing potential of learning-based depth estimation to complement, and in some cases replace, active depth sensors in real-world deployments.

In mapping, we explored how classical geometry and graphics pipelines can be integrated with neural representations to reconstruct accurate geometry and enable photorealistic rendering across both object-level and scene-level settings. Our contributions span mobile object capture, compact mesh-based neural rendering for unbounded scenes, and hybrid mesh–splat representations. Together, these works demonstrate that neural 3D representations provide a powerful alternative to classical reconstruction and texture mapping methods, addressing limitations such as noisy surfaces and texture bleeding artifacts. At the same time, our results also show that classical methods remain more robust and reliable in certain settings, particularly in terms of geometric consistency, scalability, and

efficiency. In particular, classical mesh decimation continues to be an effective strategy for achieving compact and efficient geometric representations. This observation underscores the importance of hybrid approaches that combine classical and neural techniques to improve reconstruction completeness, reduce storage and computational overhead, and enhance rendering quality.

We further showed that omnidirectional 360° cameras offer important advantages for large-scale indoor mapping that conventional perspective cameras cannot easily provide. While research on perspective cameras is relatively mature, it faces inherent hardware-imposed limitations. Looking ahead, developing algorithms tailored to specialized sensing modalities, such as fisheye and 360° cameras, represents a promising direction for pushing the boundaries of mapping research. These advances further contribute to making high-fidelity digital twins practical in large-scale and resource-constrained settings.

In generation, we demonstrated how reconstructed digital twins can be leveraged to synthesize high-quality training data for perception models, particularly in challenging UAV-based scenarios. Through UAVTwin and UAV4D, we showed that combining neural rendering with controllable dynamic agents enables effective data augmentation and robust novel-view synthesis in scenes with complex motion, top-down viewpoints, and dense human activity. These results emphasize the role of digital twins as a bridge between real-world data collection and scalable learning in simulation.

Overall, this work advances the design of photorealistic mapping systems that are accurate, efficient, and scalable. By tightly integrating sensing, mapping, and generation, we show that digital twins can function not only as visual reconstructions of the physical world, but also as actionable representations that support robust perception, learning, and deployment in real-world environments.

10.2 Limitations and Future Works

While this thesis advances the photorealistic 3D mapping across multiple aspects, several limitations and open problems remain:

10.2.1 Sensing and Perception:

Although SelfDeco achieves accurate depth completion without dense supervision, its current training strategy is primarily tailored to indoor environments and may not generalize well to more diverse settings, particularly outdoor scenes where large sky regions dominate the upper portions of the image. Improving robustness across varied environmental conditions remains an important direction for future work. For DnD, the sparse depth maps projected from incomplete 3D reconstructions can be inaccurate or misaligned with the input image, especially in highly dynamic scenes. In addition, the method relies on an offline-reconstructed 3D map to obtain sparse geometric priors, which limits its scalability and applicability in online or long-term deployment scenarios. Reducing or removing this dependency by incorporating online mapping techniques represents a promising avenue for improving robustness and scalability.

More broadly, recent advances in 3D foundation models for monocular depth estimation and multi-view reconstruction suggest several important directions for future work. While existing models achieve strong performance by relying on large-scale transformer architectures and extensive supervised training data, this dependence on dense ground-truth depth limits their adaptability across environments and deployment settings. Future research should therefore focus on expanding the role of self-supervised training to improve generalization and flexibility.

In particular, self-supervised learning offers promising opportunities for pre-training, post-training,

and fine-tuning depth models without requiring dense annotations. We expect that combining self-supervised objectives with additional geometric priors, such as depth sensors, 3D LiDAR, and metrically consistent pose estimates from IMU, will be critical for enabling metric-aware adaptation of monocular depth foundation models. Such approaches can help bridge the gap between 3D foundation models and metrically-accurate depth estimation systems that are suitable for real-world deployment.

10.2.2 Mapping:

In TMO, while neural implicit representations significantly improve geometric completeness and visual quality, training remains substantially more time-consuming than classical reconstruction pipelines. Reducing training time through more efficient optimization strategies, improved initialization, or hybrid neural–classical pipelines would be critical for practical adoption. For LTM, although compact mesh representations are achieved, limitations arise from the underlying appearance model inherited from MipNeRF. Incorporating more expressive appearance models could further improve rendering fidelity. Additionally, the gradient-based mesh optimization stage may still introduce mesh artifacts such as self-intersections, suggesting the need for stronger geometric regularization or topology-aware optimization.

To address limitations in appearance modeling for LTM, we integrate Gaussian splatting with compact mesh representations in MeshGS. While this hybrid approach improves rendering quality, it introduces new challenges. Compared to traditional textured meshes, MeshGS requires larger disk storage and high-end GPUs for real-time rendering. Moreover, accurately assessing Gaussian splat quality and ensuring tight binding between splats and the mesh surface remain challenging. The approach is also constrained by the quality of the underlying mesh, particularly when large or irregular

faces hinder effective Gaussian initialization.

For IM360, reconstruction quality degrades as scene scale increases beyond a certain threshold. Improving scalability for very large indoor or mixed indoor and outdoor environments remains an open challenge and may require hierarchical scene representations or multi-resolution optimization strategies. Furthermore, rendering quality remains far from photorealistic due to extremely sparse viewpoints and limited visual overlap. Addressing these challenges will be essential for extending omnidirectional mapping to more complex and large-scale environments.

In summary, classical 3D reconstruction methods possess several well-established strengths, including modular design, high geometric accuracy when dense views with moderate baselines are available, and extensive optimization and refinement over structure and motion. However, these methods also suffer from well-known limitations, such as sensitivity to illumination changes, textureless regions, and low visual overlap. Our research shows that neural surface reconstruction techniques have significant potential to address many of these challenges. More recently, 3D foundation models have demonstrated the ability to infer multiple components essential to 3D vision, including feature matching, geometry, and relative camera poses, in an end-to-end manner across multi-view images. These outputs can be used to generate meshes or other 3D representations that support real-to-simulation pipelines. Nevertheless, understanding how to reliably integrate these foundation models with classical reconstruction systems remains an important direction for future work.

We have also explored a range of neural rendering techniques and shown their effectiveness in achieving photorealistic visualization. However, representations such as meshes or Gaussian splats are often optimized primarily for rendering and do not directly support broader functionalities required in graphics and robotics applications. Enabling operations such as rigging, editing, physics-based interaction, and relighting requires explicit geometric structure and semantic organization. Since many tradi-

tional graphics pipelines continue to rely on mesh-based representations, reconstructing both accurate geometry and expressive neural appearance remains necessary. Our work on MeshGS establishes an initial foundation in this direction, but further research is needed to support richer interactions and scene understanding. Promising future directions include articulation-aware modeling, semantic decomposition of scenes, and improved relighting techniques for Gaussian splatting representations.

10.2.3 Generation:

Although UAVTwin enables effective data augmentation, the synthesized data still exhibits a domain gap due to visual discrepancies between synthetic and real humans, which limits gains in downstream perception performance. Reducing this gap remains a critical direction for future work and may be addressed through improved human appearance modeling, domain adaptation techniques, or learning-based human synthesis approaches. For UAV4D, our method relies on pretrained components such as human mesh reconstruction and segmentation models. Inaccuracies in human mesh reconstruction can propagate to rendering artifacts in dynamic scenes. In addition, dynamic scene elements beyond humans, including bicycles or swaying vegetation, are not explicitly modeled and are currently treated as static background Gaussian splats. This limits the fidelity of dynamic scene reconstruction in complex real-world environments.

Looking forward, we plan to explore more carefully designed object insertion strategies and to incorporate realistic human avatars reconstructed from real-world datasets to further reduce the domain gap in synthesized data. We also aim to improve human mesh refinement by better aligning reconstructed meshes with 2D image evidence and by incorporating segmentation cues to explicitly model other dynamic components using dynamic Gaussian splats. Extending the pipeline to support

multiple categories of dynamic objects remains an important avenue for future research. Furthermore, we envision combining our generation pipeline more closely with downstream UAV applications, such as person detection and human action recognition, to enable task-aware data synthesis.

Finally, while our generation approach is primarily evaluated in UAV scenarios, its applicability extends beyond aerial platforms. One promising direction for future work is robot learning, where real-to-simulation pipelines play a crucial role in scalable training and evaluation. Applying our digital twin–based generation pipeline to robotic manipulation and navigation tasks represents an exciting opportunity to broaden the impact of this work.

Chapter A: Appendix for Dense Depth Estimation in Crowded Indoor Dynamic Scenes

A.1 Datasets

We provide a brief overview of the NAVERLABS Indoor Localization dataset [93, 166] and the specific properties that make it desirable as a dense depth estimation in crowded indoor dynamic scenes. The NAVERLABS dataset consists of various video sequences captured on five different floors in a department store and a metro station. Among these places, we selected two places, the metro station B1 (MS) and the department store B1 (Dept), to build a benchmark for depth estimation in crowded indoor dynamic scenes. The MS dataset is one of the most crowded dataset in the NAVERLABS dataset. We also chose the Dept dataset because it has totally different environments. In the NAVERLABS dataset, a mapping robot utilizes two 16-channel LiDAR sensors, six RGB cameras and four smartphone cameras. We only utilized images collected from six RGB cameras. LiDAR sensors are only used for generating groundtruth depth maps for evaluation. As the NAVERLABS dataset consists of the split of 60735 images for training and 835 images for testing in MS, and 25083 images for training and 443 images for testing in Dept, we adopt this dataset for dense depth estimation in crowded indoor dynamic scenes.

Dense depth estimation in crowded indoor dynamic scenes is a significant task for Robotics and AR applications. To the best of our knowledge, however, it is difficult to find large-scale public datasets for dense depth estimation in crowded indoor dynamic scenes. In Table A.1 (a), most indoor

Dataset	Environment	Location	Dynamic	Video	# Images	Metric	Annotation
(a) Indoor datasets							
NYUv2 [134]	Indoor	Small Rooms		✓	407K	✓	RGB-D
ScanNet [44]	Indoor	Small Rooms		✓	2.5M	✓	RGB-D
Stanford [167]	Indoor	Small Rooms		✓	72K	✓	Laser
Matterport3D [7]	Indoor	Small Rooms		✓	194K	✓	Laser
7 scenes [168]	Indoor	Small Rooms		✓	26K	✓	Laser
InLoc [169]	Indoor	Univ. bldg.		✓	10K	✓	Laser
Baidu [17]	Indoor	Mall	✓	✓	682	✓	Laser
(b) Internet Image collections							
DIW [71]	Indoor & Outdoor	Diverse	✓		495K		Ordinal
MegaDepth [69]	Outdoor	Diverse			130K		SfM
ReDWeb [153]	Indoor & Outdoor	Diverse	✓	✓	3600		Stereo
WSVD [164]	Indoor & Outdoor	Diverse	✓	✓	150K		Stereo
MC [32]	Indoor & Outdoor	Diverse	✓	✓	136K		
(c) **NAVERLABS**							
Dept [93, 166]	Indoor	Mall	✓	✓	25K	✓	Laser
MS [93, 166]	Indoor	Mall & Turnstiles	✓	✓	60K	✓	Laser

Table A.1: (a) Most large-scale indoor datasets do not have dynamic scenes except for Baidu [17]. (b) All datasets collected from the internet do not contain metric 3D models. To train depth estimation algorithms in crowded indoor dynamic scenes, a dataset must contain three properties: dynamic, metric and large amount of images. Different from previous datasets (a) and (b), NAVERLABS contains large amounts of dynamic scenes with the metric 3D model in indoor environments.

datasets [7, 17, 44, 134, 167–169] are captured from small collections of room, office, and university buildings. They cover only a restricted scale of spaces and have similar design and internal structures. Additionally, existing indoor datasets do not provide scenes containing multiple dynamic objects such as moving people except for the Baidu dataset [17]. The Baidu dataset [17] collected from a shopping mall has dynamic scenes with 3.75% crowd density. However, Baidu contains only 689 images for training. In Table A.1 (b), other works [32,69,71,153,164] explore the use of internet photo collections to train the depth estimation models for dynamic scenes. These datasets contain a large number of dynamic scenes, but they do not provide metric depth. They often have relative depth obtained by either stereo matching or COLMAP [29, 30], or ordinal depth relation from manual annotations. Thus, we are not able to evaluate our algorithms using metric depth maps for internet images.

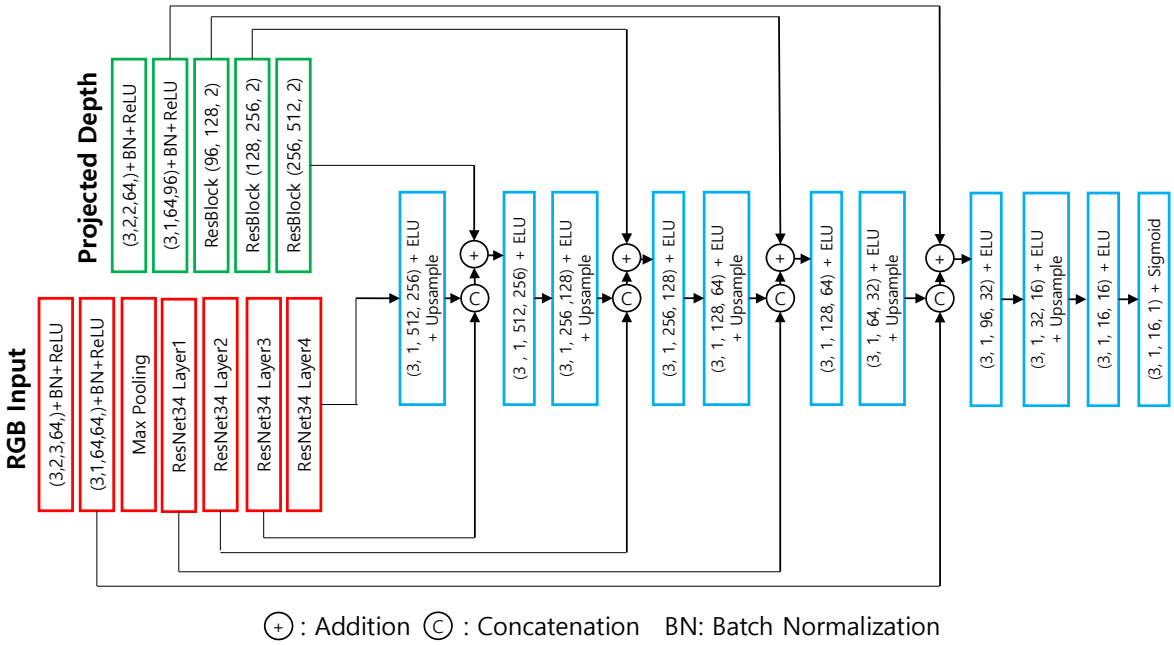


Figure A.1: The network architecture of our proposed method. The red block, green block, and blue block are an RGB encoder, a depth encoder, and a decoder. We concatenate a projected depth map and a binary mask, which indicates where the projected depth values exist. The convolutional filter is defined as (filter size, stride size, input channel, output channel). The RGB encoder adopts a ResNet 34 [55] backbone pretrained with ImageNet [56], and the depth encoder consists of two convolutional blocks and three residual blocks.

A.2 Evaluation Metrics

The depth value is represented in the metric scale (m). Following [148], we evaluate our method using the following metrics: absolute relative difference (Abs Rel), square relative difference (Sq Rel), root mean squared error (RMSE), RMSE in logarithmic scale (RMSE log), and δ_i meaning the percentage of predicted pixels for which the relative error is less than a threshold i.

A.3 Implementation Details

We implement our method in PyTorch [380] and conduct all experiments on a V100 GPU. We train our network with a batch size of 8 images with size 1024×768 for 20 epochs. We use the fol-

lowing set of weights for each loss term in the loss function: λ_d = 0.001, λ_{ph} = 1, λ_s = 0.3, λ_f = 0.1, and λ_n = 0.001. We utilize the Adam [137] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We use an initial learning rate of 10^{-4} for the first 10 epochs and halve it for the remaining 10 epochs. We use human masks to eliminate the points in human regions to reduce the noise on projected depth maps.

A.4 DnD with Localization

DnD network requires a monocular image and the corresponding reprojected depth map derived from the 3D model with the camera position. In order to verify the applicability, it is necessary to demonstrate both pose estimation and depth map projection about the new test images. Thus, we implement the whole pipeline of DnD, including mapping and localization process using Kapture [18], which is the visual localization toolbox. As a mapping procedure, the sparse 3D model is reconstructed via Structure-from-Motion (SfM) [29] in the set of training images. When a test image is given, image retrieval (AP-GeM [381]) is performed to obtain top k ranked images in the database of the 3D model, which can consider covisibility with the images and estimate a more accurate camera pose. The local feature extractor (R2D2 [129]) produces 2D-2D matches between the test image and top-ranked database images, resulting in 2D-3D matches between the test image and the 3D model. Perspective-n-Point (PnP [125]) problem with Random Sample Consensus (RANSAC [126]) is solved in these matching points and consequently yields the predicted camera pose. With the visible 3D points from the test image and the estimated pose, we can obtain the reprojected depth map and activate our DnD framework. We assign 5 for the top k, and the results of the MS dataset are reported in Fig. A.2 and Table A.2.

Method	3D Model	F+B / F (Lower is better)				F+B / F (Higher is better)		
		Abs Rel	Sq Rel	RMSE	RMSE log	$\delta_{1.25}$	$\delta_{1.25^2}$	$\delta_{1.25^3}$
DnD	SfM + MVS	0.189 / 0.240	0.20 / 0.16	1.76 / 2.44	0.084 / 0.133	0.806 / 0.677	0.881 / 0.798	0.919 / 0.856
DnD	SfM	0.194 / 0.249	0.17 / 0.15	1.85 / 2.56	0.092 / 0.141	0.774 / 0.638	0.874 / 0.785	0.918 / 0.850

Table A.2: The performance with or without Kapture [18] in the MS dataset. The upper line in the table shows the results of the camera poses, which are used in reconstruction of the 3D model (SfM + MVS). The lower line in the table indicates the performance of the whole pipeline including visual localization under the sparse 3D model from SfM (without MVS). Although the SfM point clouds are highly sparse and visual localization methods has pose uncertainty, the comparable results show the robustness of our framework.

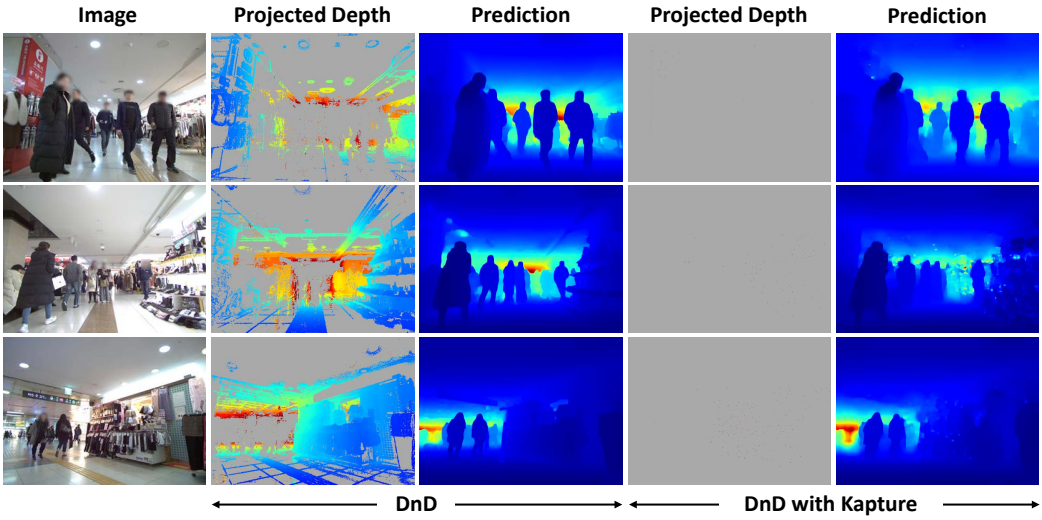


Figure A.2: DnD (2-3 columns) estimates depth from the image and the projected depth map, which is derived from the 3D model. Since DnD with Kapture [18] (4-5 columns) exploit the sparse 3D model from SfM, the projected depth maps are also highly sparse, and the background details of the estimated depth maps decrease. However, the overall qualities, especially moving people, are predicted robustly. (low high; grey means empty depth values.)

Chapter B: Appendix for Textured Mesh Acquisition of Objects with a Mobile Device by using Differentiable Rendering

B.1 Data Collection

We provide specific properties of our data collection process. As mentioned in the manuscript, ARKit-video collects long video sequences to apply the RGBD SLAM methods [181] to extract poses and fuse a 3D model. However, we fail to apply the state-of-the-art RGBD SLAM algorithm [181] for accurate pose estimation due to noisy depth measurements. Instead, we utilize accurate poses from our RGBD-aided Structure from Motion (SfM) for geometry reconstruction and texture optimization. ARKit-video includes *recliner chair*, *cafe stand*, *delivery robot*, *office chair*, and *camera stand*. We can apply TSDF-fusion [36] with poses from RGBD-adied SfM to reconstruct geometry in Fig. 6. in the manuscript because of the video sequence. AR-capture includes *robot arm 1*, *robot arm 2*, *plant*, *tree*, *sofa*, *bike*, *recliner chair*, *cafe stand*, *delivery robot*, *office chair*, and *camera stand*. Except for *sofa* and *tree*, AR-capture contains less than fifty images. The *sofa* contains 108 images because the mobilephone camera has difficulty capturing this sofa due to the large size (This sofa can seat up to 15 people). We require a number of images to capture the whole shape of *sofa*. The *tree* contains 111 images and is also large size object. In addition to this, the *tree* has a very complex shape (e.g. leaf) and thin structure (e.g. stem).

B.2 Implementation Details

In preprocessing stage, we apply both single-view and multi-view filtering two times. We regard 10 neighboring frames as source images and set the threshold ϵ to 0.1. We keep depth pixels if the ratio of inliers is greater than 80%. We remove depth points larger than 2.55m. Our RGBD-aided Structure from Motion (SfM) is based on COLMAP [29] implemented in C++. Similar to the existing COLMAP, we use Ceres Solver to perform bundle adjustment. After SfM, we apply MVS algorithm [30, 34] to provide the depth and normal for geometry reconstruction. Here, we also leverage the filtering algorithm to remove the noisy values for the depth and normal maps obtained from the MVS algorithm. For the first stage of trainning process in Sec. 3.2, we set w_d to 1e-5 and w_n to 1e-3. After the first stage, we build sparse voxel volume around surfaces obtained from mesh. We set a depth level of octree to 8 for all classes. During the training for two MLPs (the SDF network and color network), we also model the background by NeRF++ [197]. The sampling strategy is also similar to the NeuS [39]. For geometry reconstruction and texture optimization, we use 41 - 108 images at 1280 x 720 pixels sampled from video sequence of ARKit-video. Since AR-capture has large images at 4032x3096 resolutions, we resize their resolution to 1008x774 resolution for geometry reconstruction and texture optimization. The LiDAR sensor in the iOS device provides depth maps and confidence maps with size 256 x 192. For mesh simplification, we exploit the quadric mesh simplification algorithm provided by [382]. The learning rates for texture optimization start from 1e-3 and decay the learning rate by 0.1 once the epoch reaches 500 and 2000 respectively. As mentioned in manuscript, we use the public github code for the classical MVS [30, 34] and texture mapping [9] which are implemented in C++. We implement all geometry reconstruction and texture optimization in PyTorch [380] and conduct all experiments on V100 GPU.

Chapter C: Appendix for Lightweight Textured Mesh Extraction and Refinement of Large Unbounded Scenes for Efficient Storage and Real-time Rendering

C.1 Implementation Details

For geometry reconstruction in Section 3.1, the model structure of geometry model f_{sdf} consists of the grid feature and shallow MLPs. Following the BakedSDF paper, color model g_{color} adopts the appearance model of Ref-NeRF [383]. For mip-NeRF 360 dataset and Tanks & Temples, we set 0.01 to a weight for eikonal regularizaiton except for room scene. We use 0.1 for eikonal regularization for the room scene and two scenes from Deep Blending dataset [6]. For modeling appearance, we build on training techniques from mip-NeRF 360 [88] except for integrated positional encoding. We set the distortion loss weight to 0.002. Our learning schedule is 20k iterations of optimization using Adam [137], a learning rate is annealed from 1e-3 to 1e-4 To design the model structure, we adopted the appearance and density model of NeRF2Mesh [42] because this model has a lightweight structure and can generate diffuse texture and specular features. We use multiresolution hash encoding [196] with hash table size 2^{16}, coarsest resolution of 16, highest resolution of 4096, 16 levels and a number of feature dimension per entry of 2. To effectively train the specular model, we apply L1 regularization on the specular color with a weight set to 1e-5. In Section 3.2, for extracting the clean mesh, we employ the visibility and free-space culling techniques introduced by BakedSDF [3]. After Section 3.3, to export the texture, we first apply xatlas [384] to obtain per-vertex UV mapping. Subsequently, we can calculate diffuse color and specular features using f_{color} after the procedures outlined in Section 3.3. All these color

maps and features are 3-channel .png files, which are lightweight compared to the large feature maps used in neural texture. For real-time rendering, we follow the same method presented by previous research [42,43,238].The MLP for converting specular features to color is stored in a .json file. In rendering, a fragment shader incorporates this file to apply view-dependent effects. For experiments on the mip-NeRF 360 dataset [88], we use three outdoor scenes—bicycle, garden, and stump—and four indoor scenes—room, counter, kitchen, and bonsai. These scenes include highly reflective surfaces and thin structures, posing challenges for mesh-based rendering. Additionally, we include playroom and drjohnson from the deep blending dataset [6], as well as barn and courthouse from the Tanks & Temples dataset [100]. We implement all neural implicit representations, including Geometry Reconstruction, Appearance Modeling, and Optimization components, using PyTorch [385].The mesh decimation part is implemented in C++. Furthermore, all experiments are conducted on a V100 GPU.

C.2 Additional Explanations

To initialize the appearance in Section 3.1, g_{color} incorporates not only the position and viewing direction but also utilizes the predicted surface normal and a 256-dimensional geometric feature as input. This 256-dimensional geometric feature and surface normal are produced by the geometry MLP f_{sdf}. When we modify the vertex position, and deformed vertices end up in empty space, the geometry MLP becomes incapable of computing a useful geometric feature, thereby impeding the rendering of accurate colors. Embedding the neural feature for each vertex is an option [214]; however, it requires significant GPU memory, particularly for large unbounded scenes. Thus, we only use the geometry MLP for extracting mesh.

For mesh decimation in Section 3.2, we group the vertices into 8 ($2 \times 2 \times 2$) clusters and

calculate the quadric error for each cluster. Subsequently, we employ geometry-aware QSlim for each cluster to perform edge decimation. If the quadric error for each cluster exceeds a certain threshold, we halt the execution of QSlim.

C.3 Additional Qualitative Comparisons

In the manuscript, we present mesh reconstruction results exclusively for Bicycle, Garden, Counter, and Room (refer to Fig. 4 in the manuscript) due to the limited space. We show results for the remaining scenes: Stump, Kitchen, and Bonsai. We show both the mesh reconstruction and rendering results for these scenes, as shown in Fig. C.1 and Fig. C.2. Furthermore, we visualize both the mesh reconstruction and rendering results for Playroom and Dr. Johnson from the Tanks & Temples dataset, and for Barn and Courthouse from the Deep Blending dataset, as illustrated in Fig. C.3 and Fig. C.4.

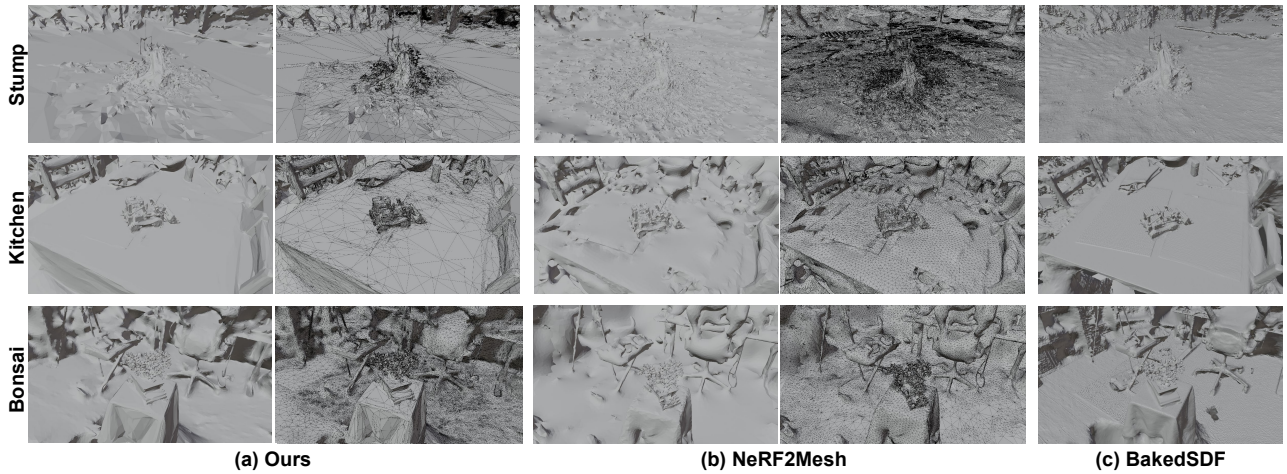


Figure C.1: Qualitative comparison between (a) Ours, (b) NeRF2Mesh [42], and (c) BakedSDF [3]. Our method and NeRF2Mesh show shading mesh and wireframe mesh extracted without texture, respectively.



Figure C.2: Qualitative comparison between (a) Ours, (b) NeRF2Mesh [42], (c) BakedSDF* [3], and (d) MobileNeRF [43]. The visual results of BakedSDF* generated in its first stage through volumetric rendering.

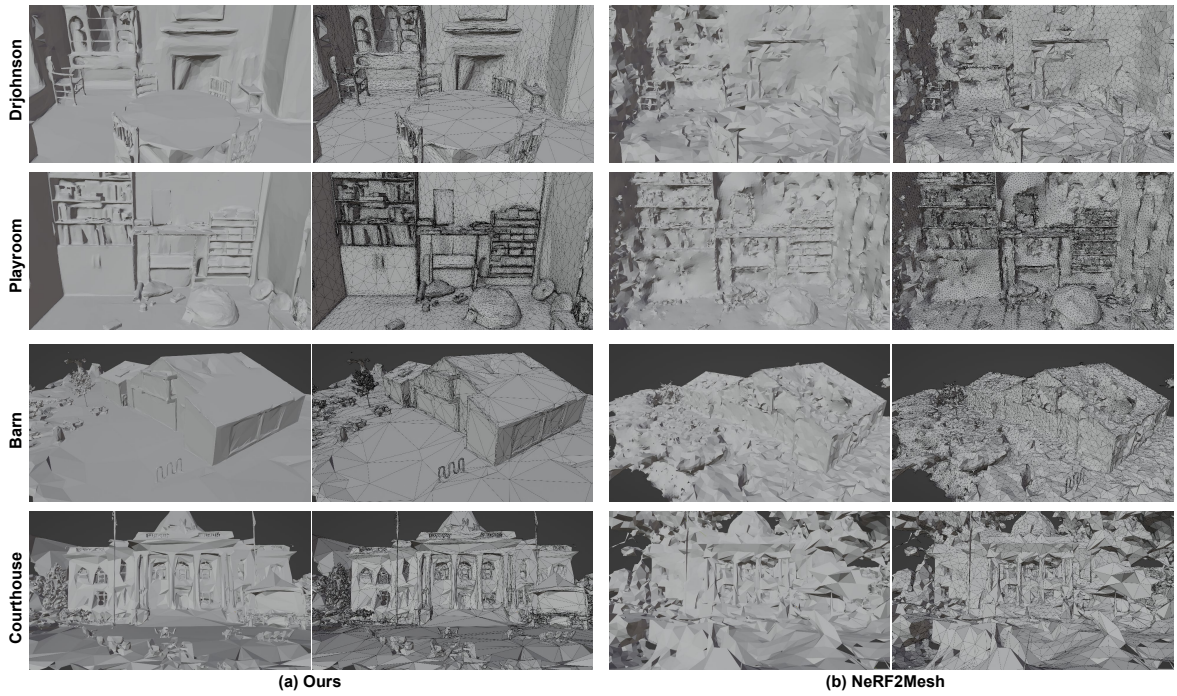


Figure C.3: Qualitative comparison between (a) Ours and (b) NeRF2Mesh [42]. Our method and NeRF2Mesh show shading mesh and wireframe mesh extracted without texture, respectively.



Figure C.4: Qualitative comparison between (a) Ours, (b) NeRF2Mesh [42], and (c) MobileNeRF [43].

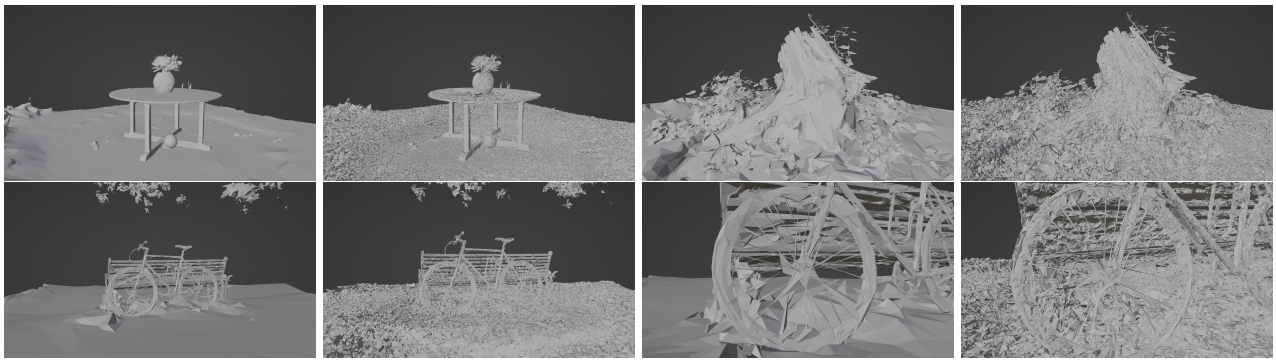


Figure C.5: Comparison of mesh quality when optimized with decimation (columns 1, 3) vs. without decimation (columns 2, 4).

Chapter D: Appendix for Neural Digital Twins for UAVs using Gaussian Splatting

D.1 Details of Human Recognition Experiments

We select the Okutama-Action dataset [13], which includes both action labels and bounding box annotations. This dataset is notably challenging and unique, as it provides multi-action labels. To the best of our knowledge, it is the only dataset offering multi-action detection labels for evaluation. While UAVHuman [313] is more akin to a single-action dataset captured at low altitude, Okutama-Action dataset presents a greater challenge due to its multi-action nature. We follow the official paper [13], which includes a test set consisting of 10 video sequences, as detailed below:

1. Morning: 1.1.8, 1.1.9, 2.1.8, 2.1.9, totaling 434 images with detection, tracking IDs, and action labels.
2. Noon: 1.2.1, 1.2.3, 1.2.10, 2.2.1, 2.2.3, 2.2.10, totaling 664 images with detection, tracking IDs, and action labels.

For action recognition, we focus on five non-interaction labels: running, walking, lying, sitting, and standing.

D.1.1 Details of Person Detection Experiments

D.1.2 Details of Action Recognition Experiments

For action recognition, we use R3D [15], a 3D convolutional neural network adapted from the widely used 2D ResNet architecture. R3D is designed to capture spatiotemporal features from video

Table D.1: **Details of Train Dataset.** "K" denotes the 10^3 unit.

Name	time	# images
R1.2.2	Noon	1K
R2.2.2	Noon	1.4K
R1.1.1	Morning	2.2K
R2.1.1	Morning	1.2K
Syn-N	Noon	3K
Syn-M	Morning	3K

Table D.2: **AP Comparison on the Okutama-Action [13] of YOLOv8 family.**

Train	Test	mAP_{50}			$mAP_{50:95}$		
		yolov8n	yolov8s	yolov8m	yolov8n	yolov8s	yolov8m
Syn-N	Noon	40.8	39.7	44.2	11.6	10.1	11.6
R1.2.2	Noon	46.8	52.5	58.0	14.0	**17.0**	18.9
R1.2.2 + Syn-N	Noon	**50.2**	**53.4**	**61.5**	**14.6**	15.0	**20.3**
R2.2.2	Noon	40.7	43.1	51.6	12.5	14.1	16.4
R2.2.2 + Syn-N	Noon	**46.3**	**47.9**	**56.1**	**13.7**	**16.5**	**17.4**
Syn-M	Morning	19.7	16.8	31.3	4.5	3.7	21.9
R1.1.1	Morning	40.6	44.9	60.0	11.8	13.4	19.1
R1.1.1 + Syn-M	Morning	**43.1**	**48.1**	**61.5**	**12.5**	**14.8**	**20.3**
R2.1.1	Morning	30.6	33.6	35.2	**7.5**	8.4	8.3
R2.1.1 + Syn-M	Morning	**31.7**	**44.6**	**47.6**	7.4	**12.0**	**12.5**

clips for action classification. To ensure a fair comparison, we utilize ground-truth tracking results and bounding boxes to extract person-centric video clips. These cropped clips are then fed into R3D for action recognition. We freeze the backbone trained on Kinetic dataset and fine-tune only the final block for this task. This takes 16-frame clips as input and predict the classification results.

D.2 Multi-view Regularization

For multi-view regularization, we follow PGSR approach [12] which shows the state-of-the-art surface reconstruction performance on benchmark datasets. A patch-based normalized cross-correlation (NCC) loss is applied between two gray renders I and $\hat{I}$ to force the multi-view photometric consistency. Also, the multi-view geometric consistency regularization L_{mvegeo} is defined using the forward and backward projection error ϕ of pixel p computed by using the reference frame and neighboring frame.

$$
\begin{aligned}
L_{mvgeo} &= \sum_{p \in P} \phi(p) \\
L_{mvpho} &= \sum_{\mathrm{p} \in P_r} \sum_{p \in \mathrm{p}} (1 - NCC(\hat{I}(\mathcal{H}p), I(p)))
\end{aligned}
\tag{D.1}
$$

where P_r is the set of all patches obtained from the rendered image I and $\mathcal{H}$ is the homography matrix between the reference and neighboring frames. The single-view normal loss is defined as:

$$L_{svgeo} = \sum_{p \in C} ||N_d(p) - N(p)|| \tag{D.2}$$

where C represents the rendered image and N_d denotes the normal of the local plane. Here, four neighboring pixels are projected into 3D points, and the normal is computed using these four 3D points. The multi-view regularization loss is defined as:

$$L_{mvreg} = \lambda_a L_{mvgeo} + \lambda_b L_{mvpho} + \lambda_c L_{svgeo} \tag{D.3}$$

where λ_a, λ_b, λ_c are set to 0.01, 0.2, and 0.05.

D.3 Implementation Details

For building digital twin part, we implement MsGS based on Gaussian splatting [10]. We use 32 dimension vector for per-sequence embedding. Thus, for 25 video sequences, we define 25 vectors for each embedding. For the appearance MLP, we employ two hidden layers of size 128 with ReLU activation. The model and Gaussian splats are optimized using the Adam optimizer [137]. In mask refinement module, ρ_1 and ρ_2 are set to filter out 70 % and 80 % of the masks, respectively. The appearance threshold ρ_{pho} is defined based on the statistical properties of the photometric loss L_{pho} values. Specifically, the mean and standard deviation of the photometric loss are computed. Instead of using a threshold set at the mean plus the standard deviation (*mean* + *std*), the threshold is determined as: ρ_{pho} = *mean* - $\frac{std}{2}$. This formulation sets the threshold slightly below the mean, thereby reducing the

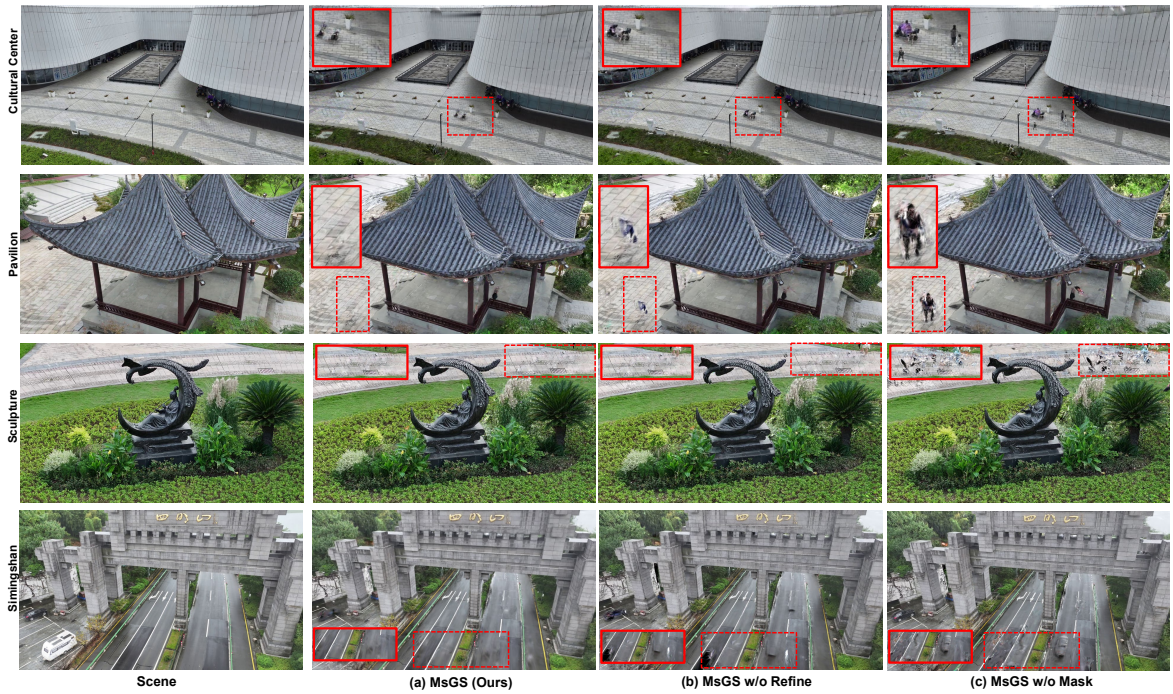


Figure D.1: **Qualitative Comparison of MsGS (Ours), "MsGS w/o Refine", "MsGS w/o Mask"** on the DroneSplat dataset. Without masks, the 4th columns shows that significant artifacts appear in regions occupied by transient and dynamic objects. When using masks refined by the mask refinement module, our method (2nd column) effectively reduces artifacts compared to "MsGS w/o Refine" (3rd column), which relies solely on SAM masks.

influence of higher variance. As a result, values below this threshold may be considered less significant or filtered out. We run all experiments for 30k iterations. We set the loss weights to λ_{pho}=0.2 and λ_s=100.

In neural data generation, we define the human scale based on the reconstructed background mesh to control the size of synthetic humans. Specifically, we use a scale of 0.135 for the Noon scene and 0.195 for the Morning scene in the Okutama dataset. For rendering foreground humans, we utilize the Synbody rendering toolbox. We usually randomly position 10 $\sim$ 15 synthetic humans for data generation. We manually select a training image that represents the environmental lighting and input it into DiffusionLight [339] to generate environmental maps. We follow the official YOLOv8 [14] fine-tuning protocols for training the human detection model.

Table D.3: **Quantitative Comparison on NeRF-OSR**

Method	PSNR ↑ / SSIM ↑ / LPIPS ↓			
	stjohann	lwp	st	Mean
3DGS [10]	13.14/ 0.571 / 0.372	14.41 / 0.515 / 0.406	13.47 / 0.512 / 0.436	13.67 / 0.531 / 0.404
GOF [52]	11.67 / 0.542 / 0.411	13.90 / 0.527 / 0.420	13.18 / 0.528 / 0.428	12.91 / 0.528 / 0.428
2DGS [11]	10.77 / 0.521 / 0.415	14.31 / 0.530 / 0.411	14.31 / 0.569 / 0.420	13.13 / 0.54 / 0.415
PGSR [12]	11.52 / 0.544 / 0.407	14.26 / 0.494 / 0.420	14.17 / 0.504 / 0.407	13.31 / 0.514 / 0.411
MsGS (Ours)	**19.05 / 0.766 / 0.222**	**16.92 / 0.638 / 0.324**	**18.11 / 0.650 / 0.311**	**18.03 / 0.684 / 0.285**

D.4 Mask Refinement

For clarity, we provide a high-level overview of our mask refinement module using pseudocode in Algorithm 1. Figure D.1 illustrates the effectiveness of our method compared to using only SAM masks and our approach without masks.

Algorithm 1 Filtering and Processing SAM Masks

Require: Area of each entity mask $A(S(e_{id}))$, Area of original SAM mask $A(\overline{M})$, Thresholds ρ_1, ρ_2, Dilation function $Dilate(\cdot)$, Error map per mask $R(e_{id})$, photometric loss threshold ρ_{pho}
Ensure: Processed SAM mask $\overline{M}$

Thresholding: Select entity masks $S'(e_{id})$ where $A(S(e_{id})) < \rho_1$ and $A(\overline{M}) < \rho_2$
Dilation: Apply dilation to SAM mask: $\overline{M} \leftarrow Dilate(\overline{M})$
Mask Selection: Gather all entity masks that overlap with $\overline{M}$: $M' \leftarrow Dilate(\overline{M}) \cup \hat{M}$
for each entity mask $\hat{M}(e_{id}) \in M'$ **do**
    if $R(e_{id}) > \rho_{pho}$ **then**
        Add $\hat{M}(e_{id})$ to $\overline{M}$
    else
        Remove $\hat{M}(e_{id})$ from $\overline{M}$
    end if
end for

Table D.4: **Quantitative Comparison on DroneSplat Dataset.** In the manuscript, we present the mean values across six scenes. This table provides detailed results for each individual scene.

Method	Mesh	Mask	PSNR ↑ / SSIM ↑ / LPIPS ↓						
			Cultural Center	Intersection	Pavilion	Sculpture	Simingshan	TangTian	Mean
3DGS [10]	X	O	22.58 / 0.792 / 0.179	18.17 / 0.586 / 0.398	17.70 / 0.504 / 0.297	18.67 / 0.514 / 0.301	22.61 / 0.798 / 0.0.166	17.35 / 0.493 / 0.432	19.51 / 0.614 / 0.295
2DGS [11]	O	O	21.41 / 0.712 / 0.165	18.57 / 0.633 / 0.379	17.93 / 0.535 / 0.378	18.90 / 0.561 / 0.255	22.76 / 0.898 / 0.170	17.13 / 0.500 / 0.431	19.45 / 0.622 / 0.282
PGSR [12]	O	O	22.37 / 0.737 / 0.195	17.18 / 0.590 / 0.387	17.51 / 0.507 / 0.281	18.57 / 0.561 / 0.252	22.82 / 0.898 / 0.158	16.16 / 0.487 / 0.435	19.10 / 0.630 / 0.284
MsGS (Ours)	O	O	22.75 / 0.801 / 0.185	18.26 / 0.62 / 0.364	17.65 / 0.515 / 0.273	19.46 / 0.596 / 0.218	22.63 / 0.812 / 0.154	17.41 / 0.512 / 0.413	**19.70 / 0.643 / 0.267**
MsGS w/o Refine	O	O	22.57 / 0.737 / 0.194	17.56 / 0.598 / 0.391	17.44 / 0.508 / 0.282	19.40 / 0.593 / 0.221	22.95 / 0.812 / 0.155	17.17 / 0.503 / 0.423	19.51 / 0.625 / 0.267

Table D.5: **Quantitative Comparison on Okutama Dataset.** Drone1-Noon comprises six video sequences, each with distinct characteristics. In the manuscript, we present the mean values across these six scenes, while this table provides detailed results for each individual scene.

Method	Mesh	Mask	Drone1-Noon (PSNR ↑ / SSIM ↑ / LPIPS ↓)						
			1.2.2	1.2.4	1.2.6	1.2.8	1.2.9	1.2.11	Mean
3DGS [10]	X	O	32.35 / 0.945 / 0.171	35.80 / 0.961 / 0.143	24.87 / 0.826 / 0.285	28.50 / 0.916 / 0.166	31.76 / 0.895 / 0.189	32.39 / 0.918 / 0.158	31.48 / 0.917 / 0.177
GOF [52]	O	O	31.29 / 0.941 / 0.170	34.09 / 0.957 / 0.151	25.07 / 0.845 / 0.274	26.97 / 0.886 / 0.214	29.72 / 0.914 / 0.185	30.01 / 0.912 / 0.174	29.95 / 0.914 / 0.189
2DGS [11]	O	X	30.82 / 0.934 / 0.216	34.28 / 0.951 / 0.194	24.73 / 0.868 / 0.257	29.32 / 0.930 / 0.159	33.59 / 0.921 / 0.182	31.63 / 0.907 / 0.202	31.22 / 0.923 / 0.197
PGSR [12]	O	X	24.07 / 0.938 / 0.141	33.27 / 0.963 / 0.113	24.70 / 0.889 / 0.202	29.16 / 0.943 / 0.114	30.30 / 0.936 / 0.129	29.76 / 0.921 / 0.137	29.17 / 0.936 / 0.137
PGSR [12]	O	O	25.19 / 0.943 / 0.140	33.49 / 0.963 / 0.113	24.49 / 0.891 / 0.200	29.04 / 0.942 / 0.116	30.76 / 0.937 / 0.129	30.14 / 0.920 / 0.137	29.47 / 0.937 / 0.137
MsGS (Ours)	O	O	33.17 / 0.946 / 0.165	35.49 / 0.962 / 0.153	30.31 / 0.904 / 0.167	31.56 / 0.944 / 0.120	34.13 / 0.926 / 0.157	32.20 / 0.914 / 0.161	33.08 / 0.937 / 0.154

Table D.6: **Quantitative Comparison on Okutama Dataset.** Drone1-Morning comprises seven video sequences, each with distinct characteristics. In the manuscript, we present the mean values across these six scenes, while this table provides detailed results for each individual scene.

Method	Mesh	Mask	Drone1-Morning (PSNR ↑ / SSIM ↑ / LPIPS ↓)							
			1.1.1	1.1.2	1.1.3	1.1.4	1.1.7	1.1.10	1.1.10	Mean
3DGS [10]	X	O	28.19/ 0.926 / 0.202	28.70 / 0.912 / 0.223	27.13 / 0.917 / 0.244	23.35 / 0.890 / 0.220	25.35 / 0.881 / 0.254	22.83 / 0.864 / 0.220	29.07 / 0.915 / 0.155	26.33 / 0.901 / 0.222
GOF [52]	O	O	29.45 / 0.937 / 0.159	28.29 / 0.914 / 0.183	30.73 / 0.928 / 0.192	30.96 / 0.940 / 0.087	29.43 / 0.910 / 0.176	22.43 / 0.853 / 0.233	29.69 / 0.914 / 0.159	29.04 / 0.918 / 0.164
2DGS [11]	O	X	28.23 / 0.924 / 0.232	28.24 / 0.899 / 0.258	27.06 / 0.914 / 0.279	27.55 / 0.898 / 0.204	27.86 / 0.881 / 0.239	19.57 / 0.865 / 0.247	27.89 / 0.916 / 0.164	27.09 / 0.899 / 0.236
PGSR [12]	O	X	28.49 / 0.934 / 0.155	26.68 / 0.914 / 0.176	25.75 / 0.922 / 0.232	23.64 / 0.897 / 0.185	26.37 / 0.896 / 0.206	18.18 / 0.841 / 0.221	27.59 / 0.911 / 0.149	25.57 / 0.906 / 0.188
PGSR [12]	O	O	28.77 / 0.942 / 0.135	27.79/ 0.919 / 0.147	29.54 / 0.934 / 0.160	30.04 / 0.939 / 0.079	28.03 / 0.914 / 0.151	18.21 / 0.842 / 0.223	26.93 / 0.909 / 0.154	27.71 / 0.920 / 0.142
MsGS (Ours)	O	O	31.30 / 0.940 / 0.145	30.10 / 0.925 / 0.176	29.90 / 0.933 / 0.219	31.12 / 0.918 / 0.150	32.12 / 0.912 / 0.180	30.43 / 0.919 / 0.146	33.35 / 0.931 / 0.124	31.07 / 0.925 / 0.166

Table D.7: **Quantitative Comparison on Okutama Dataset.** "-" denotes a missing implementation. Drone2-Noon comprises five video sequences, each with distinct characteristics. In the manuscript, we present the mean values across these six scenes, while this table provides detailed results for each individual scene.

Method	Mesh	Mask	Drone2-Noon (PSNR ↑ / SSIM ↑ / LPIPS ↓)					
			2.2.2	2.2.4	2.2.8	2.2.9	2.2.11	Mean
3DGS [10]	X	O	32.76 / 0.935 / 0.199	31.00 / 0.934 / 0.188	38.71 / 0.962 / 0.173	33.95 / 0.943 / 0.133	34.05 / 0.928 / 0.177	34.10 / 0.943 / 0.175
GOF [52]	O	O	29.97 / 0.929 / 0.209	33.60 / 0.963 / 0.144	32.99 / 0.934 / 0.254	31.63 / 0.928 / 0.166	30.44 / 0.919 / 0.203	32.09 / 0.938 / 0.193
2DGS [11]	O	X	30.13 / 0.923 / 0.241	31.08 / 0.941 / 0.199	36.75 / 0.952 / 0.228	31.56 / 0.933 / 0.167	32.12 / 0.912 / 0.227	32.49 / 0.936 / 0.211
PGSR [12]	O	X	31.31 / 0.943 / 0.172	30.04 / 0.952 / 0.141	34.75 / 0.968 / 0.122	32.78 / 0.944 / 0.131	29.37 / 0.924 / 0.177	31.94 / 0.950 / 0.143
PGSR [12]	O	O	31.96 / 0.943 / 0.174	29.19 / 0.940 / 0.141	34.77 / 0.966 / 0.122	32.67 / 0.943 / 0.131	29.70 / 0.924 / 0.177	31.84 / 0.946 / 0.152
MsGS (Ours)	O	O	31.54 / 0.935 / 0.196	31.89 / 0.945 / 0.167	37.19 / 0.959 / 0.175	33.69 / 0.941 / 0.123	33.04 / 0.921 / 0.184	33.59 / 0.944 / 0.188

Table D.8: **Quantitative Comparison on Okutama Dataset.** Drone2-Morning comprises seven video sequences, each with distinct characteristics. In the manuscript, we present the mean values across these six scenes, while this table provides detailed results for each individual scene.

Method	Mesh	Mask	Drone2-Morning (PSNR ↑ / SSIM ↑ / LPIPS ↓)							
			2.1.1	2.1.2	2.1.3	2.1.4	2.1.6	2.1.7	2.1.10	Mean
3DGS [10]	X	O	29.69 / 0.915 / 0.164	28.92 / 0.940 / 0.148	30.06 / 0.920 / 0.211	27.82 / 0.881 / 0.233	31.18 / 0.923 / 0.141	19.73 / 0.804 / 0.377	25.61/ 0.888 / 0.216	27.32 / 0.891 / 0.216
GOF [52]	O	O	29.47 / 0.907 / 0.177	25.48 / 0.912 / 0.229	26.66 / 0.911 / 0.218	24.13 / 0.876 / 0.160	30.00 / 0.914 / 0.226	24.32 / 0.874 / 0.222	25.30 / 0.884 / 0.00	26.43 / 0.896 / 0.212
2DGS [11]	O	X	29.97 / 0.910 / 0.190	28.07 / 0.919 / 0.241	31.12 / 0.925 / 0.214	25.83 / 0.878 / 0.260	30.41 / 0.918 / 0.163	27.51 / 0.885 / 0.247	24.56 / 0.877 / 0.250	28.33 / 0.901 / 0.222
PGSR [12]	O	X	28.23 / 0.917 / 0.134	23.77 / 0.938 / 0.118	27.80 / 0.932 / 0.144	24.07 / 0.875 / 0.233	30.20 / 0.924 / 0.131	24.25 / 0.888 / 0.185	23.05 / 0.879 / 0.198	26.11 / 0.907 / 0.166
PGSR [12]	O	O	28.41 / 0.918 / 0.132	24.09 / 0.940 / 0.115	28.10 / 0.932 / 0.141	24.01 / 0.874 / 0.237	30.44 / 0.924 / 0.131	28.16 / 0.909 / 0.132	23.17 / 0.878 / 0.199	26.96 / 0.911 / 0.156
MsGS (Ours)	O	O	30.76 / 0.924 / 0.142	32.69 / 0.950 / 0.114	32.47 / 0.939 / 0.159	29.42 / 0.897 / 0.219	30.63 / 0.925 / 0.134	28.54 / 0.910 / 0.169	25.76 / 0.895 / 0.173	29.89 / 0.918 / 0.161

Chapter E: Appendix for Dynamic Neural Rendering of Human-Centric UAV Imagery usingn Gaussian Splatting

E.1 Dataset & Training Details

E.1.1 Dataset Details

VisDrone: The VisDrone dataset [53] is a standard benchmark for object detection and tracking in the UAV research community. It was captured at altitudes of 40 to 50 meters in China under varying weather and illumination conditions. This dataset includes many densely packed small objects in UAV images. We selected four different scenes that primarily consist of pedestrians and complex background environments.

Okutama-Action: The Okutama-Action dataset [13] contains video sequences of 12 action categories, captured using cameras mounted on two flying UAVs at altitudes ranging from 10 to 45 meters. We chose four scenes that exhibit significant camera variation and distinct viewing angles.

Manipal-UAV: The Manipal-UAV dataset [54] is designed for person detection in UAV images. It was acquired in India using two UAVs flying at altitudes between 10 and 50 meters. Unlike the previous datasets, it explicitly provides altitude information. We selected four scenes captured at 40 and 50 meters, representing the highest-altitude cases.

We select four scenes from each dataset, with most scenes comprising approximately 200 images in total, including both training and test sets.

1. VisDrone: uav0013_00000, uav0079_00480, uav0084_00000, uav0099_02109

2. Manipal-UAV: 40_GND_P1, 40_VGT_P1, 50_GND_P1, 50_RD_P1

3. Okutama-Action: 1_2_2, 1_2_6, 1_2_8, 1_2_9

In the VisDrone dataset, humans have an average size of approximately 29×78 pixels, while in the Manipal-UAV dataset, the average size is around 7×17 pixels. In the Okutama dataset, humans typically appear at an average size of 17×44 pixels.



Scene SAM2 [362] ViTPose [377] HMR2.0 [361]

Figure E.1: **Visual Examples of Initial Estimates.** Each initial estimate is used as input for the subsequent human-scene reconstruction process.

E.1.2 Training Details

Initialization Figure E.1 presents visual examples of human masks generated by SAM2 [362], 2D keypoint poses estimated by VitPose [377], and human meshes reconstructed by HMR2.0 [361]. We observe that human bounding boxes are generally more reliable than the reconstructed human mesh parameters. Therefore, we set η_{box} to 1.2 to filter out erroneous projected human meshes in image space that appear excessively large compared to their corresponding bounding boxes.

Joint Human-Scene Reconstruction To reconstruct the background mesh, we first filter point maps based on confidence scores. The confidence threshold η_{conf} is determined according to the UAV's altitude. Specifically, we set η_{conf} to 5 for the Manipal-UAV dataset, 20 for Okutama-Action, and 40 for the VisDrone dataset. For high-altitude captures (e.g., 40–50 meters), a lower confidence threshold is used since most point maps correspond to static background. In contrast, for low-altitude scenarios, a higher confidence threshold is applied to exclude point maps from dynamic regions. For scale op-

timization, we first initialize the scale σ to 40. We then select approximately 5–10 human instances and sample 5 frames at intervals of 5 frames. Keypoints corresponding to the main body are used for optimization We set the learning rate α to 0.1 and perform T_{opt} to 30 optimization steps. We note that the root translation is typically associated with the pelvis joint. To convert the ground contact point to the root translation, we apply a vertical offset of 1 unit.

3D Gaussian Sceen Modeling To optimize all Gaussians, we use various loss functions and regularizations in Eq. (5) in the main paper. The opacity regularization L_o is defined as

$$L_o = \sum -o_{all} log(o_{all}) - \sum m_{sky} log(1 - o_{all}) \tag{E.1}$$

where o_{all} is the rendered opacity map from all Gaussian attributes and m_{sky} is the sky mask. Most of the UAV scenes selected do not include sky regions. Only two scenes, uav0013_00000 and uav0099_02109 from the VisDrone dataset, contain visible sky areas. The remaining ten scenes exclude sky regions from consideration.

Following GART [379] and OmniRe [350], L_{smpl} includes KNN regularization L_{knn} and temporal smoothness regularization L_{smooth}. We apply KNN regularization for all Gaussian attributes to constrain the transformation and appearance of neighbors in order to reduce the artifacts.

$$L_{knn} = \sum_{(\mu,r,s,c,o)\in g_h} \lambda_{knn} STD_{i\in KNN(\mu_i)}(\mu_i, r_i, s_i, c_i, o_i), \tag{E.2}$$

where STD denotes the standard deviation. Furthermore, we apply temporal smoothness regularization to ensure that human body poses θ_t vary smoothly over time by encouraging consistency between

the current pose and its adjacent frames,

$$L_{smooth} = 0.5\lambda_{smooth}||(\theta_t - \theta_{t-1}) + (\theta_t - \theta_{t+1})||. \tag{E.3}$$

We set λ_{pho} to 0.2, λ_o to 0.05, λ_{knn} to 0.01, λ_{smooth} to 0.001. We optimize all Gaussian attributes and all SMPL parameters using Adam optimizer for 30,000 iterations. For optimizing background Gaussian attributes, we set the learning rates for rotations, scales, zeroth-order spherical harmonics, the remaining spherical harmonics to 0.001, 0.005, 0.0025, and 0.000125 respectively. To optimize the human Gaussian attributes, we set the learning rates for rotations, scales, zeroth-order spherical harmonics, and higher-order spherical harmonics to 0.0016, 0.005, 0.0025, and 0.000125, respectively. For human pose parameters, the learning rates for rotation and translation are set to 0.00001 and 0.0005, respectively. For densification for Gaussian attributes, we utilize the strategy of AbsGS [386]. The densification threshold for the position gradient is set to 0.0003.

E.2 Baseline Details

Description of the baseline approaches we used for comparison.

TK-Planes [104] extends the K-Planes Neural Radiance Field (NeRF) by introducing a set of tiered feature vectors. These vectors are designed to effectively capture both conceptual information about the scene and fine-grained details. An image decoder then transforms the output feature maps into RGB images. By operating on all feature vectors, the decoder can learn both static and dynamic objects, making the approach well-suited for UAV scenes with subtle dynamic variations. The original authors provide experimental results across all datasets.

4DGS [105] utilizes 4D K-Planes and a MLP to encode voxel features from 3D Gaussians, aggregating

nearby spatial and temporal Gaussians that share similar characteristics. It then employs a multi-head Gaussian deformation decoder to decode the feature and predict changes in position, rotation, and scale. We run the official code from the public GitHub repository using the provided hyperparameter settings.

DfGS [106] introduces a deformation field to 3D-GS, extending it to a canonical space. A deformation MLP network is used to learn the offsets from the canonical 3D-GS to the final 3D shape. This network takes timestamps and 3D Gaussian positions as input and predicts the deformation in position, rotation, and scale of 3D Gaussians across frames. We run the official code from the public GitHub repository using the provided hyperparameter settings.

E.3 Experimental Results

E.3.1 Zoom-in Human Regions

Fig. E.2 shows example visualizations of human regions, comparing our method with the ground truth images. In the VisDrone dataset, human figures are approximately 30×79 pixels in size, while in the Okutama-Action dataset, they are around 17×44 pixels.

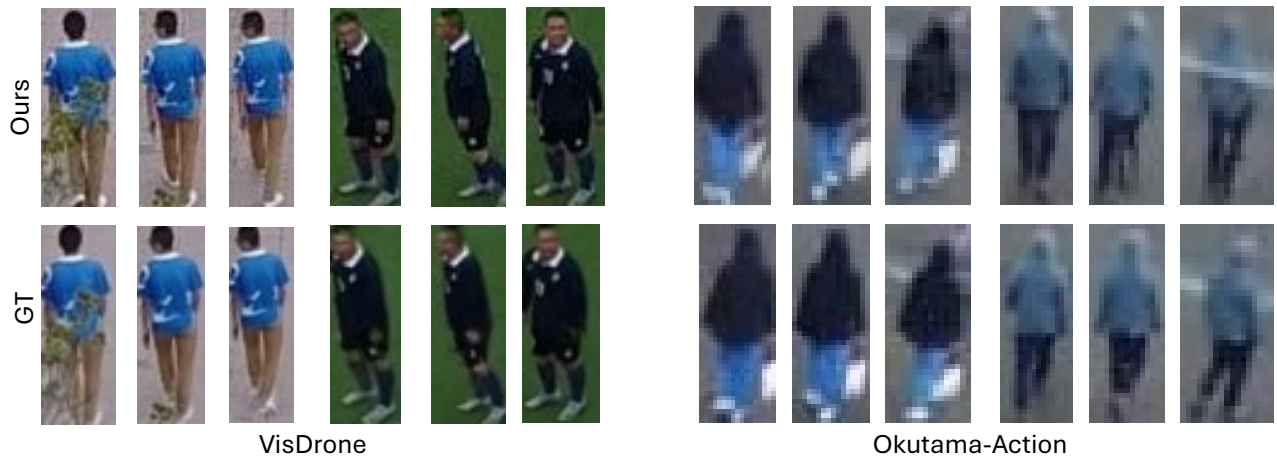


Figure E.2: **Visual Examples of Zoom-in Human Regions** on VisDrone and Okutama-Action Datasets

Bibliography

[1] Rasmus Jensen, Anders Dahl, George Vogiatzis, Engin Tola, and Henrik Aanæs. Large scale multi-view stereopsis evaluation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 406–413, 2014.

[2] Jaehoon Choi, Rajvi Shah, Qinbo Li, Yipeng Wang, Ayush Saraf, Changil Kim, Jia-Bin Huang, Dinesh Manocha, Suhib Alsisan, and Johannes Kopf. Ltm: Lightweight textured mesh extraction and refinement of large unbounded scenes for efficient storage and real-time rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5053–5063, 2024.

[3] Lior Yariv, Peter Hedman, Christian Reiser, Dor Verbin, Pratul P. Srinivasan, Richard Szeliski, Jonathan T. Barron, and Ben Mildenhall. Bakedsdf: Meshing neural sdfs for real-time view synthesis. *arXiv*, 2023.

[4] Joanna Waczyńska, Piotr Borycki, Sławomir Tadeja, Jacek Tabor, and Przemysław Spurek. Games: Mesh-based adapting and modification of gaussian splatting. *arXiv preprint arXiv:2402.01459*, 2024.

[5] Antoine Guédon and Vincent Lepetit. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5354–5363, 2024.

[6] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics (ToG)*, 37(6):1–15, 2018.

[7] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *arXiv preprint arXiv:1709.06158*, 2017.

[8] Iro Armeni, Sasha Sax, Amir R Zamir, and Silvio Savarese. Joint 2d-3d-semantic data for indoor scene understanding. *arXiv preprint arXiv:1702.01105*, 2017.

[9] Michael Waechter, Nils Moehrle, and Michael Goesele. Let there be color! large-scale texturing of 3d reconstructions. In *European conference on computer vision*, pages 836–850. Springer, 2014.

[10] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), 2023.

[11] Binbin Huang, Zehao Yu, Anpei Chen, Andreas Geiger, and Shenghua Gao. 2d gaussian splatting for geometrically accurate radiance fields. *arXiv preprint arXiv:2403.17888*, 2024.

[12] Danpeng Chen, Hai Li, Weicai Ye, Yifan Wang, Weijian Xie, Shangjin Zhai, Nan Wang, Haomin Liu, Hujun Bao, and Guofeng Zhang. Pgsr: Planar-based gaussian splatting for efficient and high-fidelity surface reconstruction. *arXiv preprint arXiv:2406.06521*, 2024.

[13] Mohammadamin Barekatain, Miquel Martí, Hsueh-Fu Shih, Samuel Murray, Kotaro Nakayama, Yutaka Matsuo, and Helmut Prendinger. Okutama-action: An aerial view video dataset for concurrent human action detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 28–35, 2017.

[14] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics yolov8. 2023, 2023.

[15] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun, and Manohar Paluri. A closer look at spatiotemporal convolutions for action recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 6450–6459, 2018.

[16] Ze Liu, Jia Ning, Yue Cao, Yixuan Wei, Zheng Zhang, Stephen Lin, and Han Hu. Video swin transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3202–3211, 2022.

[17] Xun Sun, Yuanfan Xie, Pei Luo, and Liang Wang. A dataset for benchmarking image-based localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7436–7444, 2017.

[18] Noé Pion, Martin Humenberger, Gabriela Csurka, Yohann Cabon, and Torsten Sattler. Benchmarking image retrieval for visual localization. In *2020 International Conference on 3D Vision (3DV)*, pages 483–494. IEEE, 2020.

[19] Jaehoon Choi, Dongki Jung, Yonghan Lee, Deokhwa Kim, Dinesh Manocha, and Donghwan Lee. Selfdeco: Self-supervised monocular depth completion in challenging indoor environments. In *Proc. of the IEEE International Conference on Robotics and Automation*, Xi'an, China, 2021.

[20] Dongki Jung, Jaehoon Choi, Yonghan Lee, Deokhwa Kim, Changick Kim, Dinesh Manocha, and Donghwan Lee. Dnd: Dense depth estimation in crowded dynamic indoor scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12797–12807, 2021.

[21] Jaehoon Choi, Dongki Jung, Taejae Lee, Sangwook Kim, Youngdong Jung, Dinesh Manocha, and Donghwan Lee. Tmo: Textured mesh acquisition of objects with a mobile device by using differentiable rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16674–16684, 2023.

[22] Jaehoon Choi, Yonghan Lee, Hyungtae Lee, Heesung Kwon, and Dinesh Manocha. Meshgs: Adaptive mesh-aligned gaussian splatting for high-quality rendering. *arXiv preprint arXiv:2410.08941*, 2024.

[23] Dongki Jung, Jaehoon Choi, Yonghan Lee, and Dinesh Manocha. Im360: Textured mesh reconstruction for large-scale indoor mapping with 360 cameras. *arXiv preprint arXiv:2502.12545*, 2025.

[24] Jaehoon Choi, Dongki Jung, Yonghan Lee, Sungmin Eum, Dinesh Manocha, and Heesung Kwon. Uavtwin: Neural digital twins for uavs using gaussian splatting. *arXiv preprint arXiv:2504.02158*, 2025.

[25] Jaehoon Choi, Dongki Jung, Christopher Maxey, Yonghan Lee, Sungmin Eum, Dinesh Manocha, and Heesung Kwon. Uav4d: Dynamic neural rendering of human-centric uav imagery using gaussian splatting. *arXiv preprint arXiv:2506.05011*, 2025.

[26] Fangchang Ma, Guilherme Venturelli Cavalheiro, and Sertac Karaman. Self-supervised sparse-to-dense: Self-supervised depth completion from lidar and monocular camera. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3288–3295. IEEE, 2019.

[27] Abdelrahman Eldesokey, Michael Felsberg, and Fahad Shahbaz Khan. Confidence propagation through cnns for guided sparse depth regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[28] Xinjing Cheng, Peng Wang, and Ruigang Yang. Depth estimation via affinity learned with convolutional spatial propagation network. In *European Conference on Computer Vision*, pages 103–119. Springer, 2018.

[29] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4104–4113, 2016.

[30] Johannes L Schönberger, Enliang Zheng, Jan-Michael Frahm, and Marc Pollefeys. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision*, pages 501–518. Springer, 2016.

[31] Zehao Yu and Shenghua Gao. Fast-mvsnet: Sparse-to-dense multi-view stereo with learned propagation and gauss-newton refinement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1949–1958, 2020.

[32] Zhengqi Li, Tali Dekel, Forrester Cole, Richard Tucker, Noah Snavely, Ce Liu, and William T Freeman. Learning the depths of moving people by watching frozen people. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4521–4530, 2019.

[33] Jin Han Lee, Myung-Kyu Han, Dong Wook Ko, and Il Hong Suh. From big to small: Multi-scale local planar guidance for monocular depth estimation. *arXiv preprint arXiv:1907.10326*, 2019.

[34] Qingshan Xu and Wenbing Tao. Planar prior assisted patchmatch multi-view stereo. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 12516–12523, 2020.

[35] Lior Yariv, Jiatao Gu, Yoni Kasten, and Yaron Lipman. Volume rendering of neural implicit surfaces. *Advances in Neural Information Processing Systems*, 34:4805–4815, 2021.

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

[37] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, 32(3):1–13, 2013.

[38] Qian-Yi Zhou and Vladlen Koltun. Color map optimization for 3d reconstruction with consumer depth cameras. *ACM Transactions on Graphics (ToG)*, 33(4):1–10, 2014.

[39] Peng Wang, Lingjie Liu, Yuan Liu, Christian Theobalt, Taku Komura, and Wenping Wang. Neus: Learning neural implicit surfaces by volume rendering for multi-view reconstruction. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021.

[40] Lixiang Lin, Jianke Zhu, and Yisu Zhang. Multiview textured mesh recovery by differentiable rendering. *IEEE Transactions on Circuits and Systems for Video Technology*, pages 1–1, 2022.

[41] Jacob Munkberg, Jon Hasselgren, Tianchang Shen, Jun Gao, Wenzheng Chen, Alex Evans, Thomas Mueller, and Sanja Fidler. Extracting Triangular 3D Models, Materials, and Lighting From Images. *arXiv:2111.12503*, 2021.

[42] Jiaxiang Tang, Hang Zhou, Xiaokang Chen, Tianshu Hu, Errui Ding, Jingdong Wang, and Gang Zeng. Delicate textured mesh recovery from nerf via adaptive surface refinement. *arXiv preprint arXiv:2303.02091*, 2022.

[43] Zhiqin Chen, Thomas Funkhouser, Peter Hedman, and Andrea Tagliasacchi. Mobilenerf: Exploiting the polygon rasterization pipeline for efficient neural field rendering on mobile architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16569–16578, 2023.

[44] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5828–5839, 2017.

[45] Hakyeong Kim, Andreas Meuleman, Hyeonjoong Jang, James Tompkin, and Min H Kim. Omnisdf: Scene reconstruction using omnidirectional signed distance functions and adaptive binoctrees. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20227–20236, 2024.

[46] Antoine Guédon and Vincent Lepetit. Sugar: Surface-aligned gaussian splatting for efficient 3d mesh reconstruction and high-quality mesh rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5354–5363, 2024.

[47] Hanlin Chen, Fangyin Wei, Chen Li, Tianxin Huang, Yunsong Wang, and Gim Hee Lee. Vcr-gaus: View consistent depth-normal regularizer for gaussian surface reconstruction. *Advances in Neural Information Processing Systems*, 37:139725–139750, 2025.

[48] Haolin Xiong, Sairisheek Muttukuru, Rishi Upadhyay, Pradyumna Chari, and Achuta Kadambi. Sparsegs: Real-time 360° sparse view synthesis using gaussian splatting. *Arxiv*, 2023.

[49] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Zip-nerf: Anti-aliased grid-based neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 19697–19705, 2023.

[50] Tianhe Ren, Shilong Liu, Ailing Zeng, Jing Lin, Kunchang Li, He Cao, Jiayu Chen, Xinyu Huang, Yukang Chen, Feng Yan, et al. Grounded sam: Assembling open-world models for diverse visual tasks. *arXiv preprint arXiv:2401.14159*, 2024.

[51] Takashi Otonari, Satoshi Ikehata, and Kiyoharu Aizawa. Entity-nerf: Detecting and removing moving entities in urban scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20892–20901, 2024.

[52] Zehao Yu, Torsten Sattler, and Andreas Geiger. Gaussian opacity fields: Efficient adaptive surface reconstruction in unbounded scenes. *ACM Transactions on Graphics (TOG)*, 43(6):1–13, 2024.

[53] Yaru Cao, Zhijian He, Lujia Wang, Wenguan Wang, Yixuan Yuan, Dingwen Zhang, Jinglin Zhang, Pengfei Zhu, Luc Van Gool, Junwei Han, et al. Visdrone-det2021: The vision meets drone object detection challenge results. In *Proceedings of the IEEE/CVF International conference on computer vision*, pages 2847–2854, 2021.

[54] KR Akshatha, AK Karunakar, B Satish Shenoy, K Phani Pavan, V Dhareshwar Chinmay, et al. Manipal-uav person detection dataset: A step towards benchmarking dataset and algorithms for small object detection. *ISPRS Journal of Photogrammetry and Remote Sensing*, 195:77–89, 2023.

[55] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[56] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[57] Jae Sung Park and Dinesh Manocha. Efficient probabilistic collision detection for non-gaussian noise distributions. *IEEE Robotics and Automation Letters*, 5(2):1024–1031, 2020.

[58] Adarsh Jagan Sathyamoorthy, Utsav Patel, Tianrui Guan, and Dinesh Manocha. Frozone: Freezing-free, pedestrian-friendly navigation in human crowds. *IEEE Robotics and Automation Letters*, 2020.

[59] Adarsh Jagan Sathyamoorthy, Jing Liang, Utsav Patel, Tianrui Guan, Rohan Chandra, and Dinesh Manocha. Densecavoid: Real-time navigation in dense crowds using anticipatory behaviors. *arXiv preprint arXiv:2002.03038*, 2020.

[60] Gregory J Stein, Christopher Bradley, and Nicholas Roy. Learning over subgoals for efficient navigation of structured, unknown environments. In *Conference on Robot Learning*, pages 213–222, 2018.

[61] Simon Lynen, Bernhard Zeisl, Dror Aiger, Michael Bosse, Joel Hesch, Marc Pollefeys, Roland Siegwart, and Torsten Sattler. Large-scale, real-time visual–inertial localization revisited. *The International Journal of Robotics Research*, 39(9):1061–1084, 2020.

[62] Muhammad Zubair Irshad, Mauro Comi, Yen-Chen Lin, Nick Heppert, Abhinav Valada, Rares Ambrus, Zsolt Kira, and Jonathan Tremblay. Neural fields in robotics: A survey. *arXiv preprint arXiv:2410.20220*, 2024.

[63] Guangming Wang, Lei Pan, Songyou Peng, Shaohui Liu, Chenfeng Xu, Yanzi Miao, Wei Zhan, Masayoshi Tomizuka, Marc Pollefeys, and Hesheng Wang. Nerf in robotics: A survey. *arXiv preprint arXiv:2405.01333*, 2024.

[64] Timothy Chen, Ola Shorinwa, Joseph Bruno, Aiden Swann, Javier Yu, Weijia Zeng, Keiko Nagami, Philip Dames, and Mac Schwager. Splat-nav: Safe real-time robot navigation in gaussian splatting maps. *arXiv preprint arXiv:2403.02751*, 2024.

[65] JunEn Low, Maximilian Adang, Javier Yu, Keiko Nagami, and Mac Schwager. Sous vide: Cooking visual drone navigation policies in a gaussian splatting vacuum. *arXiv preprint arXiv:2412.16346*, 2024.

[66] Lukas Meyer, Floris Erich, Yusuke Yoshiyasu, Marc Stamminger, Noriaki Ando, and Yukiyasu Domae. Pegasus: Physically enhanced gaussian splatting simulation system for 6dof object pose dataset generation. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10710–10715. IEEE, 2024.

[67] Xiaoshen Han, Minghuan Liu, Yilun Chen, Junqiu Yu, Xiaoyang Lyu, Yang Tian, Bolun Wang, Weinan Zhang, and Jiangmiao Pang. Re ^{3} sim: Generating high-fidelity simulation data via 3d-photorealistic real-to-sim for robotic manipulation. *arXiv preprint arXiv:2502.08645*, 2025.

[68] Shaoting Zhu, Linzhan Mou, Derun Li, Baijun Ye, Runhan Huang, and Hang Zhao. Vr-robo: A real-to-sim-to-real framework for visual robot navigation and locomotion. *arXiv preprint arXiv:2502.01536*, 2025.

[69] Zhengqi Li and Noah Snavely. Megadepth: Learning single-view depth prediction from internet photos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2041–2050, 2018.

[70] Kourosh Sartipi, Tien Do, Tong Ke, Khiem Vuong, and Stergios I Roumeliotis. Deep depth estimation from visual-inertial slam. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10038–10045. IEEE, 2020.

[71] Weifeng Chen, Zhao Fu, Dawei Yang, and Jia Deng. Single-image depth perception in the wild. In *Advances in neural information processing systems*, pages 730–738, 2016.

[72] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks. In *2016 Fourth international conference on 3D vision (3DV)*, pages 239–248. IEEE, 2016.

[73] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. Deep ordinal regression network for monocular depth estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2002–2011, 2018.

[74] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[75] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1851–1858, 2017.

[76] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J Brostow. Digging into self-supervised monocular depth estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3828–3838, 2019.

[77] Jia-Wang Bian, Huangying Zhan, Naiyan Wang, Zhichao Li, Le Zhang, Chunhua Shen, Ming-Ming Cheng, and Ian Reid. Unsupervised scale-consistent depth learning from video. *International Journal of Computer Vision*, 129(9):2548–2564, 2021.

[78] Shuzhe Wang, Vincent Leroy, Yohann Cabon, Boris Chidlovskii, and Jerome Revaud. Dust3r: Geometric 3d vision made easy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20697–20709, 2024.

[79] Jianyuan Wang, Minghao Chen, Nikita Karaev, Andrea Vedaldi, Christian Rupprecht, and David Novotny. Vggt: Visual geometry grounded transformer. *arXiv preprint arXiv:2503.11651*, 2025.

[80] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[81] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE international symposium on mixed and augmented reality*, pages 127–136. Ieee, 2011.

[82] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987.

[83] Brian Karis and Epic Games. Real shading in unreal engine 4. *Proc. Physically Based Shading Theory Practice*, 4(3):1, 2013.

[84] Samuli Laine, Janne Hellsten, Tero Karras, Yeongho Seol, Jaakko Lehtinen, and Timo Aila. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics (TOG)*, 39(6):1–14, 2020.

[85] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[86] James T Kajiya and Brian P Von Herzen. Ray tracing volume densities. *ACM SIGGRAPH computer graphics*, 18(3):165–174, 1984.

[87] Gilad Baruch, Zhuoyuan Chen, Afshin Dehghan, Tal Dimry, Yuri Feigin, Peter Fu, Thomas Gebauer, Brandon Joffe, Daniel Kurz, Arik Schwartz, et al. Arkitscenes: A diverse real-world dataset for 3d indoor scene understanding using mobile rgb-d data. *arXiv preprint arXiv:2111.08897*, 2021.

[88] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-nerf 360: Unbounded anti-aliased neural radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5470–5479, 2022.

[89] Andrew Melnik, Benjamin Alt, Giang Nguyen, Artur Wilkowski, Qirui Wu, Sinan Harms, Helge Rhodin, Manolis Savva, Michael Beetz, et al. Digital twin generation from visual data: A survey. *arXiv preprint arXiv:2504.13159*, 2025.

[90] John A Christian and Scott Cryan. A survey of lidar technology and its use in spacecraft relative navigation. In *AIAA Guidance, Navigation, and Control (GNC) Conference*, page 4641, 2013.

[91] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *European Conference on Computer Vision*, pages 746–760. Springer, 2012.

[92] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361, 2012.

[93] Donghwan Lee, Soohyun Ryu, Suyong Yeon, Yonghan Lee, Deokhwa Kim, Cheolho Han, Yohann Cabon, Philippe Weinzaepfel, Nicolas Guérin, Gabriela Csurka, et al. Large-scale localization datasets in crowded indoor spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3227–3236, 2021.

[94] Fangchang Ma and Sertac Karaman. Sparse-to-dense: Depth prediction from sparse depth samples and a single image. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4796–4803. IEEE, 2018.

[95] Yanchao Yang, Alex Wong, and Stefano Soatto. Dense depth posterior (ddp) from single image and sparse range. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3353–3362, 2019.

[96] Sungho Yoon and Ayoung Kim. Balanced depth completion between dense depth inference and sparse range measurements via kiss-gp. *arXiv preprint arXiv:2008.05158*, 2020.

[97] Richard Hartley. *Multiple view geometry in computer vision*, volume 665. Cambridge university press, 2003.

[98] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[99] Michael Garland and Paul S Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, 1997.

[100] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG)*, 36(4):1–13, 2017.

[101] Dongki Jung, Jaehoon Choi, Yonghan Lee, Somi Jeong, Taejae Lee, Dinesh Manocha, and Suyong Yeon. Edm: Equirectangular projection-oriented dense kernelized feature matching. *arXiv preprint arXiv:2502.20685*, 2025.

[102] Celso M De Melo, Antonio Torralba, Leonidas Guibas, James DiCarlo, Rama Chellappa, and Jessica Hodgins. Next-generation deep learning based on simulators and synthetic data. *Trends in cognitive sciences*, 26(2):174–187, 2022.

[103] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J Black. Smpl: A skinned multi-person linear model. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, pages 851–866. 2023.

[104] Christopher Maxey, Jaehoon Choi, Yonghan Lee, Hyungtae Lee, Dinesh Manocha, and Heesung Kwon. Tk-planes: Tiered k-planes with high dimensional feature vectors for dynamic uav-based scenes. *arXiv preprint arXiv:2405.02762*, 2024.

[105] Guanjun Wu, Taoran Yi, Jiemin Fang, Lingxi Xie, Xiaopeng Zhang, Wei Wei, Wenyu Liu, Qi Tian, and Xinggang Wang. 4d gaussian splatting for real-time dynamic scene rendering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 20310–20320, 2024.

[106] Ziyi Yang, Xinyu Gao, Wen Zhou, Shaohui Jiao, Yuqing Zhang, and Xiaogang Jin. Deformable 3d gaussians for high-fidelity monocular dynamic scene reconstruction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 20331–20341, 2024.

[107] Xinjing Cheng, Peng Wang, Chenye Guan, and Ruigang Yang. Cspn++: Learning context and resource aware convolutional spatial propagation networks for depth completion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 10615–10622, 2020.

[108] Abdelrahman Eldesokey, Michael Felsberg, Karl Holmquist, and Michael Persson. Uncertainty-aware cnns for depth completion: Uncertainty from beginning to end. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12014–12023, 2020.

[109] Jie Tang, Fei-Peng Tian, Wei Feng, Jian Li, and Ping Tan. Learning guided convolutional network for depth completion. *arXiv preprint arXiv:1908.01238*, 2019.

[110] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 270–279, 2017.

[111] Chaoyang Wang, Jose Miguel Buenaposada, Rui Zhu, and Simon Lucey. Learning depth from monocular videos using direct methods. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2022–2030, 2018.

[112] Yanchao Yang, Alex Wong, and Stefano Soatto. Dense depth posterior (ddp) from single image and sparse range. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3353–3362, 2019.

[113] Junsheng Zhou, Yuwang Wang, Kaihuai Qin, and Wenjun Zeng. Moving indoor: Unsupervised video depth learning in challenging environments. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8618–8627, 2019.

[114] Jia-Wang Bian, Huangying Zhan, Naiyan Wang, Tat-Jun Chin, Chunhua Shen, and Ian Reid. Unsupervised depth learning in challenging indoor video: Weak rectification to rescue. *arXiv preprint arXiv:2006.02708*, 2020.

[115] Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger. Sparsity invariant cnns. In *2017 International Conference on 3D Vision (3DV)*, pages 11–20. IEEE, 2017.

[116] Abdelrahman Eldesokey, Michael Felsberg, and Fahad Shahbaz Khan. Propagating confidences through cnns for sparse data regression. *arXiv preprint arXiv:1805.11913*, 2018.

[117] Lucas Teixeira, Martin R Oswald, Marc Pollefeys, and Margarita Chli. Aerial single-view depth completion with image-guided uncertainty estimation. *IEEE Robotics and Automation Letters*, 5(2):1055–1062, 2020.

[118] Yinda Zhang and Thomas Funkhouser. Deep depth completion of a single rgb-d image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 175–185, 2018.

[119] Jiaxiong Qiu, Zhaopeng Cui, Yinda Zhang, Xingdi Zhang, Shuaicheng Liu, Bing Zeng, and Marc Pollefeys. Deeplidar: Deep surface normal guided depth prediction from sparse lidar data and a single color image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3313–3322, 2019.

[120] Yan Xu, Xinge Zhu, Jianping Shi, Guofeng Zhang, Hujun Bao, and Hongsheng Li. Depth completion from sparse lidar data with depth-normal constraints. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2811–2820, 2019.

[121] Maximilian Jaritz, Raoul De Charette, Emilie Wirbel, Xavier Perrotton, and Fawzi Nashashibi. Sparse and dense data with cnns: Depth completion and semantic segmentation. In *2018 International Conference on 3D Vision (3DV)*, pages 52–60, 2018.

[122] Yu-Kai Huang, Tsung-Han Wu, Yueh-Cheng Liu, and Winston H Hsu. Indoor depth completion with boundary consistency and self-attention. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, 2019.

[123] Dmitry Senushkin, Ilia Belikov, and Anton Konushin. Decoder modulation for indoor depth completion. *arXiv preprint arXiv:2005.08607*, 2020.

[124] Yilun Zhang, Ty Nguyen, Ian D. Miller, Shreyas S. Shivakumar, Steven Chen, Camillo J. Taylor, and Vijay Kumar. Dfinenet: Ego-motion estimation and depth refinement from sparse, noisy depth input with rgb guidance. *arXiv preprint arXiv:1903.06397*, 2019.

[125] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o(n) solution to the pnp problem. *International Journal of Computer Vision*, 81(2):155–166, 2009.

[126] Martin A Fischler and Robert C Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[127] Wang Zhao, Shaohui Liu, Yezhi Shu, and Yong-Jin Liu. Towards better generalization: Joint depth-pose learning without posenet. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9151–9161, 2020.

[128] Torsten Sattler, Qunjie Zhou, Marc Pollefeys, and Laura Leal-Taixé. Understanding the limitations of cnn-based absolute camera pose regression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3302–3312, 2019.

[129] Jerome Revaud, Cesar De Souza, Martin Humenberger, and Philippe Weinzaepfel. R2d2: Reliable and repeatable detector and descriptor. In *Advances in Neural Information Processing Systems*, pages 12405–12415, 2019.

[130] Marius Muja and David G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP*, volume 1, pages 331–340, 2009.

[131] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[132] Hang Su, Varun Jampani, Deqing Sun, Orazio Gallo, Erik Learned-Miller, and Jan Kautz. Pixel-adaptive convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11166–11175, 2019.

[133] Donghwan Lee, Soohyun Ryu, Suyong Yeon, Yonghan Lee, Deokhwa Kim, Cheolho Han, Yohann Cabon, Philippe Weinzaepfel, Nicolas Guérin, Gabriela Csurka, et al. Large-scale localization datasets in crowded indoor spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3227–3236, 2021.

[134] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *European conference on computer vision*, pages 746–760. Springer, 2012.

[135] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[136] Adam Paszke, Sam Gross, Soumith Chintala, et al. Automatic differentiation in pytorch. *NIPS-W*, 2017.

[137] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[138] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual worlds as proxy for multi-object tracking analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2016.

[139] Jing Liang, Yi-Ling Qiao, and Dinesh Manocha. Of-vo: Reliable navigation among pedestrians using commodity sensors. *arXiv preprint arXiv:2004.10976*, 2020.

[140] Jing Liang, Utsav Patel, Adarsh Jagan Sathyamoorthy, and Dinesh Manocha. Crowdsteer: Realtime smooth and collision-free robot navigation in dense crowd scenarios trained using high-fidelity simulation. *arXiv preprint arXiv:2004.03089*, 2020.

[141] Yasutaka Furukawa and Carlos Hernández. Multi-view stereo: A tutorial. *Foundations and Trends® in Computer Graphics and Vision*, 9(1-2):1–148, 2015.

[142] Noah Snavely, Steven M Seitz, and Richard Szeliski. Modeling the world from internet photo collections. *International journal of computer vision*, 80(2):189–210, 2008.

[143] Nathan Piasco, Désiré Sidibé, Cédric Demonceaux, and Valérie Gouet-Brunet. A survey on visual-based localization: On the benefit of heterogeneous data. *Pattern Recognition*, 74:90–109, 2018.

[144] Wenzheng Chen, Huan Ling, Jun Gao, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. *Advances in neural information processing systems*, 32, 2019.

[145] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. Building rome in a day. *Communications of the ACM*, 54(10):105–112, 2011.

[146] Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. Mvsnet: Depth inference for unstructured multi-view stereo. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 767–783, 2018.

[147] Onur Ozyesil, Vladislav Voroninski, Ronen Basri, and Amit Singer. A survey of structure from motion. *arXiv preprint arXiv:1701.08493*, 2017.

[148] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in neural information processing systems*, pages 2366–2374, 2014.

[149] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE international conference on computer vision*, pages 2650–2658, 2015.

[150] Fayao Liu, Chunhua Shen, Guosheng Lin, and Ian Reid. Learning depth from single monocular images using deep convolutional neural fields. *IEEE transactions on pattern analysis and machine intelligence*, 38(10):2024–2039, 2015.

[151] Wei Yin, Yifan Liu, Chunhua Shen, and Youliang Yan. Enforcing geometric constraints of virtual normal for depth prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5684–5693, 2019.

[152] Daniel Zoran, Phillip Isola, Dilip Krishnan, and William T Freeman. Learning ordinal relationships for mid-level vision. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 388–396, 2015.

[153] Ke Xian, Chunhua Shen, Zhiguo Cao, Hao Lu, Yang Xiao, Ruibo Li, and Zhenbo Luo. Monocular relative depth perception with web stereo data supervision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 311–320, 2018.

[154] Ke Xian, Jianming Zhang, Oliver Wang, Long Mai, Zhe Lin, and Zhiguo Cao. Structure-guided ranking loss for single image depth prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 611–620, 2020.

[155] Ravi Garg, Vijay Kumar BG, Gustavo Carneiro, and Ian Reid. Unsupervised cnn for single view depth estimation: Geometry to the rescue. In *European Conference on Computer Vision*, pages 740–756. Springer, 2016.

[156] Zhichao Yin and Jianping Shi. Geonet: Unsupervised learning of dense depth, optical flow and camera pose. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1983–1992, 2018.

[157] Vitor Guizilini, Rares Ambrus, Sudeep Pillai, Allan Raventos, and Adrien Gaidon. 3d packing for self-supervised monocular depth estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2485–2494, 2020.

[158] Feng Xue, Guirong Zhuo, Ziyuan Huang, Wufei Fu, Zhuoyue Wu, and Marcelo H Ang Jr. Toward hierarchical self-supervised monocular absolute depth estimation for autonomous driving applications. *arXiv preprint arXiv:2004.05560*, 2020.

[159] Z Yang, P Wang, W Xu, L Zhao, and R Nevatia. Unsupervised learning of geometry with edge-aware depth-normal consistency. arxiv 2017. *arXiv preprint arXiv:1711.03665*.

[160] Chao Liu, Jinwei Gu, Kihwan Kim, Srinivasa G Narasimhan, and Jan Kautz. Neural rgb (r) d sensing: Depth and uncertainty from a video camera. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10986–10995, 2019.

[161] Zachary Teed and Jia Deng. Deepv2d: Video to depth with differentiable structure from motion. In *International Conference on Learning Representations*, 2020.

[162] Xuan Luo, Jia-Bin Huang, Richard Szeliski, Kevin Matzen, and Johannes Kopf. Consistent video depth estimation. *ACM Transactions on Graphics (TOG)*, 2020.

[163] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[164] Chaoyang Wang, Simon Lucey, Federico Perazzi, and Oliver Wang. Web stereo video supervision for depth prediction from dynamic scenes. In *2019 International Conference on 3D Vision (3DV)*, pages 348–357. IEEE, 2019.

[165] Suryansh Kumar, Ram Srivatsav Ghorakavi, Yuchao Dai, and Hongdong Li. Dense depth estimation of a complex dynamic scene without explicit 3d motion estimation. *arXiv preprint arXiv:1902.03791*, 2019.

[166] Min Young Chang, Suyong Yeon, Soohyun Ryu, and Donghwan Lee. Spoxelnet: Spherical voxel-based deep place recognition for 3d point clouds of crowded indoor spaces. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8564–8570, 2020.

[167] Iro Armeni, Sasha Sax, Amir R Zamir, and Silvio Savarese. Joint 2d-3d-semantic data for indoor scene understanding. *arXiv preprint arXiv:1702.01105*, 2017.

[168] Jamie Shotton, Ben Glocker, Christopher Zach, Shahram Izadi, Antonio Criminisi, and Andrew Fitzgibbon. Scene coordinate regression forests for camera relocalization in rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2930–2937, 2013.

[169] Hajime Taira, Masatoshi Okutomi, Torsten Sattler, Mircea Cimpoi, Marc Pollefeys, Josef Sivic, Tomas Pajdla, and Akihiko Torii. Inloc: Indoor visual localization with dense matching and view synthesis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7199–7209, 2018.

[170] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 573–580. IEEE, 2012.

[171] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[172] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2462–2470, 2017.

[173] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[174] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[175] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 2004.

[176] Zehao Yu, Lei Jin, and Shenghua Gao. P2net: Patch-match and plane-regularization for unsupervised indoor depth estimation. *arXiv preprint arXiv:2007.07696*, 2020.

[177] Apple arkit. `https://developer.apple.com/documentation/arkit/`. Accessed: 2022-11-09.

[178] Google arcore. `https://developers.google.com/ar`. Accessed: 2022-11-09.

[179] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (ToG)*, 32(6):1–11, 2013.

[180] Thomas Whelan, Stefan Leutenegger, Renato Salas-Moreno, Ben Glocker, and Andrew Davison. Elasticfusion: Dense slam without a pose graph. Robotics: Science and Systems, 2015.

[181] Angela Dai, Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Christian Theobalt. Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration. *ACM Transactions on Graphics (ToG)*, 36(4):1, 2017.

[182] Alex Flint, Oleg Naroditsky, Christopher P Broaddus, Andriy Grygorenko, Stergios Roumeliotis, and Oriel Bergig. Visual-based inertial navigation, December 11 2018. US Patent 10,152,795.

[183] Jiaming Sun, Xi Chen, Qianqian Wang, Zhengqi Li, Hadar Averbuch-Elor, Xiaowei Zhou, and Noah Snavely. Neural 3d reconstruction in the wild. In *ACM SIGGRAPH 2022 Conference Proceedings*, pages 1–9, 2022.

[184] Changchang Wu. Towards linear-time incremental structure from motion. In *2013 International Conference on 3D Vision-3DV 2013*, pages 127–134. IEEE, 2013.

[185] Noah Snavely, Steven M Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3d. In *ACM siggraph 2006 papers*, pages 835–846. 2006.

[186] Aji Resindra Widya, Yusuke Monno, Masatoshi Okutomi, Sho Suzuki, Takuji Gotoda, and Kenji Miki. Stomach 3d reconstruction based on virtual chromoendoscopic image generation. In *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 1848–1852. IEEE, 2020.

[187] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[188] George Vogiatzis, Philip HS Torr, and Roberto Cipolla. Multi-view stereo via volumetric graph-cuts. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 391–398. IEEE, 2005.

[189] Silvano Galliani, Katrin Lasinger, and Konrad Schindler. Massively parallel multiview stereopsis by surface normal diffusion. June 2015.

[190] Qingshan Xu, Weihang Kong, Wenbing Tao, and Marc Pollefeys. Multi-scale geometric consistency guided and planar prior assisted multi-view stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[191] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, volume 7, 2006.

[192] Patrick Labatut, J-P Pons, and Renaud Keriven. Robust and efficient surface reconstruction from range data. In *Computer graphics forum*, volume 28, pages 2275–2290. Wiley Online Library, 2009.

[193] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312, 1996.

[194] Ricardo Martin-Brualla, Noha Radwan, Mehdi SM Sajjadi, Jonathan T Barron, Alexey Dosovitskiy, and Daniel Duckworth. Nerf in the wild: Neural radiance fields for unconstrained photo collections. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7210–7219, 2021.

[195] Alex Yu, Ruilong Li, Matthew Tancik, Hao Li, Ren Ng, and Angjoo Kanazawa. Plenoctrees for real-time rendering of neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5752–5761, 2021.

[196] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *arXiv preprint arXiv:2201.05989*, 2022.

[197] Kai Zhang, Gernot Riegler, Noah Snavely, and Vladlen Koltun. Nerf++: Analyzing and improving neural radiance fields. *arXiv preprint arXiv:2010.07492*, 2020.

[198] Lior Yariv, Yoni Kasten, Dror Moran, Meirav Galun, Matan Atzmon, Basri Ronen, and Yaron Lipman. Multiview neural surface reconstruction by disentangling geometry and appearance. *Advances in Neural Information Processing Systems*, 33:2492–2502, 2020.

[199] Jingyang Zhang, Yao Yao, and Long Quan. Learning signed distance field for multi-view surface reconstruction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6525–6534, 2021.

[200] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3504–3515, 2020.

[201] Michael Oechsle, Songyou Peng, and Andreas Geiger. Unisurf: Unifying neural implicit surfaces and radiance fields for multi-view reconstruction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5589–5599, 2021.

[202] Dejan Azinović, Ricardo Martin-Brualla, Dan B Goldman, Matthias Nießner, and Justus Thies. Neural rgb-d surface reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6290–6301, 2022.

[203] François Darmon, Bénédicte Bascle, Jean-Clément Devaux, Pascal Monasse, and Mathieu Aubry. Improving neural implicit surfaces geometry with patch warping. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6260–6269, 2022.

[204] Xiaoxiao Long, Cheng Lin, Peng Wang, Taku Komura, and Wenping Wang. Sparseneus: Fast generalizable neural surface reconstruction from sparse views. *arXiv preprint arXiv:2206.05737*, 2022.

[205] Jiepeng Wang, Peng Wang, Xiaoxiao Long, Christian Theobalt, Taku Komura, Lingjie Liu, and Wenping Wang. Neuris: Neural reconstruction of indoor scenes using normal priors. *arXiv preprint arXiv:2206.13597*, 2022.

[206] Haoyu Guo, Sida Peng, Haotong Lin, Qianqian Wang, Guofeng Zhang, Hujun Bao, and Xiaowei Zhou. Neural 3d scene reconstruction with the manhattan-world assumption. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5511–5520, 2022.

[207] Victor Lempitsky and Denis Ivanov. Seamless mosaicing of image-based texture maps. In *2007 IEEE conference on computer vision and pattern recognition*, pages 1–6. IEEE, 2007.

[208] Sai Bi, Nima Khademi Kalantari, and Ravi Ramamoorthi. Patch-based optimization for image-based texture mapping. *ACM Trans. Graph.*, 36(4):106–1, 2017.

[209] Angjoo Kanazawa, Shubham Tulsiani, Alexei A. Efros, and Jitendra Malik. Learning category-specific mesh reconstruction from image collections. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[210] Shubham Goel, Angjoo Kanazawa, and Jitendra Malik. Shape and viewpoint without keypoints. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XV*, page 88–104, Berlin, Heidelberg, 2020. Springer-Verlag.

[211] Jingwei Huang, Justus Thies, Angela Dai, Abhijit Kundu, Chiyu Jiang, Leonidas J Guibas, Matthias Nießner, Thomas Funkhouser, et al. Adversarial texture optimization from rgb-d scans. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1559–1568, 2020.

[212] Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. *ACM Trans. Graph.*, 38(4), jul 2019.

[213] Fanbo Xiang, Zexiang Xu, Milos Hasan, Yannick Hold-Geoffroy, Kalyan Sunkavalli, and Hao Su. Neutex: Neural texture mapping for volumetric neural rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7119–7128, June 2021.

[214] Bangbang Yang, Chong Bao, Junyi Zeng, Hujun Bao, Yinda Zhang, Zhaopeng Cui, and Guofeng Zhang. Neumesh: Learning disentangled neural mesh-based implicit field for geometry and texture editing. In *European Conference on Computer Vision*, pages 597–614. Springer, 2022.

[215] Paul Henderson, Vagia Tsiminaki, and Christoph H Lampert. Leveraging 2d data to learn textured 3d mesh generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7498–7507, 2020.

[216] Shubham Goel, Georgia Gkioxari, and Jitendra Malik. Differentiable stereopsis: Meshes from multiple views using differentiable rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8635–8644, June 2022.

[217] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. Pointrend: Image segmentation as rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[218] Amos Gropp, Lior Yariv, Niv Haim, Matan Atzmon, and Yaron Lipman. Implicit geometric regularization for learning shapes. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

[219] Michael Zollhöfer, Patrick Stotko, Andreas Görlitz, Christian Theobalt, Matthias Nießner, Reinhard Klein, and Andreas Kolb. State of the art on 3d reconstruction with rgb-d cameras. In *Computer graphics forum*, volume 37, pages 625–652. Wiley Online Library, 2018.

[220] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In *CVPR Deep Learning for Visual SLAM Workshop*, 2018.

[221] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[222] Matan Atzmon and Yaron Lipman. Sal: Sign agnostic learning of shapes from raw data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2565–2574, 2020.

[223] Zehao Yu, Songyou Peng, Michael Niemeyer, Torsten Sattler, and Andreas Geiger. Monosdf: Exploring monocular geometric cues for neural implicit surface reconstruction. *Advances in neural information processing systems*, 35:25018–25032, 2022.

[224] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5855–5864, 2021.

[225] Marie-Julie Rakotosaona, Fabian Manhardt, Diego Martin Arroyo, Michael Niemeyer, Abhijit Kundu, and Federico Tombari. Nerfmeshing: Distilling neural radiance fields into geometrically-accurate 3d meshes. In *Proc. of the International Conf. on 3D Vision (3DV)*, 2023.

[226] Yiyi Liao, Simon Donne, and Andreas Geiger. Deep marching cubes: Learning explicit surface representations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2916–2925, 2018.

[227] Jun Gao, Wenzheng Chen, Tommy Xiang, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Learning deformable tetrahedral meshes for 3d reconstruction. *Advances In Neural Information Processing Systems*, 33:9936–9947, 2020.

[228] Tianchang Shen, Jun Gao, Kangxue Yin, Ming-Yu Liu, and Sanja Fidler. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. *Advances in Neural Information Processing Systems*, 34:6087–6101, 2021.

[229] Tianchang Shen, Jacob Munkberg, Jon Hasselgren, Kangxue Yin, Zian Wang, Wenzheng Chen, Zan Gojcic, Sanja Fidler, Nicholas Sharp, and Jun Gao. Flexible isosurface extraction for gradient-based mesh optimization. *ACM Transactions on Graphics (TOG)*, 42(4):1–16, 2023.

[230] Jon Hasselgren, Nikolai Hofmann, and Jacob Munkberg. Shape, light, and material decomposition from images using monte carlo rendering and denoising. *Advances in Neural Information Processing Systems*, 35:22856–22869, 2022.

[231] Jon Hasselgren, Nikolai Hofmann, and Jacob Munkberg. Shape, Light, and Material Decomposition from Images using Monte Carlo Rendering and Denoising. *arXiv:2206.03380*, 2022.

[232] Michael Garland and Paul S Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In *Proceedings Visualization'98 (Cat. No. 98CB36276)*, pages 263–269. IEEE, 1998.

[233] Baptiste Nicolet, Alec Jacobson, and Wenzel Jakob. Large steps in inverse rendering of geometry. *ACM Transactions on Graphics (TOG)*, 40(6):1–13, 2021.

[234] Werner Palfinger. Continuous remeshing for inverse rendering. *Computer Animation and Virtual Worlds*, 33(5):e2101, 2022.

[235] Sara Fridovich-Keil, Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5501–5510, 2022.

[236] Anpei Chen, Zexiang Xu, Andreas Geiger, Jingyi Yu, and Hao Su. Tensorf: Tensorial radiance fields. In *European Conference on Computer Vision*, pages 333–350. Springer, 2022.

[237] Christian Reiser, Songyou Peng, Yiyi Liao, and Andreas Geiger. Kilonerf: Speeding up neural radiance fields with thousands of tiny mlps. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14335–14345, 2021.

[238] Peter Hedman, Pratul P Srinivasan, Ben Mildenhall, Jonathan T Barron, and Paul Debevec. Baking neural radiance fields for real-time view synthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5875–5884, 2021.

[239] Michael Deering, Stephanie Winner, Bic Schediwy, Chris Duffy, and Neil Hunt. The triangle processor and normal vector shader: a vlsi system for high performance graphics. *Acm siggraph computer graphics*, 22(4):21–30, 1988.

[240] Fan Lu, Yan Xu, Guang Chen, Hongsheng Li, Kwan-Yee Lin, and Changjun Jiang. Urban radiance field representation with deformable neural mesh primitives. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 465–476, 2023.

[241] Jeffrey Yunfan Liu, Yun Chen, Ze Yang, Jingkang Wang, Sivabalan Manivasagam, and Raquel Urtasun. Real-time neural rasterization for large scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8416–8427, 2023.

[242] William J Schroeder, Jonathan A Zarge, and William E Lorensen. Decimation of triangle meshes. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 65–70, 1992.

[243] Jarek Rossignac and Paul Borrel. Multi-resolution 3d approximations for rendering complex scenes. In *Modeling in computer graphics: methods and applications*, pages 455–465. Springer, 1993.

[244] Hugues Hoppe. View-dependent refinement of progressive meshes. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 189–198, 1997.

[245] Michael Garland. *Quadric-based polygonal surface simplification*. Carnegie Mellon University, 1999.

[246] Carl Erikson and Dinesh Manocha. Gaps: General and automatic polygonal simplification. In *Proceedings of the 1999 symposium on Interactive 3D graphics*, pages 79–88, 1999.

[247] Jonathan Cohen, Marc Olano, and Dinesh Manocha. Appearance-preserving simplification. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 115–122, 1998.

[248] Peter Lindstrom. Out-of-core simplification of large polygonal models. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 259–262, 2000.

[249] Peter Lindstrom and Claudio T Silva. A memory insensitive technique for large model simplification. In *Proceedings Visualization, 2001. VIS'01.*, pages 121–550. IEEE, 2001.

[250] Chao Wang and Xiaohu Guo. Plane-based optimization of geometry and texture for rgb-d reconstruction of indoor scenes. In *2018 International Conference on 3D Vision (3DV)*, pages 533–541. IEEE, 2018.

[251] Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. Differentiable rendering: A survey. *arXiv preprint arXiv:2006.12057*, 2020.

[252] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3d reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7708–7717, 2019.

[253] Justin Johnson, Nikhila Ravi, Jeremy Reizenstein, David Novotny, Shubham Tulsiani, Christoph Lassner, and Steve Branson. Accelerating 3d deep learning with pytorch3d. In *SIGGRAPH Asia 2020 Courses*, pages 1–1. 2020.

[254] Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, Merlin Nimier-David, Delio Vicini, Tizian Zeltner, Baptiste Nicolet, Miguel Crespo, Vincent Leroy, and Ziyi Zhang. Mitsuba 3 renderer, 2022. https://mitsuba-renderer.org.

[255] Selena Zihan Ling, Nicholas Sharp, and Alec Jacobson. Vectoradam for rotation equivariant geometry optimization. *Advances in Neural Information Processing Systems*, 35:4111–4122, 2022.

[256] Nelson Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2):99–108, 1995.

[257] Fujun Luan, Shuang Zhao, Kavita Bala, and Zhao Dong. Unified shape and svbrdf recovery using differentiable monte carlo rendering. In *Computer Graphics Forum*, volume 40, pages 101–113. Wiley Online Library, 2021.

[258] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. Laplacian mesh optimization. In *Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, pages 381–389, 2006.

[259] Leif Kobbelt. Sqrt(3)-subdivision. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 103–112, 2000.

[260] Zehao Yu, Anpei Chen, Bozidar Antic, Songyou Peng, Apratim Bhattacharyya, Michael Niemeyer, Siyu Tang, Torsten Sattler, and Andreas Geiger. Sdfstudio: A unified framework for surface reconstruction, 2022.

[261] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.

[262] Brian Karis and Epic Games. Real shading in unreal engine 4. *Proc. Physically Based Shading Theory Practice*, 4(3):1, 2013.

[263] Yuan-Chen Guo, Yan-Pei Cao, Chen Wang, Yu He, Ying Shan, and Song-Hai Zhang. Vmesh: Hybrid volume-mesh representation for efficient view synthesis. In *SIGGRAPH Asia 2023 Conference Papers*, pages 1–11, 2023.

[264] Mark Segal and Kurt Akeley. The opengl graphics system: A specification (version 4.6 (core profile)-october 22, 2019). *The Khronos Group Inc.[cit. 2020-12-05]. Dostupné z: https://www. khronos. org/registry/0penGL/specs/gl/glspec46. core. pdf*, 2021.

[265] Shenhan Qian, Tobias Kirschstein, Liam Schoneveld, Davide Davoli, Simon Giebenhain, and Matthias Nießner. Gaussianavatars: Photorealistic head avatars with rigged 3d gaussians. *arXiv preprint arXiv:2312.02069*, 2023.

[266] Cheng Sun, Min Sun, and Hwann-Tzong Chen. Direct voxel grid optimization: Super-fast convergence for radiance fields reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5459–5469, 2022.

[267] Christian Reiser, Rick Szeliski, Dor Verbin, Pratul Srinivasan, Ben Mildenhall, Andreas Geiger, Jon Barron, and Peter Hedman. Merf: Memory-efficient radiance fields for real-time view synthesis in unbounded scenes. *ACM Transactions on Graphics (TOG)*, 42(4):1–12, 2023.

[268] Zhaoshuo Li, Thomas Müller, Alex Evans, Russell H Taylor, Mathias Unberath, Ming-Yu Liu, and Chen-Hsuan Lin. Neuralangelo: High-fidelity neural surface reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8456–8465, 2023.

[269] Zian Wang, Tianchang Shen, Merlin Nimier-David, Nicholas Sharp, Jun Gao, Alexander Keller, Sanja Fidler, Thomas Müller, and Zan Gojcic. Adaptive shells for efficient neural radiance field rendering. *arXiv preprint arXiv:2311.10091*, 2023.

[270] Georgios Kopanas, Julien Philip, Thomas Leimkühler, and George Drettakis. Point-based neural rendering with per-view optimization. In *Computer Graphics Forum*, volume 40, pages 29–43. Wiley Online Library, 2021.

[271] Darius Rückert, Linus Franke, and Marc Stamminger. Adop: Approximate differentiable one-pixel point rendering. *ACM Transactions on Graphics (ToG)*, 41(4):1–14, 2022.

[272] Christoph Lassner and Michael Zollhofer. Pulsar: Efficient sphere-based neural rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1440–1449, 2021.

[273] Wang Yifan, Felice Serena, Shihao Wu, Cengiz Öztireli, and Olga Sorkine-Hornung. Differentiable surface splatting for point-based geometry processing. *ACM Transactions on Graphics (TOG)*, 38(6):1–14, 2019.

[274] Matthias Zwicker, Hanspeter Pfister, Jeroen Van Baar, and Markus Gross. Ewa splatting. *IEEE Transactions on Visualization and Computer Graphics*, 8(3):223–238, 2002.

[275] Mixamo.

[276] Mateusz Janiszewski, Masoud Torkan, Lauri Uotinen, and Mikael Rinne. Rapid photogrammetry with a 360-degree camera for tunnel mapping. *Remote Sensing*, 14(21):5494, 2022.

[277] Sorin Herban, Domenica Costantino, Vincenzo Saverio Alfio, and Massimiliano Pepe. Use of low-cost spherical cameras for the digitisation of cultural heritage structures into 3d point clouds. *Journal of Imaging*, 8(1):13, 2022.

[278] Pierre Moulon, Pascal Monasse, Romuald Perrot, and Renaud Marlet. Openmvg: Open multiple view geometry. In *Reproducible Research in Pattern Recognition: First International Workshop, RRPR 2016, Cancún, Mexico, December 4, 2016, Revised Selected Papers 1*, pages 60–74. Springer, 2017.

[279] Jiaming Sun, Zehong Shen, Yuang Wang, Hujun Bao, and Xiaowei Zhou. Loftr: Detector-free local feature matching with transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8922–8931, 2021.

[280] Iaroslav Melekhov, Aleksei Tiulpin, Torsten Sattler, Marc Pollefeys, Esa Rahtu, and Juho Kannala. Dgc-net: Dense geometric correspondence network. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1034–1042. IEEE, 2019.

[281] Prune Truong, Martin Danelljan, and Radu Timofte. Glu-net: Global-local universal network for dense flow and correspondences. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6258–6268, 2020.

[282] Prune Truong, Martin Danelljan, Luc Van Gool, and Radu Timofte. Learning accurate dense correspondences and when to trust them. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 5714–5724, 2021.

[283] Johan Edstedt, Ioannis Athanasiadis, Mårten Wadenbäck, and Michael Felsberg. Dkm: Dense kernelized feature matching for geometry estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17765–17775, 2023.

[284] Johan Edstedt, Qiyu Sun, Georg Bökman, Mårten Wadenbäck, and Michael Felsberg. Roma: Robust dense feature matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19790–19800, 2024.

[285] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 224–236, 2018.

[286] Michał Tyszkiewicz, Pascal Fua, and Eduard Trulls. Disk: Learning local features with policy gradient. *Advances in Neural Information Processing Systems*, 33:14254–14265, 2020.

[287] Guangcong, Zhaoxi Chen, Chen Change Loy, and Ziwei Liu. Sparsenerf: Distilling depth ranking for few-shot novel view synthesis. *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[288] T. Berriel Martins and Javier Civera. Feature splatting for better novel view synthesis with low overlap. 2024.

[289] Yonghan Lee, Jaehoon Choi, Dongki Jung, Jaeseong Yun, Soohyun Ryu, Dinesh Manocha, and Suyong Yeon. Mode-gs: Monocular depth guided anchored 3d gaussian splatting for robust ground-view scene rendering. *arXiv preprint arXiv:2410.04646*, 2024.

[290] Yuting Xiao, Jingwei Xu, Zehao Yu, and Shenghua Gao. Debsdf: Delving into the details and bias of neural indoor scene reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[291] Santhosh Kumar Ramakrishnan, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alexander Clegg, John M Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel X Chang, Manolis Savva, Yili Zhao, and Dhruv Batra. Habitat-matterport 3d dataset (HM3d): 1000 large-scale 3d environments for embodied AI. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021.

[292] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60:91–110, 2004.

[293] Paul-Edouard Sarlin, Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superglue: Learning feature matching with graph neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4938–4947, 2020.

[294] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[295] Chris Sweeney. Theia multiview geometry library: Tutorial & reference, 2015.

[296] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment—a modern synthesis. In *Vision Algorithms: Theory and Practice: International Workshop on Vision Algorithms Corfu, Greece, September 21–22, 1999 Proceedings*, pages 298–372. Springer, 2000.

[297] Linfei Pan, Dániel Baráth, Marc Pollefeys, and Johannes L Schönberger. Global structure-from-motion revisited. In *European Conference on Computer Vision (ECCV)*, 2024.

[298] Xingyi He, Jiaming Sun, Yifan Wang, Sida Peng, Qixing Huang, Hujun Bao, and Xiaowei Zhou. Detector-free structure from motion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21594–21603, 2024.

[299] San Jiang, Kan You, Yaxin Li, Duojie Weng, and Wu Chen. 3d reconstruction of spherical images: a review of techniques, applications, and prospects. *Geo-spatial Information Science*, pages 1–30, 2024.

[300] Bolivar Solarte, Chin-Hsuan Wu, Kuan-Wei Lu, Yi-Hsuan Tsai, Wei-Chen Chiu, and Min Sun. Robust 360-8pa: Redesigning the normalized 8-point algorithm for 360-fov images. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11032–11038. IEEE, 2021.

[301] Yanping Fu, Qingan Yan, Long Yang, Jie Liao, and Chunxia Xiao. Texture mapping for 3d reconstruction with rgb-d sensor. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4645–4653, 2018.

[302] Changwoon Choi, Sang Min Kim, and Young Min Kim. Balanced spherical grid for egocentric view synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16590–16599, 2023.

[303] Qiang Zhao, Wei Feng, Liang Wan, and Jiawan Zhang. Sphorb: A fast and robust binary feature on the sphere. *International journal of computer vision*, 113:143–159, 2015.

[304] Christiano Gava, Vishal Mukunda, Tewodros Habtegebrial, Federico Raue, Sebastian Palacio, and Andreas Dengel. Sphereglue: Learning keypoint matching on high resolution spherical images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6134–6144, 2023.

[305] Paul-Edouard Sarlin, Cesar Cadena, Roland Siegwart, and Marcin Dymczyk. From coarse to fine: Robust hierarchical localization at large scale. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12716–12725, 2019.

[306] Hongkai Chen, Zixin Luo, Lei Zhou, Yurun Tian, Mingmin Zhen, Tian Fang, David Mckinnon, Yanghai Tsin, and Long Quan. Aspanformer: Detector-free image matching with adaptive span transformer. In *European Conference on Computer Vision*, pages 20–36. Springer, 2022.

[307] Zehong Shen, Jiaming Sun, Yuang Wang, Xingyi He, Hujun Bao, and Xiaowei Zhou. Semi-dense feature matching with transformers and its applications in multiple-view geometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(6):7726–7738, 2022.

[308] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *IEEE robotics & automation magazine*, 18(4):80–92, 2011.

[309] Ainaz Eftekhar, Alexander Sax, Jitendra Malik, and Amir Zamir. Omnidata: A scalable pipeline for making multi-task mid-level vision datasets from 3d scans. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10786–10796, 2021.

[310] Marc Eder, Mykhailo Shvets, John Lim, and Jan-Michael Frahm. Tangent images for mitigating spherical distortion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12426–12434, 2020.

[311] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 586–595, 2018.

[312] Yi-Ting Shen, Yaesop Lee, Heesung Kwon, Damon M Conover, Shuvra S Bhattacharyya, Nikolas Vale, Joshua D Gray, G Jeremy Leong, Kenneth Evensen, and Frank Skirlo. Archangel: A hybrid uav-based human detection benchmark with position and pose metadata. *IEEE Access*, 11:80958–80972, 2023.

[313] Tianjiao Li, Jun Liu, Wei Zhang, Yun Ni, Wenqian Wang, and Zhiheng Li. Uav-human: A large benchmark for human behavior understanding with unmanned aerial vehicles. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16266–16275, 2021.

[314] Yi-Ting Shen, Hyungtae Lee, Heesung Kwon, and Shuvra S Bhattacharyya. Progressive transformation learning for leveraging virtual images in training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 835–844, 2023.

[315] Yi-Ting Shen, Hyungtae Lee, Heesung Kwon, and Shuvra S Bhattacharyya. Diversifying human pose in synthetic data for aerial-view human detection. *arXiv preprint arXiv:2405.15939*, 2024.

[316] Wenshan Wang, Yaoyu Hu, and Sebastian Scherer. Tartanvo: A generalizable learning-based vo. In *Conference on Robot Learning*, pages 1761–1772. PMLR, 2021.

[317] Nitin Saini, Elia Bonetto, Eric Price, Aamir Ahmad, and Michael J Black. Airpose: Multi-view fusion network for aerial 3d human pose and shape estimation. *IEEE Robotics and Automation Letters*, 7(2):4805–4812, 2022.

[318] Elia Bonetto and Aamir Ahmad. Synthetic data-based detection of zebras in drone imagery. In *2023 European Conference on Mobile Robots (ECMR)*, pages 1–8. IEEE, 2023.

[319] Sahil Khose, Anisha Pal, Aayushi Agarwal, Deepanshi, Judy Hoffman, and Prithvijit Chattopadhyay. Skyscenes: A synthetic dataset for aerial scene understanding. In *European Conference on Computer Vision*, pages 19–35. Springer, 2024.

[320] Giulia Rizzoli, Francesco Barbato, Matteo Caligiuri, and Pietro Zanuttigh. Syndrone-multimodal uav dataset for urban scenarios. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2210–2220, 2023.

[321] Zhitao Yang, Zhongang Cai, Haiyi Mei, Shuai Liu, Zhaoxi Chen, Weiye Xiao, Yukun Wei, Zhongfei Qing, Chen Wei, Bo Dai, et al. Synbody: Synthetic dataset with layered human models for 3d human perception and modeling. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 20282–20292, 2023.

[322] Michael J Black, Priyanka Patel, Joachim Tesch, and Jinlong Yang. Bedlam: A synthetic dataset of bodies exhibiting detailed lifelike animated motion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8726–8737, 2023.

[323] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

[324] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics: Results of the 11th International Conference*, pages 621–635. Springer, 2018.

[325] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam, 2022.

[326] Jinsub Yim. Synplay: Importing real-world diversity for a synthetic human dataset. Master's thesis, University of Maryland, College Park, 2024.

[327] Ricardo Martin-Brualla, Noha Radwan, Mehdi SM Sajjadi, Jonathan T Barron, Alexey Dosovitskiy, and Daniel Duckworth. Nerf in the wild: Neural radiance fields for unconstrained photo collections. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7210–7219, 2021.

[328] Xingyu Chen, Qi Zhang, Xiaoyu Li, Yue Chen, Ying Feng, Xuan Wang, and Jue Wang. Hallucinated neural radiance fields in the wild. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12943–12952, 2022.

[329] Jonas Kulhanek, Songyou Peng, Zuzana Kukelova, Marc Pollefeys, and Torsten Sattler. Wildgaussians: 3d gaussian splatting in the wild. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[330] Dongbin Zhang, Chuming Wang, Weitao Wang, Peihao Li, Minghan Qin, and Haoqian Wang. Gaussian in the wild: 3d gaussian splatting for unconstrained image collections. In *European Conference on Computer Vision*, pages 341–359. Springer, 2025.

[331] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4015–4026, 2023.

[332] Zehao Yu, Anpei Chen, Binbin Huang, Torsten Sattler, and Andreas Geiger. Mip-splatting: Alias-free 3d gaussian splatting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19447–19456, 2024.

[333] Qi Lu, Jason Kuen, Shen Tiancheng, Gu Jiuxiang, Guo Weidong, Jia Jiaya, Lin Zhe, and Yang Ming-Hsuan. High-quality entity segmentation. In *ICCV*, 2023.

[334] Konstantinos Rematas, Andrew Liu, Pratul P Srinivasan, Jonathan T Barron, Andrea Tagliasacchi, Thomas Funkhouser, and Vittorio Ferrari. Urban radiance fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12932–12942, 2022.

[335] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. In *European Conference on Computer Vision*, pages 38–55. Springer, 2024.

[336] Neill DF Campbell, George Vogiatzis, Carlos Hernández, and Roberto Cipolla. Using multiple hypotheses to improve depth-maps for multi-view stereo. In *Computer Vision–ECCV 2008: 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part I 10*, pages 766–779. Springer, 2008.

[337] Georgios Pavlakos, Vasileios Choutas, Nima Ghorbani, Timo Bolkart, Ahmed AA Osman, Dimitrios Tzionas, and Michael J Black. Expressive body capture: 3d hands, face, and body from a single image. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10975–10985, 2019.

[338] Naureen Mahmood, Nima Ghorbani, Nikolaus F Troje, Gerard Pons-Moll, and Michael J Black. Amass: Archive of motion capture as surface shapes. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5442–5451, 2019.

[339] Pakkapon Phongthawee, Worameth Chinchuthakun, Nontaphat Sinsunthithet, Varun Jampani, Amit Raj, Pramook Khungurn, and Supasorn Suwajanakorn. Diffusionlight: Light probes for free by painting a chrome ball. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 98–108, 2024.

[340] Jiadong Tang, Yu Gao, Dianyi Yang, Liqi Yan, Yufeng Yue, and Yi Yang. Dronesplat: 3d gaussian splatting for robust 3d reconstruction from in-the-wild drone imagery. *arXiv preprint arXiv:2503.16964*, 2025.

[341] Viktor Rudnev, Mohamed Elgharib, William Smith, Lingjie Liu, Vladislav Golyanik, and Christian Theobalt. Nerf for outdoor scene relighting. In *European Conference on Computer Vision*, pages 615–631. Springer, 2022.

[342] Peihao Li, Shaohui Wang, Chen Yang, Bingbing Liu, Weichao Qiu, and Haoqian Wang. Nerf-ms: neural radiance fields with multi-sequence. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 18591–18600, 2023.

[343] Jiaqi Lin, Zhihao Li, Xiao Tang, Jianzhuang Liu, Shiyong Liu, Jiayue Liu, Yangdi Lu, Xiaofei Wu, Songcen Xu, Youliang Yan, et al. Vastgaussian: Vast 3d gaussians for large scene reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5166–5175, 2024.

[344] Christopher Maxey, Jaehoon Choi, Hyungtae Lee, Dinesh Manocha, and Heesung Kwon. Uav-sim: Nerf-based synthetic data generation for uav-based perception. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5323–5329. IEEE, 2024.

[345] Jiaxuan Zhu and Hao Tang. Dynamic scene reconstruction: Recent advance in real-time rendering and streaming. *arXiv preprint arXiv:2503.08166*, 2025.

[346] Colton Stearns, Adam Harley, Mikaela Uy, Florian Dubost, Federico Tombari, Gordon Wetzstein, and Leonidas Guibas. Dynamic gaussian marbles for novel view synthesis of casual monocular videos. In *SIGGRAPH Asia 2024 Conference Papers*, pages 1–11, 2024.

[347] Qianqian Wang, Vickie Ye, Hang Gao, Weijia Zeng, Jake Austin, Zhengqi Li, and Angjoo Kanazawa. Shape of motion: 4d reconstruction from a single video. 2024.

[348] Jonathon Luiten, Georgios Kopanas, Bastian Leibe, and Deva Ramanan. Dynamic 3d gaussians: Tracking by persistent dynamic view synthesis. In *2024 International Conference on 3D Vision (3DV)*, pages 800–809. IEEE, 2024.

[349] Zhen Xu, Sida Peng, Haotong Lin, Guangzhao He, Jiaming Sun, Yujun Shen, Hujun Bao, and Xiaowei Zhou. 4k4d: Real-time 4d view synthesis at 4k resolution. In *CVPR*, 2024.

[350] Ziyu Chen, Jiawei Yang, Jiahui Huang, Riccardo de Lutio, Janick Martinez Esturo, Boris Ivanovic, Or Litany, Zan Gojcic, Sanja Fidler, Marco Pavone, Li Song, and Yue Wang. Omnire: Omni urban scene reconstruction. In *The Thirteenth International Conference on Learning Representations*, 2025.

[351] Sara Fridovich-Keil, Giacomo Meanti, Frederik Rahbæk Warburg, Benjamin Recht, and Angjoo Kanazawa. K-planes: Explicit radiance fields in space, time, and appearance. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12479–12488, 2023.

[352] Hang Gao, Ruilong Li, Shubham Tulsiani, Bryan Russell, and Angjoo Kanazawa. Monocular dynamic view synthesis: A reality check. *Advances in Neural Information Processing Systems*, 35:33768–33780, 2022.

[353] Ruijie Zhu, Yanzhe Liang, Hanzhi Chang, Jiacheng Deng, Jiahao Lu, Wenfei Yang, Tianzhu Zhang, and Yongdong Zhang. Motiongs: Exploring explicit motion guidance for deformable 3d gaussian splatting. *Advances in Neural Information Processing Systems*, 37:101790–101817, 2024.

[354] Chengwei Zheng, Lixin Xue, Juan Zarate, and Jie Song. Gstar: Gaussian surface tracking and reconstruction. *arXiv preprint arXiv:2501.10283*, 2025.

[355] Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything: Unleashing the power of large-scale unlabeled data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10371–10381, 2024.

[356] Haofei Xu, Jing Zhang, Jianfei Cai, Hamid Rezatofighi, and Dacheng Tao. Gmflow: Learning optical flow via global matching. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8121–8130, 2022.

[357] Nikita Karaev, Ignacio Rocco, Benjamin Graham, Natalia Neverova, Andrea Vedaldi, and Christian Rupprecht. Cotracker: It is better to track together. In *European Conference on Computer Vision*, pages 18–35. Springer, 2024.

[358] Qianqian Wang, Yifei Zhang, Aleksander Holynski, Alexei A Efros, and Angjoo Kanazawa. Continuous 3d perception model with persistent state. *arXiv preprint arXiv:2501.12387*, 2025.

[359] Haiwen Feng, Junyi Zhang, Qianqian Wang, Yufei Ye, Pengcheng Yu, Michael J Black, Trevor Darrell, and Angjoo Kanazawa. St4rtrack: Simultaneous 4d reconstruction and tracking in the world. *arXiv preprint arXiv:2504.13152*, 2025.

[360] Muhammed Kocabas, Jen-Hao Rick Chang, James Gabriel, Oncel Tuzel, and Anurag Ranjan. Hugs: Human gaussian splats. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 505–515, 2024.

[361] Shubham Goel, Georgios Pavlakos, Jathushan Rajasegaran, Angjoo Kanazawa, and Jitendra Malik. Humans in 4d: Reconstructing and tracking humans with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 14783–14794, 2023.

[362] Nikhila Ravi, Valentin Gabeur, Yuan-Ting Hu, Ronghang Hu, Chaitanya Ryali, Tengyu Ma, Haitham Khedr, Roman Rädle, Chloe Rolland, Laura Gustafson, et al. Sam 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*, 2024.

[363] Zachary Teed and Jia Deng. Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras. *Advances in neural information processing systems*, 34:16558–16569, 2021.

[364] Jianyuan Wang, Nikita Karaev, Christian Rupprecht, and David Novotny. Vggsfm: Visual geometry grounded deep structure from motion. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 21686–21697, 2024.

[365] Taekyung Kim, Jaehoon Choi, Seokeon Choi, Dongki Jung, and Changick Kim. Just a few points are all you need for multi-view stereo: A novel semi-supervised learning method for multi-view stereo. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6178–6186, 2021.

[366] Vincent Leroy, Yohann Cabon, and Jérôme Revaud. Grounding image matching in 3d with mast3r. In *European Conference on Computer Vision*, pages 71–91. Springer, 2024.

[367] Zhoutong Zhang, Forrester Cole, Zhengqi Li, Michael Rubinstein, Noah Snavely, and William T Freeman. Structure and motion from casual videos. In *European Conference on Computer Vision*, pages 20–37. Springer, 2022.

[368] Johannes Kopf, Xuejian Rong, and Jia-Bin Huang. Robust consistent video depth estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1611–1621, 2021.

[369] Bardienus Duisterhof, Lojze Zust, Philippe Weinzaepfel, Vincent Leroy, Yohann Cabon, and Jerome Revaud. Mast3r-sfm: a fully-integrated solution for unconstrained structure-from-motion. *arXiv preprint arXiv:2409.19152*, 2024.

[370] Youtian Lin, Zuozhuo Dai, Siyu Zhu, and Yao Yao. Gaussian-flow: 4d reconstruction with dynamic 3d gaussian particle. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21136–21145, 2024.

[371] Zeyu Yang, Hongye Yang, Zijie Pan, and Li Zhang. Real-time photorealistic dynamic scene representation and rendering with 4d gaussian splatting. *arXiv preprint arXiv:2310.10642*, 2023.

[372] Yuanxing Duan, Fangyin Wei, Qiyu Dai, Yuhang He, Wenzheng Chen, and Baoquan Chen. 4d-rotor gaussian splatting: towards efficient novel view synthesis for dynamic scenes. In *ACM SIGGRAPH 2024 Conference Papers*, pages 1–11, 2024.

[373] Carl Doersch, Yi Yang, Mel Vecerik, Dilara Gokay, Ankush Gupta, Yusuf Aytar, Joao Carreira, and Andrew Zisserman. Tapir: Tracking any point with per-frame initialization and temporal refinement. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10061–10072, 2023.

[374] Jinyu Yang, Mingqi Gao, Zhe Li, Shang Gao, Fangjing Wang, and Feng Zheng. Track anything: Segment anything meets videos. *arXiv preprint arXiv:2304.11968*, 2023.

[375] Wei Jiang, Kwang Moo Yi, Golnoosh Samei, Oncel Tuzel, and Anurag Ranjan. Neuman: Neural human radiance field from a single video. In *European Conference on Computer Vision*, pages 402–418. Springer, 2022.

[376] Lea Müller, Hongsuk Choi, Anthony Zhang, Brent Yi, Jitendra Malik, and Angjoo Kanazawa. Reconstructing people, places, and cameras. *arXiv preprint arXiv:2412.17806*, 2024.

[377] Yufei Xu, Jing Zhang, Qiming Zhang, and Dacheng Tao. Vitpose: Simple vision transformer baselines for human pose estimation. *Advances in neural information processing systems*, 35:38571–38584, 2022.

[378] Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980.

[379] Jiahui Lei, Yufu Wang, Georgios Pavlakos, Lingjie Liu, and Kostas Daniilidis. Gart: Gaussian articulated template models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 19876–19887, 2024.

[380] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[381] Jerome Revaud, Jon Almazán, Rafael S Rezende, and Cesar Roberto de Souza. Learning with average precision: Training image retrieval with a listwise loss. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5107–5116, 2019.

[382] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3d: A modern library for 3d data processing. *arXiv preprint arXiv:1801.09847*, 2018.

[383] Dor Verbin, Peter Hedman, Ben Mildenhall, Todd Zickler, Jonathan T Barron, and Pratul P Srinivasan. Ref-nerf: Structured view-dependent appearance for neural radiance fields. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5481–5490. IEEE, 2022.

[384] Jonathan Young. xatlas.

[385] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[386] Zongxin Ye, Wenyu Li, Sidun Liu, Peng Qiao, and Yong Dou. Absgs: Recovering fine details in 3d gaussian splatting. In *Proceedings of the 32nd ACM International Conference on Multimedia*, pages 1053–1061, 2024.