Dynamic Scene Reconstruction and Understanding

Yu-Shiang Wong

A dissertation submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

of

University College London.

Department of Computer Science
University College London

May 2, 2023
I, Yu-Shiang Wong, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.
Abstract

Traditional approaches to 3D reconstruction have achieved remarkable progress in static scene acquisition. The acquired data serves as priors or benchmarks for many vision and graphics tasks, such as object detection and robotic navigation. Thus, obtaining interpretable and editable representations from a raw monocular RGB-D video sequence is an outstanding goal in scene understanding. However, acquiring an interpretable representation becomes significantly more challenging when a scene contains dynamic activities; for example, a moving camera, rigid object movement, and non-rigid motions. These dynamic scene elements introduce a scene factorization problem, i.e., dividing a scene into elements and jointly estimating elements’ motion and geometry. Moreover, the monocular setting brings in the problems of tracking and fusing partially occluded objects as they are scanned from one viewpoint at a time.

This thesis explores several ideas for acquiring an interpretable model in dynamic environments. Firstly, we utilize synthetic assets such as floor plans and object meshes to generate dynamic data for training and evaluation. Then, we explore the idea of learning geometry priors with an instance segmentation module, which predicts the location and grouping of indoor objects. We use the learned geometry priors to infer the occluded object geometry for tracking and reconstruction. While instance segmentation modules usually have a generalization issue, i.e., struggling to handle unknown objects, we observed that the empty space information in the background geometry is more reliable for detecting moving objects. Thus, we proposed a segmentation-by-reconstruction strategy for acquiring rigidly-moving objects and backgrounds. Finally, we present a novel neural representation to learn a factorized
scene representation, reconstructing every dynamic element. The proposed model supports both rigid and non-rigid motions without pre-trained templates. We demonstrate that our systems and representation improve the reconstruction quality on synthetic test sets and real-world scans.
Impact Statement

Reconstructing and understanding our indoor environments are fundamental to deploying artificial intelligence (AI) applications. Two successful examples are autonomous cleaning robots and 3D reconstruction using commodity hardware. Robotic vacuum cleaner integrates simultaneous localization and mapping (SLAM) and path planning technology, enabling robots to reconstruct floor plans in real-time and perform floor cleaning tasks. On the other hand, 3D reconstruction using a depth sensor such as a Kinect provides a low-cost solution to obtaining object geometry and facilitates commercial design and visual effects industries. Although these applications are now billion-dollar markets, they are mainly restricted to static settings, i.e., objects or scenes remain stationary. To unlock the next-generation applications, such as intelligent home-assistant robots or mixed reality (MR), one of the fundamental building blocks is the ability to reconstruct dynamic objects and humans, allowing robots to analyze dynamic events and assist humans in doing tedious daily tasks. This thesis investigated three research problems for acquiring dynamic indoor scene data using monocular RGB-D video input. First, Chapter 4 demonstrates a joint optimization framework to track and reconstruct rigidly-moving objects by leveraging intra-category priors. Then, Chapter 5 develops an unsupervised dense reconstruction system for acquiring rigidly-moving objects and background geometry. Finally, Chapter 6 proposes a novel neural representation to formulate a joint optimization problem for modeling rigid and non-rigid activities via volumetric rendering. The outcomes of this thesis have been published as two international conference papers at Eurographics 2021 and IEEE/CVF Conference on Computer Vision and Pattern Recognition 2021. In summary, this thesis makes new progress in
analyzing and understanding our indoor environments with large camera motions, unknown moving objects, and non-rigid activities.
Acknowledgements

First, I am grateful to my advisor Niloy Mitra for his guidance during my PhD journey. He is always encouraging and patient in teaching me how to address research problems and motivating me to achieve high-impact results. I really appreciate his help.

I would also like to extend my gratitude to Matthias Nießner for his supervision during my summer visits and collaboration. I appreciate his expert insights, feedback, and help in setting up a scanning pipeline, which saved me an enormous time. I also want to acknowledge my collaborators: Norman Muller, Angela Dai, and Changjian Li. Thank you for your contributions. I am particularly grateful for your help in the (tough) last-hour submissions.

During my PhD, I am grateful for being a member of the SmartGeometry group and for the daily help from the group members and visiting students: Paul, Aron, Robin, Tuanfeng, James, Moos, Tom, Carlo, Linjie, Xuelin, Meng, Changjian, Philipp, Eric, Preddy, Luca, Sanjeev, and Animesh. Thank you for assisting with group tasks, solving technical issues, as well as those (funny) daily chats in the office. Thanks to Xuelin, Changjian, and other members of the group for their help with 3D scanning, which took several hours.

Also, I would like to thank my London friends, who remind me that London is a wonderful city. Chanawee and Yu-Hsuan, thanks for connecting me to funny events in London and sharing living tips with me. NhuY, many thanks for sharing delicious receipts and bubble teas with me and for your help during the Covid-19 pandemic. It would have been a stressful time without you. I also want to express my appreciation to my friends in Taiwan: Bo-ren, Bo-cheng, Lin-Lin, Rih-ding,
Acknowledgements

Yang-sheng, Jia-wei, and Po-an. Thank you all for your friendship. It is always fun to meet you guys.

Finally, I would like to express my deepest gratitude to my parents and sister for their unconditional support, love, and understanding. Thank you for supporting me in going through these challenging moments, reminding me about healthy lifestyles, and being proud of my achievements. This thesis is dedicated to you.
Publications

Note that our work presented in this thesis has been published as two conference papers. Their references are listed in the following.

• Chapters 3 and 5:

• Chapters 4:

• Chapter 6:
## Contents

1 Introduction 16

2 Related Work 22
   2.1 Dynamic Indoor Scene Reconstruction 22
      2.1.1 Background Reconstruction 22
      2.1.2 Rigid Object Reconstruction 23
      2.1.3 Non-rigid Object Reconstruction 24
      2.1.4 Rigid Objects and Background Reconstruction 25
      2.1.5 Full Dynamic Scene Reconstruction 26
   2.2 Surface Representation 27
      2.2.1 Classical Models 27
      2.2.2 Learning-Based Models 28
      2.2.3 Differential Rendering 29
   2.3 Synthetic Data for Learning 30

3 Dynamic Indoor Scene Dataset 31
   3.1 Introduction 31
   3.2 Data Generation 32
   3.3 Dataset Analysis 34
   3.4 Evaluation Metrics 34
      3.4.1 Reconstruction 34
      3.4.2 Tracking 37
   3.5 Discussion and Limitations 38
## 4 Supervised Dynamic Objects Reconstruction

4.1 Introduction ................................................. 40

4.2 Method ....................................................... 41

4.2.1 Object Detection ........................................... 41

4.2.2 Object Completion and Correspondences ................. 43

4.2.3 Differentiable Pose Optimization ........................ 44

4.2.4 Object Tracking and Reconstruction ...................... 45

4.2.5 Implementation Details ................................. 45

4.3 Evaluations and Results .................................... 46

4.3.1 Comparison to State of the Art ......................... 47

4.3.2 Does Object Completion Help Tracking? ................. 49

4.3.3 Real-world Dynamic RGB-D Sequences ................. 49

4.3.4 Detection and Completion Evaluation ................... 49

4.4 Discussion and Limitations ................................ 51

## 5 Unsupervised Dynamic Scene Reconstruction

5.1 Introduction .................................................. 53

5.2 Method ....................................................... 55

5.2.1 Overview .................................................. 55

5.2.2 Background Reconstruction and Camera Estimation .... 56

5.2.3 Asynchronous Foreground Reconstruction ............... 58

5.2.4 Post-Processing ........................................... 61

5.2.5 Mesh Extraction ........................................... 62

5.3 Evaluations and Results .................................... 62

5.3.1 Computational Time and Implementation Details ....... 62

5.3.2 Evaluation on DYNSYNTH Test Set ..................... 63

5.3.3 Evaluation on Real-world Data ........................ 65

5.3.4 Evaluation on CoFusion Dataset ......................... 68

5.4 Discussion and Limitations ................................ 68
## Contents

### 6 Neural Dynamic Scene Representation 73

- 6.1 Introduction 73
- 6.2 Image Formation Model 75
  - 6.2.1 Volume Rendering 75
  - 6.2.2 Volume Rendering with Implicit Surface 76
  - 6.2.3 Attributes Rendering 77
  - 6.2.4 Volume Rendering with Factored Neural Representation 78
- 6.3 Method 78
  - 6.3.1 Initialization 79
  - 6.3.2 Joint Optimization 79
- 6.4 Evaluation 82
  - 6.4.1 Datasets 82
  - 6.4.2 Comparison 85
  - 6.4.3 Evaluation Metrics 85
  - 6.4.4 Quantitative Evaluation 85
  - 6.4.5 Qualitative Evaluation 87
  - 6.4.6 Ablation study 88
  - 6.4.7 Applications 88
  - 6.4.8 Memory and Implementation Details 91
- 6.5 Discussion and Limitations 91

### 7 Conclusions 94

- 7.1 Summary 94
- 7.2 Future Work 95
- 7.3 Remark 97

### Appendices

- A A Glossary of the Terminology 98
- B RigidFusion’s Experiment Details 99

### Bibliography 101
List of Figures

1.1  Cyclic dependency in dynamic scene reconstruction. . . . . . . . . 17
1.2  The occlusion problem in tracking moving objects. . . . . . . . . 18
3.1  Data generation pipeline and examples. . . . . . . . . . . . . . . . 32
3.2  Example scenes in our DYN SYNTH dataset. . . . . . . . . . . . . 35
3.3  A comparison of the occlusion settings.. . . . . . . . . . . . . . . 36
3.4  Assessing reconstruction quality evaluation and types of errors. . . 37
3.5  Assessing tracking performance evaluation and types of errors. . . 38
4.1  The illustration of our joint optimization formulation. . . . . . . . 41
4.2  Overview of our network architecture. . . . . . . . . . . . . . . . 42
4.3  Our network architecture. . . . . . . . . . . . . . . . . . . . . . . 46
4.4  Qualitative comparison on DYN SYNTH. . . . . . . . . . . . . . 47
4.5  Qualitative comparison to state of the art on real-world sequences. . 50
5.1  RigidFusion’s 4D reconstruction results. . . . . . . . . . . . . . . 54
5.2  RigidFusion’s input buffer. . . . . . . . . . . . . . . . . . . . . . 55
5.3  RigidFusion’s system diagram. . . . . . . . . . . . . . . . . . . . . 56
5.4  TSDF illustration. . . . . . . . . . . . . . . . . . . . . . . . . . . 57
5.5  Qualitative evaluation on DYN SYNTH. . . . . . . . . . . . . . 63
5.6  Qualitative evaluation on real-world data. . . . . . . . . . . . . . 67
6.1  Factored neural representation. . . . . . . . . . . . . . . . . . . . 74
6.2  Rendering neural factored representation. . . . . . . . . . . . . . 76
6.3  Pipeline. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 77
List of Figures

6.4 Dataset. ................................................................. 81
6.5 Comparisons on our synthetic dataset. .............................. 83
6.6 Comparisons of the object reconstruction on our synthetic dataset. 84
6.7 Comparisons of scene reconstruction on the BEHAVE dataset. .... 86
6.8 Reconstruction and applications on our synthetic dataset. ......... 89
6.9 Reconstruction and applications on the BEHAVE dataset. ......... 90
6.10 Our network architecture. ............................................ 92
# List of Tables

1.1 Indoor RGB-D dataset. ........................................ 19  
1.2 Comparison of the dense reconstruction methods using monocular RGB-D input. ............................... 20  

4.1 Evaluation of MOTA on DYNSTH. .......................... 48  
4.2 Evaluation of object pose estimation on individual RGB-D frames. ................................................. 49  
4.3 Evaluation of 3D detection and instance completion. ................................................................. 51  

5.1 Quantitative evaluation of the background reconstruction and tracking on DYNSTH. ............................ 63  
5.2 Quantitative evaluation of the foreground reconstruction and tracking on DYNSTH. ............................ 64  
5.3 The analysis of our real-world examples. .................. 69  
5.4 The evaluation of moving object detection. ............... 69  
5.5 Background tracking evaluation on TUM RGB-D dataset. ............................................................... 70  
5.6 Foreground tracking evaluation on CoFusion’s dataset. ................................................................. 70  

6.1 Reconstruction error on our synthetic dataset. ........... 86  
6.2 Reconstruction error on BEHAVE using the training camera. ......................................................... 87  
6.3 Ablation study on our synthetic dataset. ................. 88  

A.1 The terminology used in this thesis. ....................... 98  

B.1 RidgidFusion’s experimental setting ....................... 99  
B.2 CoFusion’s experimental setting ......................... 100  
B.3 MaskFusion’s experimental setting ....................... 100
Chapter 1

Introduction

Capturing 3D scene geometry is a long-standing challenge in the vision and graphics fields. Among the various capture settings in the literature (such as a small scene equipped with multiple pre-calibrated RGB cameras [1, 2] or an expensive laser scanner [3]) scanning scene geometry with a depth sensor has become a popular solution as it is ad hoc and inexpensive. While a depth sensor provides the 3D information (i.e., the distance from a point to the camera), acquiring accurate 3D scene geometry from monocular RGB-D input is still challenging because of the missing information caused by occlusion and noisy depth measurements.

Several robust solutions [4, 5, 6, 7, 8, 9, 10, 11] have been developed for capturing static scenes with monocular RGB-D input, and they have been successfully applied to capture a large-scale indoor dataset [11, 12]. However, there are limited options for capturing dynamic scenes, and available solutions focus on the simplified problem settings by assuming the existence of background geometry through the initialization stage [13, 14] or restricting capturing to a single dynamic object recorded by a stationary camera [15, 16, 17, 18].

This is rather surprising since our surroundings are mostly dynamic due to objects moving around during our regular interactions; for instance, a person moving a box, table, or chair. The ability to faithfully record indoor environments with moving objects using commodity hardware (for example, a smartphone with a mounted depth sensor) would open up many possibilities to capture our environments in their natural settings and, in turn, provide rich data priors for several vision
Figure 1.1: Cyclic dependency in dynamic scene reconstruction. Solving dynamic scene reconstruction involves breaking the cyclic dependency between detecting the moving objects, e.g., the dynamic chair among the other static chairs, and estimating the movement of each point.

The primary challenge in capturing dynamic scenes is to solve a complex solution space involving factorizing individual dynamic elements from RGB-D videos, associating dynamic elements between frames, classifying their motion type (i.e., rigid or non-rigid movement), establishing the corresponding 3D points between frames, estimating their motions, and produce 4D reconstructions in the form of fused canonical models that aggregates multiple partial scans at different time steps to the shared world-space geometry for each dynamic element. As Figure 1.1 shows, in the absence of prior knowledge about object positions, shapes, or background geometry, dynamic scene reconstruction naturally leads to a cyclic dependency on segmenting dynamic elements and tracking their movement because motion segmentation (i.e., partitioning pixels into dynamic elements according to movement) requires accurate tracking, and object tracking requires precise segmentation.

The second challenge is estimating motion from partial scans, as shown in Figure 1.2, where two input frames contain different parts of scene geometry. This is also known as the occlusion issue in the surface reconstruction literature due to the camera view change and scene clutter, which prohibit finding correct correspondences between frames, i.e., the same points observed at different time steps. A common solution [11, 19, 20] for this issue is to establish a set of robust correspondences by employing carefully designed high-dimension features such as SIFT [21]
Figure 1.2: The occlusion problem in tracking moving objects. Object movement and the monocular input setting make object tracking difficult. Here the two input chair segments have little overlapping, which leads to a challenging motion estimation problem.

or FPFH [19] to filter outlier correspondences caused by view change. However, this is harder in dynamic scenes, as moving objects are small, and their movement may lead to less overlapping between frames.

The third challenge is formulating a joint optimization framework for solving the reconstruction of multiple dynamic elements, including rigid objects, non-rigid objects, and backgrounds. While this multi-object optimization requires instance segmentation, i.e., the pixel grouping according to each individual object on the image, instance segmentation is usually solved in a pre-processing step. Joint optimization provides a chance to also optimize instance segmentation with other variables, such as geometry and appearances. Hence, it can improve overall performance. However, it is non-trivial due to the explosion of the variables, especially for non-rigid motion. Non-rigid motion is more complex than rigid motion because every pixel (point) has a different amount of movement, significantly increasing the number of variables. Therefore, non-rigid and rigid objects are usually discussed separately in the literature.

Further, the difficulty of acquiring 4D data leads to the absence of a large-scale training dataset. This challenge prohibits conducting quantitative evaluations or leveraging supervised methods that use prior knowledge to solve 4D reconstruction from data. Although several dynamic datasets have been developed, as summarized in Table 1.1, they are targeting different sub-tasks, such as background reconstruction.
Table 1.1: Indoor RGB-D dataset. Dynamic RGB-D data has received more focus in recent years. However, the scale, scene sizes, and variety of the developed dynamic datasets are still lower than the state-of-the-art static datasets.

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Scale</th>
<th>Year</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUN-RGBD [26]</td>
<td>Static</td>
<td>10,000 images</td>
<td>2013</td>
<td>static images</td>
</tr>
<tr>
<td>S3DIS [27]</td>
<td>Static</td>
<td>271 rooms</td>
<td>2016</td>
<td>static rooms</td>
</tr>
<tr>
<td>Matterport3D [28]</td>
<td>Static</td>
<td>10,800 panoramic views</td>
<td>2017</td>
<td>static rooms</td>
</tr>
<tr>
<td>ScanNet [12]</td>
<td>Static</td>
<td>1,500 rooms</td>
<td>2018</td>
<td>static rooms</td>
</tr>
<tr>
<td>Bonn [22]</td>
<td>Dynamic</td>
<td>25 sequences</td>
<td>2019</td>
<td>single scene, 4 subjects with 3 objects</td>
</tr>
<tr>
<td>DeepDeform [23]</td>
<td>Dynamic</td>
<td>400 sequences</td>
<td>2020</td>
<td>static cameras, each scene contains 1-2 deformable objects</td>
</tr>
<tr>
<td>IPhone [29]</td>
<td>Dynamic</td>
<td>14 sequences</td>
<td>2022</td>
<td>local motion</td>
</tr>
<tr>
<td>BeHave [25]</td>
<td>Dynamic</td>
<td>321 sequences</td>
<td>2022</td>
<td>static cameras, 8 subjects interacting with 20 objects</td>
</tr>
<tr>
<td>HOI4D [30]</td>
<td>Dynamic</td>
<td>4,000 sequences</td>
<td>2022</td>
<td>egocentric cameras, table-top activities</td>
</tr>
</tbody>
</table>

in dynamic environments [22], non-rigid registration [23], or robotic simulation and navigation [24]. The most relevant dataset is BEHAVE [25], which uses four static cameras to record human-object interactions and provide both motion estimation and fused canonical geometry. However, this dataset was released only recently, with limited variety, i.e., they recorded dynamic events with twenty common objects in five indoor scenes.

In Chapter 3, we will explore synthesizing virtual scanning containing dynamic events, building upon the rapid progress on synthetic 3D assets in recent years [40, 41, 42]. We aim to create ground truth data where we virtually move objects around by simulating real-world scanning scenarios in dynamic environments with rigidly moving objects and utilize the generated data for training and evaluation. Therefore, we developed a data generation framework to synthesize a large-scale dynamic dataset containing rigidly-moving objects, non-static camera trajectories, and an evaluation protocol for assessing the quality of 4D reconstruction results.

Chapter 4 will present a supervised method for reconstructing dynamic rigid objects by leveraging our new dataset. We break the cyclic dependency between segmentation and tracking, and tackle the occlusion issue by employing novel
Table 1.2: Comparison of the dense reconstruction methods using monocular RGB-D input. Rect.: reconstruction. FG: foreground. N: number of detected instances. N+1: incrementally solve a new foreground object. 1/N: only solve a new foreground among the N simultaneously moved objects.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>KinectFusion [6]</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>0</td>
<td>30Hz</td>
</tr>
<tr>
<td>StaticFusion [31]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>0</td>
<td>30Hz</td>
</tr>
<tr>
<td>ReFusion [22]</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>0</td>
<td>N.A.</td>
</tr>
<tr>
<td>DynaSLAM [32]</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>0</td>
<td>2Hz</td>
</tr>
<tr>
<td>IMap [33]</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>0</td>
<td>2Hz</td>
</tr>
<tr>
<td>NiceSLAM [34]</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>0</td>
<td>6Hz</td>
</tr>
<tr>
<td>Neural RGB-D [35]</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>0</td>
<td>offline</td>
</tr>
<tr>
<td>Dynamicfusion [15]</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>1</td>
<td>30Hz</td>
</tr>
<tr>
<td>VolumeDeform [16]</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1</td>
<td>30Hz</td>
</tr>
<tr>
<td>KillingFusion [18]</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1</td>
<td>30Hz</td>
</tr>
<tr>
<td>NDR [36]</td>
<td>x</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>1</td>
<td>offline</td>
</tr>
<tr>
<td>CoFusion [14]</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>N+1</td>
<td>12Hz</td>
</tr>
<tr>
<td>MaskFusion [37]</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>N</td>
<td>5Hz</td>
</tr>
<tr>
<td>MIDFusion [38]</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>N</td>
<td>2Hz</td>
</tr>
<tr>
<td>EMFusion [39]</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>N</td>
<td>4Hz</td>
</tr>
<tr>
<td>ComObjTrackNet (Chapter 4)</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>N</td>
<td>1Hz</td>
</tr>
<tr>
<td>FactoredNeRF (Chapter 6)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>N</td>
<td>offline</td>
</tr>
</tbody>
</table>

Object priors such as geometry and instance segmentation are useful but may not always exist. A pre-trained segmentation network may not recognize an object if it is not in the training distribution. This is known as the generalization problem of deep learning methods. In Chapter 5, we will present RigidFusion: a 4D reconstruction system aimed at reconstructing rigidly-moving objects without using object priors. We propose a novel segmentation-by-reconstruction step to handle the cyclic dependency. Our system can simultaneously build a background model and detect unseen moving objects in a highly dynamic environment, i.e., with non-
static camera movement, which is particularly challenging for the state-of-the-art methods [14, 37, 38].

In Chapter 6, we move one step forward to address the full dynamic scene reconstruction problem using monocular RGB-D input. We propose a novel learned representation, FactoredNeRF, using volume rendering to reconstruct rigid, non-rigid, and background geometry jointly. We demonstrated that, by having a noisy segmentation and axis-aligned bounding boxes initialization, our factorized neural representation could jointly reconstruct both rigid and non-rigid objects without using templates (such as SMPL [43]). Our learned representation supports several scene editing operations, such as novel view rendering, object removal, and pose editing, without requiring re-training the network at test time.

In summary, we compare the related problem statements and highlight our contribution in Table 1.2. The thesis is organized as follows: Chapter 2 presents a literature review; Chapter 3 introduces our data generation tool; Chapter 4 presents our completion and tracking network; Chapter 5 presents RigidFusion; and Chapter 6 presents FactoredNeRF. The conclusion and future work are given in Chapter 7. A glossary of the terminology used in this thesis is included in Appendix A.
Chapter 2

Related Work

This chapter reviews the literature on dynamic indoor scene reconstruction, ranging from works on acquiring a single moving object to a full dynamic scene, providing the background for Chapters 4 to 6. Then it introduces the literature on surface representation related to Chapter 6. Finally, this chapter discusses synthetic datasets relevant to Chapter 3.

2.1 Dynamic Indoor Scene Reconstruction

2.1.1 Background Reconstruction

Generalizing static scene reconstruction methods, such as KinectFusion [6], to dynamic scenes requires tackling a significant amount of background outliers (i.e., moving objects). These dynamic pixels or points must be detected and segmented to reconstruct the background. Thus, this line of research introduces an outlier detection step in the reconstruction pipeline based on segmentation techniques, which identifies the region (pixels or points on the input RGB-D images) that belongs to moving objects. For example, Keller et al. [44] employ hierarchical region-growing segmentation to remove outliers by maintaining a confidence score. On the other hand, Scona et al. [31] propose a non-linear optimization to jointly solve tracking and binary segmentation, i.e., foreground or background. Palazzolo et al. [22] propose a flood-fill algorithm to grow outlier masks iteratively. These methods, however, do not track and reconstruct foreground objects. In Chapter 5, we propose a novel segmentation-by-reconstruction step to detect dynamic objects and
reconstruct background and foreground objects.

### 2.1.2 Rigid Object Reconstruction

Reconstructing a rigid object is similar to background reconstruction because of the rigidity assumption that the object motion is modeled by rigid transformations, which preserves Euclidean distance under transformations. If the object segmentation is known, this problem is reduced to static reconstruction, and static methods such as KinectFusion [6] can be applied to reconstruct object geometry. However, tracking a rigidly moving object is generally harder than tracking a camera (background) due to occlusions, where parts of the object geometry cannot be seen from the camera view. In addition, occlusions are hard to be bypassed under a monocular setting. Previous works handle this issue by utilizing 3D shape priors and matching input points with template meshes [45, 46, 47, 48, 49] or employing sparse feature descriptors [50, 51, 52], which create a set of high-dimensional features to match points under rotation, translation, scale, and illumination change. The limitations of these methods are still occlusion and the requirements of templates.

More recently, deep learning methods have been adapted to object tracking. PoseCNN [53] is one of the pioneering works using a convolutional neural network on color input to predict object centers, semantic labels, and object poses. PointFusion [54] and DenseFusion [55] add depth cues by concatenating image features and point features using a convolutional neural network and PointNet [56]. These works improved the tracking accuracy but are restricted to the training objects.

To improve the generalization capability to deal with unseen objects, Wang et al. [57] proposed Normalized Object Coordinate Space (NOCS), which learns category-level object priors using the modern detection framework [58]. This approach removes the requirement of an exact object template and relaxes template-based object tracking to category-level object tracking. Moreover, 6-Pack [59] integrates DenseFusion [55] with KeypointNet [60] to predict the keypoints shared between each frame and achieve higher tracking accuracy. The main limitations of 6-Pack and DenseFusion are the requirement of the ground-truth poses and bounding boxes in the first frame, which limits its application. In Chapter 4, we employ the
off-the-shelf instance detection networks [58, 61] as a backbone and propose a novel network architecture to learn category-level geometry priors and correspondences using NOCS [57] to achieve robust object tracking.

2.1.3 Non-rigid Object Reconstruction

Many dynamic events contain non-rigid motions, for example, a human lifting a bag. The tricky parts of non-rigid object reconstruction are that the object geometry may change during acquisition, leading to an ambiguity between new information and topology change, and every point may deform differently, leading to a large number of variables. As this is a highly ill-posed problem, non-rigid object reconstruction usually assumes the camera is static, 2D object masks are known, and applying regularizations to constraint deformation.

Two common deformation regularizers are smooth deformation and local rigidity [62, 63]. Smooth deformation constrains the neighboring points to have similar motions by utilizing input connectivity, such as a mesh or a deformation graph. On the other hand, local rigidity forces the motion to be as rigid as possible by approximating a rigid transformation at each node. These two regularizations are widely adopted in non-rigid object reconstruction literature [15, 16, 18, 64]. A recent approach is to utilize the inductive bias of a multi-layer perceptron (MLP) network to regularize deformation. Research on generative modeling [65, 66] demonstrates that an MLP network tends to reconstruct a smooth latent space. This property has been applied to shape deformation [35, 67], enhancing the reconstruction quality without as-rigid-as-possible regularization [35].

Similar to the rigid tracking case in the previous section, the occlusion issue exists when using monocular video as input, making the correspondence search challenging. Previous works address this problem using sparse feature matching [16], dense tracking [15, 18, 68], learning-based feature matching [23, 69, 70], and optimizing MLP networks with object pose initialization [36]. Not surprisingly, templates or category-level shape priors are also useful to alleviate the occlusion problem. This line of work includes using hand-crafted parametric models [43, 71, 72, 73, 74, 75] and learning-based templates [76, 77, 78, 79]. In Chapter 6, we
also employ the MLP network as deformation regularization and propose a novel joint optimization framework to reconstruct both non-rigid objects, rigid objects, and background geometry together.

2.1.4 Rigid Objects and Background Reconstruction

Dense reconstruction in dynamic indoor scenes with rigid motion assumption is still a relatively new research topic. The main challenge is simultaneously handling the uncertainty of detecting unknown moving objects and tracking static background under a moving camera. Previous works can be classified into two types: using motion residuals and leveraging data priors, according to the signals used for detecting moving objects.

Lu and Gabe [13] proposed detecting moving objects in a frame-to-frame fashion by examining motion outliers in the background model. Multiple moving objects are extracted by finding the disjoint region on the outlier mask. CoFusion [14] follows a similar concept and proposes an efficient framework using Conditional Random Field (CRF) [80] to solve motion segmentation in real time. Note that the above methods [13, 14] only focus on small indoor scenes (e.g., 1-2 sqm) and are restricted to slowly moved camera motion as reported in [31, 81]. With the rapid progress in object detection and segmentation (see the recent segmentation benchmark [82, 12]), semantic priors have become a popular solution because segmentation networks can learn data prior from the existing large-scale image datasets. MaskFusion [37] is one of the recent works that propose a real-time pipeline to integrate surfel tracking [83] with an instance segmentation network [58]. Although segmentation prior [58] can provide object masks, it suffers from a low recall issue and slow inference time, e.g., 2Hz, due to the deep network architecture. To alleviate the low recall issue, MID-Fusion [38] and EMFusion [39] accumulate segmentation prediction using volumetric grids [4] and back-project object masks from the integrated object models. Although these works improve the robustness, the moving object detection problem is not completely solved due to instance segmentation being a rather weak initialization for motion segmentation in cluttered scenes, e.g., a scene with many indoor objects. In Chapter 5, we propose a novel dynamic
scene reconstruction system without using common object priors such as template meshes or image segmentation networks to enhance the generalization ability of object detection and segmentation. We applied our system to several challenging examples that scene contains large camera motion and rigidly moving objects.

2.1.5 Full Dynamic Scene Reconstruction

Full dynamic scene reconstruction without rigidity assumption is a more general problem statement. However, the existence of non-rigid motion creates extra complexity in segmentation and optimization. The early efforts on this problem can be categorized into three types. First, decoupling the handling of each scene element uses semantic priors [37, 38, 39] or motion segmentation [14, 84, 85]. These methods optimize individual objects separately, and the object segmentation is not jointly optimized. Second, capturing dynamic effects by over-fitting to 4D space-time volume [86, 87, 88, 89] uses a global neural radiance field (NeRF) [90]. These methods are restricted to local motion and do not provide the ability to extract and edit individual scene elements. Third, employing two (or three) neural networks for modeling background and foreground [90, 91, 92, 93] (and Actor [93]) from monocular color input. These works examine unsupervised segmentation [90, 91, 93] or self-supervised [92] segmentation, but they do not decompose each foreground object and recover object trajectories.

In other efforts, researchers have separately investigated the effect of using multi-view input from a (static) camera array. Multi-view setup with static cameras provides a good regularization for occlusion and demonstrates promising results [94, 95]. This can be further integrated with a parametric human template [43, 71] and enhance reconstruction quality on object-human interaction [96] or multiple human activities [97]. However, setting up multiple calibrated cameras is non-trivial and therefore restricts its applications.

In Chapter 6, we present FactoredNeRF: a learned dynamic scene representation that can be extracted, without requiring pre-training and parametric templates, simultaneously with object tracking and segmentation. Further, unlike many other multi-view pipelines [94, 95, 96, 97], we dynamically aggregate information across
views to recover from occlusion under a moving camera setting. Once trained, our representation can not only be viewed from novel camera paths but also used to make changes to object trajectories and placements.

2.2 Surface Representation

Efficiently representing surface geometry is important for indoor reconstruction. In this section, we summarize the literature from classical to learning-based approaches and introduce differential rendering for optimizing learning-based models, which are relevant to Chapter 6.

2.2.1 Classical Models

Truncated signed distance fields (TSDF) and surface elements (surfel) are two widely used representations in the literature on surface reconstruction. The pioneering work by Curless and Levoy [4] introduces the idea of using a volumetric grid to represent geometry as a level-set function, named a truncated signed distance field (TSDF). Each voxel in the volumetric grid is associated with a truncated scalar and a weight. The scalar represents a signed distance value from a voxel to the nearest surface, and the truncation ignores the far-surface regions as free-space information is less relevant to the surface topology. The weight is used to implement moving averages for efficiency. This implicit representation can model complex geometry and suits for parallel processing and real-time systems [6, 7, 11, 98, 83] since each voxel can be independently updated.

On the other hand, the surfel representation proposed by Pfister et al. [99] is designed for memory-efficient graphic rendering and latterly applied to indoor reconstruction [83, 44] due to its lightweight model size. A surfel consists of a point (coordinates), a radius, and an orientation vector (normal), acting as a linear approximation of the local geometry at the point position. The quality of the surfel reconstruction is controlled by the density of sampled points, providing a speed-quality trade-off, which can be linked to the \( r \)-sample theory [100, 101] on surface meshing.

In Chapters 4 and 5, we employ the TSDF representation to model objects and
2.2. Surface Representation

scene geometry, and we utilize TSDF to model geometry prior (in Chapter 4) as well as detect moving objects (in Chapter 5).

2.2.2 Learning-Based Models

The recent introduction of the implicit neural representation [65, 66, 102] has resulted in an explosion of works to overfit a single object or to encode object collections. The core idea is to employ several multi-layer perceptions (MLPs), taking inputs as coordinates and regressing a signed distance value. Compared to the dense grid models [4], neural models bypass the cubic memory complexity and significantly compress the model size. However, this formulation [65, 66, 102] tends to miss high-frequency details and have slow convergence.

Several follow-up works have been proposed to address the expressiveness issue of the MLPs models. Mildenhall et al. [103] and Tancik et al. [104] revisit feature lifting techniques from the machine learning literature [105] and lift input coordinates to a high-dimensional space using Fourier features. Sitzmann et al. [106] propose a periodic activation function with a carefully designed weight initialization scheme to capture high-frequency details. Coarse-to-fine approaches [107, 108] tackle high-frequency signals using a two-level architecture, where the first level performs coarse reconstruction, and the second level refines the coarse output by predicting a displacement.

Researchers have also proposed hybrid (i.e., explicit and implicit) representations by adapting classic representations such as a volumetric grid, providing fast convergence and high quality. Local grid models decompose the space into small voxels attached with feature vectors to learn the local structure [109, 110] or local frequencies [111] using a shallow neural network. Hierarchical models [112, 113] propose to use an octree to perform dynamic decomposition and preserve geometry details. However, the re-employed grid structure brings back the memory consumption issue. Point-based models aim to combine the simplicity of point data structure with implicit representation by using a differentiable Poisson solver [114] or optimizing point features to learn an implicit surface [115].

In Chapter 6, we employ multiple MLPs to represent both objects and back-
ground geometry and propose a novel joint optimization framework to reconstruct dynamic scenes.

### 2.2.3 Differential Rendering

Differential rendering allows optimizing implicit neural representations through a reconstruction loss that supervises the predicted color or depth using the input RGB-D frames, which is a useful tool for learning surface reconstruction. Two common approaches are ray tracing and volume rendering.

Ray tracing accounts for explicit surface intersections and defines the training loss on the surface. Ideally, a surface loss can be back-propagated through the whole tracing process, but it is impractical due to the expensive cost of the Jacobian matrix. Therefore, local approximation tricks are usually employed. This includes using implicit differentiation [116, 117] to calculate the gradient on the surface with respect to the input coordinates, max pooling the intersected anchor points to aggregate gradients [118], or unfolding the last sphere tracing step [119, 120, 121]. However, for complex geometry, these methods suffer from hard-to-propagate local gradients and weight initialization.

In contrast, volume rendering [122], leading to Neural Radiance Fields (NeRF) [103], integrates density and color samples along rays by modeling a radiance field and employs a coarse-to-fine sampling scheme to focus on surface density, without explicitly representing the underlying geometry. While NeRF can avoid the local approximation step, its limitation on surface reconstruction is the noisy and inaccurate learned implicit geometry. Follow-up works inject inductive bias through a dual representation as a signed distance field [123, 124] or occupancy [124] to define point density based on the implicit surface representations, constraining the network to reconstruct geometry.

In Chapter 6, we utilize volume rendering and propose an efficient joint optimization framework to learn multiple objects’ geometry and motion through the reconstruction loss.
2.3 Synthetic Data for Learning

The availability of depth sensors enables the development of large indoor datasets. State-of-the-art indoor datasets, such as ScanNet [12] and the Stanford 3D indoor scene dataset [27], provide rich data prior and enable the usage of deep learning methods, resulting in significant performance improvement on several indoor understanding tasks, such as object detection [12, 125], object retrieval [42], and room layout estimation [12, 126].

However, the ground truth of the collected real-world data, i.e., segmentation labels, object geometry, and camera trajectories, is expensive to create and hard to annotate accurately. For example, in order to obtain ground truth geometry, the ROBOTHOR dataset [127] sets up a room that uses a set of pre-captured objects to collect real-world indoor data. Although this strategy simplifies annotation efforts, it only has a limited variety.

Thus, synthesizing virtual data becomes a more affordable alternative. One successful example is the FlyingChairs dataset [128], which generates 22,872 image pairs and their ground-truth optical flow by randomly placing chair models with background textures. Surprisingly, despite the synthetic-to-real gap, FlowNet [128] demonstrates that learning from synthetic data can obtain certain generalization capabilities in nature scenes on the optical flow task. Similarly, several synthetic indoor simulators [24, 129, 130, 131] have been developed in recent years, which primarily benefit research on embodied artificial intelligence and smart indoor robots, such as robot navigation [24, 129, 131, 130, 132], and object rearrangement [24, 130, 127, 131, 132, 133].

Based on the above observations, in Chapter 3, we developed a data generation toolkit. We employed it to generate the first large-scale dataset for dynamic indoor scene reconstruction and used it for both training and evaluation in Chapters 4 and 5. In addition, we created several synthetic examples for the evaluation purpose in Chapter 6.
Chapter 3

Dynamic Indoor Scene Dataset

3.1 Introduction

Collecting a large-scale indoor dataset containing dynamic events is still an open research question, which restricts the usage of deep learning methods and establishing benchmarks. An alternative choice is to employ synthetic data to learn priors [128, 134] or conduct evaluations [129, 24] by leveraging the publicly available 3D indoor assets [40, 41].

This chapter explores developing an automatic pipeline to synthesize virtual indoor scans containing dynamic events. The synthesized scans would serve as the training and testing data for the succeeding chapters. To this end, we derive our dataset from a set of publicly-sourced synthetic assets, including room layouts and the CAD models of common indoor furniture, and simulate simple dynamic events in that indoor objects are moved in a room observed by a virtual moving camera. Through our toolkit, we generated rich synthetic data without manual annotations, including RGB-D images, semantic labels, instance IDs, ground truth trajectories, and complete scene geometry. One challenge here is synthesizing human actions, e.g., a person picks an object. For simplicity, we only generated rigid object motion because Chapters 3 and 4 focus only on rigid objects, and generating human motion conditional on the object motion is still an open research problem.

Two crucial factors, occlusions and geometry features related to the limitation of tracking and reconstruction [135], are carefully controlled during synthesizing, both
3.2. **Data Generation**

We developed an automatic data synthesis framework to generate motion in indoor scenes. Our synthesis pipeline starts by randomly sampling a set of floorplans, rooms, and indoor objects from publicly-sourced indoor assets. Next, we randomly synthesize object motions and a virtual moving camera looking at the dynamic objects to create the motion configuration in the selected rooms. Then, we use OpenGL to render output, including color images, depth images, instance ID maps, and semantic label maps using NYU40 classes [137]. In Figure 3.1 and the following, we elaborate on each step for synthesizing a sequence.

---

**Figure 3.1: Data generation pipeline and examples.** (a) Room sampling. We randomly select a room having sufficient empty space (right) and avoid the cluttered one (left) using the rendered occupancy maps. (b) Object sampling. We favor the object with sufficient normal variation (right) over the object with flat geometry (left). (c) Goal position sampling. We perform an occlusion test and identify potential goal positions (blue cross). (d) Motion generation. We synthesize a motion sequence using the path planning library [136]. (e) Camera motion synthesis. We add camera motion by generating random transformations. We calculate a visibility score to force the randomly generated cameras to focus on the moving object. This score is calculated by rendering the occluded (middle) and unoccluded (right) instance mask of the moving object using the instance ID map (left).

---

described in the next section. Some challenging settings, such as illumination change and motion blur, are excluded by our data generation pipeline, and we considered such areas for future research. Our generated dataset is analyzed in Section 3.3, and the evaluation metrics for assessing tracking performance and reconstruction quality are introduced in Section 3.4.
3.2. Data Generation

1. **Room sampling:** we sample a room with enough empty space to add object motion. We measure an empty space ratio as the amount of space left on the 2D floor mask, and we reject overly cluttered or empty rooms.

2. **Object sampling:** we randomly sample dynamic objects in the room and avoid featureless objects, such as a flat table. The featureless objects are identified by checking their normal variation using the discretized normal directions.

3. **Goal position sampling:** we sample collision-free positions by moving the sampled objects in the room. The object-to-object collision is detected by performing an intersection check using primitive shapes, i.e., cuboids.

4. **Motion generation:** based on the original object position and the goal position, we generate an object trajectory while avoiding collisions. This collision-free trajectory is solved using a path planner [136]. We restrict movements to 2D rigid rotation and translation on the XZ plane (floor plane) in order to create a more reasonable motion (see Figure 3.1 (c)).

5. **Camera motion synthesis:** we aim to generate realistic camera motion as a person using Kinect to record dynamic events. First, we sample several camera positions in a room while requiring cameras to look at the dynamic objects. We remove heavily occluded candidates by estimating a visibility score of the target object in both 2D (pixels) and 3D (points) domains through the rendered segmentation images and the target’s bounding box. Second, we interpolate camera motion between the camera positions. We inject random motion by randomly translating and rotating the XZ direction of a camera.

6. **Render output:** We render color, depth, semantic labels, and instance IDs using OpenGL. Each synthesized sequence has ground truth trajectories and complete meshes.

7. **Post-process (manual step):** We employ a manual post-process step to remove bad sequences that contains undetected occlusions or texture flickering due to the insufficient accuracy in the OpenGL depth buffer.
3.3 Dataset Analysis

Using our framework, we generate two large datasets and serve as training and test sets in Chapters 4 and 5. For simplicity, we named our generated dataset DYN SYNTH in the later chapters. Two examples are provided in Figure 3.2.

For learning geometry priors for tracking in Chapter 4, we increase the occlusion ratio to synthesize partially occluded motion sequences. A comparison of the two occlusion levels is shown in Figure 3.3. In total, we generated 3300 sequences of indoor scenes comprising 97,626 frames and split them into training, validation, and testing set (containing 2900, 300, and 100 sequences, respectively). The average length of camera movement is 8.3 m, and the object trajectory length is 7 m.

For the evaluation purpose in Chapter 5, we generated a training set for fine-tuning a segmentation network and a test set for benchmarking the performance. The training set contains 39,068 motion sequences spanning 151,335 RGB-D frames and 14 object categories (such as chairs, sofas, and beds). The test set consists of 20 sequences with longer motion. The average length of camera movement is 5.1 m, and the object trajectory length is 8.6 m. The rooms have an average of 6 models, leaving out walls/ceilings/wall decorations.

3.4 Evaluation Metrics

3.4.1 Reconstruction

A good reconstruction metric should handle the following issues: (a) different 3D representation, (b) lack of correspondences between ground truth surface and an output surface, and (c) the model space may be different from the canonical world space (dependent on the implementation). In order to assess surface reconstruction quality, we reconstruct ground truth mesh [4] using ground truth poses, ground truth segmentation, and depth frames. Note that this fusion result is different from the actual 3D CAD model since it contains occlusions from the virtual scanning.

We handle different surface representations by converting them into a point-based representation (i.e., point cloud) and conducting evaluation with the ground truth meshes’ vertices. Specifically, for the volumetric-based methods [6, 8, 22], we
Figure 3.2: Example scenes in our DYNSYNTH dataset. We synthesize rigid motions along the ground, across the scene. The dynamic scenes are in turn recorded by moving cameras.
use the vertices of a reconstructed mesh as an output point set $S_{gt}$. If the volumetric grid has a low resolution (e.g., less than 128), we re-sample the point on the reconstructed mesh to create an output point set. For surfel-based methods [14, 37], we use the centers of each surfel as an output point set. The surfel representation is usually very dense, so we do not perform upsampling. We tackle the correspondence issue and report Precision and Recall by employing a Chamfer distance metric, such that:

$$
\text{Precision} = \frac{1}{N} \cdot \sum_{x \in S_{pred}} \mathbf{1}(\min_{y \in S_{gt}} \|x - y\|^2 < \varepsilon) \quad (3.1)
$$

$$
\text{Recall} = \frac{1}{M} \cdot \sum_{y \in S_{gt}} \mathbf{1}(\min_{x \in S_{pred}} \|x - y\|^2 < \varepsilon), \quad (3.2)
$$

where $\mathbf{1}$ is an indicator function and $\varepsilon$ is a distance threshold to determine whether a point is successfully captured. We set $\varepsilon$ to 3cm by considering that our scene size is generally large than $3m^2$. We defined Recall as the squared distance between every point in the ground truth $S_{gt}$ to the corresponding nearest point in the output point...
3.4. Evaluation Metrics

(i) P:1.0, R:1.0.
(ii) P:0.7, R:1.0.
(iii) P:0.5, R:1.0.
(iv) P:1.0, R:0.54.

Figure 3.4: Assessing reconstruction quality evaluation and types of errors. (i) Ground truth reconstruction, (ii) A noisy reconstruction. Low precision is usually caused by tracking lost, which leads to misaligned surfaces. (iii) Another noisy reconstruction example. This happens when outliers are accumulated in the model over time due to inaccurate foreground/background segmentation. (iv) A partial reconstruction example. This is usually caused by missed detection, which skips some views of the object.

set \( S_{\text{pred}} \), and Precision as the squared distance calculated from the inverse direction, i.e., from the output points to the ground truth points. In Figure 3.4, we show several output examples and the corresponding reconstruction scores.

3.4.2 Tracking

To evaluate the quality of foreground detection and tracking, we employ the standard visual object tracking metrics [138]: multiple object tracking accuracy (MOTA) and precision (MOTP). These two metrics show the percentage of tracked frames and tracking errors:

\[
\text{MOTA} = 1 - \frac{\text{MISS} + \text{BAD} + \text{SWITCH}}{N}
\]

\[
\text{MOTP} = \sqrt{\frac{1}{N} \sum_{i} \| c_i - c^g_i \|^2},
\]

where MISS, BAD, SWITCH, N are the number of false positives, loss-tracked frames, identity switches, and input frames; and \( c \) and \( c^g \) represent the estimated and ground truth center at frame \( i \). In addition, a distance threshold is introduced to define whether the foreground object is tracked or not. We set the threshold to 5cm according to the TUM-RGBD tracking benchmark [11, 139]. The failed tracked frames are marked as BAD frames. In Figure 3.5, we show several examples and the corresponding scores.
3.5 Discussion and Limitations

In this chapter, we developed a data generation toolkit and established an evaluation protocol for benchmarking the 4D reconstruction task and resolving the lack of training data issue. We will demonstrate that our dataset can be used to learn data priors for object tracking in Chapter 4 and servers as a test set for evaluation purposes in Chapter 5. We proposed a simple but efficient toolkit. However, it has three main limitations:

- **Photo-realistic rendering.** Our toolkit is built using OpenGL rendering API, providing real-time rendering performance. However, it doesn’t render shadows, which creates a gap between our synthesized data and real-world environments. Automatically placing indoor lighting and integrating a ray tracer will be useful extensions to our toolkit.
3.5. Discussion and Limitations

- **Non-rigid motion.** Our toolkit cannot produce non-rigid motion such as human activities. This problem is still an open research problem because synthesizing non-rigid motion requires not only collision detection but also skeleton information and a posture dataset.

- **Multi-objects interaction.** We do not support object-to-object interaction as it is hard to define a meaningful multi-object movement through a set of heuristics. Designing a plausible multi-object event will require manual annotations, which is hard to scale up and generate a large dataset for training purposes.
Chapter 4

Supervised Dynamic Objects

Reconstruction

4.1 Introduction

Reconstructing moving objects requires detecting the moving objects and estimating their motion. One common strategy in this regard is utilizing an instance segmentation network [61, 140, 141, 142], which infers a mask for each known object (i.e., the object exists in the training data) and tracks the predicted masks in a frame-by-frame fashion [37, 38, 39]. However, robust object tracking is still challenging due to occlusions caused by the monocular input setting and object movements.

This chapter presents the idea of inferring unseen surfaces to track occluded moving objects. We first assume that the moving objects belonged to one of the categories in our training set. Then, we develop a joint optimization framework with differentiable pose optimization to jointly infer object centers, bounding boxes, canonical coordinates, and object poses, as shown in Figure 4.1. The object center and bounding box predictions allow us to crop object-related features. The canonical coordinates map the input space to object space, being useful for merging the object’s observations across time. Our approach differs from the recent object SLAM systems [14, 37, 81] because we consider the observed geometry and utilize geometry priors to track moving objects.

It must be here acknowledged that this is a joint project with Norman Muüßler.
4.2 Method

Figure 4.1: The illustration of our joint optimization formulation. Given a sequence of RGB-D frames, our system detects instances, completes their implicit surface, and infers the object’s canonical coordinates as correspondences. Our network is jointly trained with differentiable pose optimization to achieve multi-object tracking.

from the Technical University of Munich. I contributed to data generation, real-world data recording, implementing baselines, setting up the comparison methods, and the method discussion.

4.2 Method

An overview of our network architecture for joint object completion and correspondence regression is shown in Figure 4.2. Our method takes as input an RGB-D sequence and learns to detect object instances, complete occlusion object geometry, and infer object coordinates as the correspondences between an object to its canonical model. We then associate the predicted objects across frames to obtain object tracking and reconstruction over time.

4.2.1 Object Detection

Each RGB-D frame of the sequence is represented by a sparse grid $S_i$ of surface voxels and a dense truncated signed distance field (TSDF) $D_i$. Following the standard volumetric fusion [4], an input RGB-D frame is back-projected to 3D space to obtain a partial TSDF grid using the observed depth values and the camera pose. We follow the state-of-the-art 3D object detection networks [61, 140, 141, 142] and employ a multi-task loss [140, 141] with a sparse 3D convolutional auto-encoder to encode
4.2. Method

Figure 4.2: Overview of our network architecture. From an input frame, we employ sparse and dense backbones to extract geometry features. The extracted features are used to detect object centers, complete unseen geometry, and generate correspondence prediction. By hallucinating the occluded surface and using them to associate objects between frames, our method achieves robust dynamic object tracking in high-dynamic scenes.

surface information. The sparse encoder compresses the input to 1/16 of the original resolution, and the decoder upsamples the latent grids back to the input resolution. The output sparse features grid \( F \) is then fed to the multi-head detection module. The detection module infers objectness \( O(v) \), center coordinates \( C(v) \), object bounding box \( D(v) \), and semantic labels \( S(v) \) on each voxel \( v \). We supervise our detection module using the loss function \( L_{\text{detection}} \), such that:

\[
L_{\text{detection}} = \lambda_1 L_o + \lambda_2 L_c + \lambda_3 L_d + \lambda_4 L_s
\]

where \( \lambda \) is the weighting scalar. The individual terms are defined as:

\[
L_o = \text{BinaryCrossEntropy}(O, O')
\]

\[
L_c = \begin{cases} 
\frac{1}{2}(C - C')^2 & \text{for } \|C - C'\| \leq 0.5, \\
\|C - C'\| - \frac{1}{2} & \text{otherwise}
\end{cases}
\]

\[
L_d = \begin{cases} 
\frac{1}{2}(D - D')^2 & \text{for } \|D - D'\| \leq 0.5, \\
\|D - D'\| - \frac{1}{2} & \text{otherwise},
\end{cases}
\]

\[
L_s = \text{CrossEntropy}(S, S')
\]
4.2. Method

where \( O', C', D', \) and \( S' \) represent ground truth objectness, centers, extents, and semantic labels. The objectness \( O \) is represented as a binary mask indicating the object surface. The object center \( C \) and the bounding box \( D \) are represented as the relative offsets and extents of the voxel \( v \). We use smooth L1 loss [140] for center and extent regression. Note that the normalization constants and summation are omitted here for clarity.

Next, we apply mean-shift clustering (20 steps, with 8 voxel radius) to extract object proposals. We only perform clustering on the voxels having positive objectness scores and remove small clusters containing less than 50 elements. We estimate the remaining clusters’ bounding boxes using averaging and extract their centers, geometry, and semantic labels using majority voting.

4.2.2 Object Completion and Correspondences

Given the inferred object proposals, we perform Sparse-to-Dense feature fusion by cropping the corresponding sparse features \( f_k \) from \( F \) and the dense TSDF grid \( D \). We fuse sparse features with dense TSDF information by adding TSDF values over the feature channels. The fused features grid \( f'_k \) of the proposal \( k \) is first down-sampled by a factor of two and then fed into a dense backbone using 3D convolutions to obtain object features \( f'_o_k \). Using the predicted dense features, we predict unseen object geometry \( m_k \) through another series of dense 3D convolutional layers. The completed geometry \( m_k \) is represented as an occupancy grid and supervised using binary cross-entropy loss:

\[
L_{\text{comp}} = \lambda_{\text{comp}} \text{BinaryCrossEntropy}(m_k, m'_k)
\]  

(4.6)

with \( m'_k \) and \( \lambda_{\text{comp}} \) denoting a ground truth occupancy grid and a weight, respectively.

Leveraging the same object feature \( f'_o_k \), we employ a similar 3D convolutions network to infer object coordinates \( c_k \). Inspired by the normalized object coordinate space (NOCS) [57], we define the object coordinates in the object’s canonical space
and supervise the coordinate regression by $\ell_1$ loss.

$$L_{\text{noc}} = \lambda_{\text{noc}} \| c_k - c'_k \|$$

with $c'$ denoting the target correspondences in $\mathbb{R}^3$ and $\lambda_{\text{noc}}$ denoting a weight. Combining both $m_k$ and $c_k$, we can now perform pose estimation with unseen geometry and bypass correspondence searching.

### 4.2.3 Differentiable Pose Optimization

We solve object scale $\nu^*$, rotation $R^*$, and translation $t^*$ using differentiable Procrustes analysis, which minimizes the point-to-point distance between the predicted object coordinates $P_o$ and their predicted canonical representation $P_n$:

$$\nu^*, R^*, t^* := \underset{\nu \in \mathbb{R}^+, R \in SO_3, t \in \mathbb{R}^3}{\text{argmin}} \| P_o - (\nu R \cdot P_n + t) \|^2.$$  

The closed-form solution [143] of this objective function can be obtained through a differentiable SVD of $(P_o - \mu_o)(P_n - \mu_n)^T = UDV^T$, where $\mu_i$ and $\sigma_i$ are the means and variances of $P_i, i \in \{o, n\}$, and we have the optimal

$$\nu^* = \frac{1}{\sigma_n} tr(DS), R^* = USV^T, \text{ and } t^* = \mu_o - \nu^* R^* \mu_n, \quad (4.9)$$

where $S = \text{diag}(1, 1, \text{det}(UV^T))$.

To build a joint optimization pipeline, we supervise the estimated rotation matrix using a Frobenius norm loss, the predicted scale using an $\ell_1$ loss, and the translation using an $\ell_2$ loss, as:

$$L_{\text{pose}} = \lambda_R L_R + \lambda_\nu L_\nu + \lambda_t L_t$$

$$= \lambda_R \| R - R' \|_F + \lambda_\nu \| \nu - \nu' \| + \lambda_t \| t - t' \|_2$$

Similar to NOCS [57], we avoid the rotation ambiguity of symmetric objects by taking the minimum rotation error between the predicted rotation and the possible valid rotations. To stabilize training, we use ground truth geometry at training time.
and the predicted geometry at the test time.

Finally, our multi-task loss is defined as

\[
\min_{\theta} L_{\text{total}}(\theta) = L_{\text{detection}} + L_{\text{comp}} + L_{\text{noc}} + L_{\text{pose}}.
\] (4.11)

The weights \( \lambda_o, \lambda_c, \lambda_d, \text{ and } \lambda_e \) of the detection loss \( L_{\text{detection}} \) are set to 1.0, 0.1, 0.1, and 1.0, respectively, because object centers and bounding boxes are calculated in voxel units. The completion weights \( \lambda_{\text{comp}} \) and the correspondences weights \( \lambda_{\text{noc}} \) are set to 4, and the weights \( \lambda_R, \lambda_V, \text{ and } \lambda_t \) of the pose loss \( L_{\text{pose}} \) are set to 0.2, 0.1, and 0.1, respectively, in order to balance each term.

### 4.2.4 Object Tracking and Reconstruction

To achieve multi-object tracking and reconstruction, we associate object proposals in each frame and map them into their canonical representation using a 64\(^3\) resolution grid. Specifically, we construct object tracklets frame-by-frame by computing the pairwise distance from a detected object to all object proposals in a new frame using the predicted bounding boxes and 3D IoU. Then, we use the Hungarian algorithm [144] to find the best matches and filter outlier proposals by setting a 3D IoU threshold (set to 0.3). Non-matched object proposals will be seen as a new object and initialize a new tracklet.

We fuse the object’s canonical model by maintaining a moving average between the current model and the new information, similar to volumetric fusion but operating with occupancy. We observed associating object proposals in their canonical space (through the predicted correspondences) leads to an improvement in matching accuracy and tracking, as demonstrated in the evaluation section later (Section 4.3).

### 4.2.5 Implementation Details

We implement our network using PyTorch and its differential SVD function. We run our experiments using a single Nvidia GeForce RTX 2080 GPU. We employ an Adam optimizer with a learning rate of 1.0e-3 and a batch size of 2 due to the limited GPU memory. We set the number of object proposals to 10 and select the weights for each loss using the validation split. This weight-tuning step is essential.
4.3 Evaluations and Results

We evaluate our method against the existing methods (MaskFusion [37] and MIDFusion [38]) on our DYNSYNTH dataset consisting of 3,300 scenes (97,626 frames in total). We split the scenes into 2900/300/100 as training/validation/test and predict 10 object classes, including common indoor furniture, such as bed, cabinet, and chair class. We also conduct evaluations on real-world scans, including both dynamic and static environments. We record real-world dynamic events using a Structure Sensor\(^1\) mounted to an iPad and provide a qualitative comparison against the related methods. We evaluate our pose estimation accuracy using static data. To create real-world training data, we combine the ScanNet [12] dataset with the Scan2CAD [42] annotations. As the ScanNet dataset is recorded in 30 FPS, we sample every 20 frames and generate 114,000 frames in total, and we employ the official splits for

\(^1\)https://structure.io/

---

**Figure 4.3: Our network architecture.** Dotted lines represent skip connections. Green-colored boxes mean outputs.
4.3. Evaluations and Results

Figure 4.4: Qualitative comparison on DYSYNTH. Qualitative comparison to state of the art on DYSYNTH test sequences. Our method inferring the occluded surface maintains a better association and tracking over time leading to robust object tracking, e.g., our estimated trajectory is closer to ground truth as shown from the top-view ($t_3$). The colored lines show the estimated object trajectories.

training, validation, and testing (944/149/100). To measure the performance, we use the evaluation metrics introduced in Section 3.4 and report multiple object tracking accuracy (MOTA) on test sequences.

4.3.1 Comparison to State of the Art

MaskFusion is a surfel-based tracking and reconstruction method built on ElasticFusion [83] and MaskRCNN [58]. MaskFusion employs a segmentation refinement
4.3. Evaluations and Results

Table 4.1: Evaluation of MOTA on DYN SYNTH. Our end-to-end network jointly infers unseen geometry and object correspondences leading to notable performance improvement over the state-of-the-art methods and baselines, including disabling object completion (no comp.) and only using IoU-based matching (no corr.). Please note that our method has lower accuracy for large objects such as beds and bookshelves due to it is hard to aggregate their features using sparse convolution.

<table>
<thead>
<tr>
<th>MOTA(%)</th>
<th>bathtub</th>
<th>bed</th>
<th>bookshelf</th>
<th>cabinet</th>
<th>chair</th>
<th>desk</th>
<th>sink</th>
<th>sofa</th>
<th>table</th>
<th>toilet</th>
<th>seq. avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaskFusion [37]</td>
<td>27.7</td>
<td>76.4</td>
<td>25.4</td>
<td>24.4</td>
<td>25.3</td>
<td>33.8</td>
<td>39.2</td>
<td>5.7</td>
<td>45.8</td>
<td>27.7</td>
<td>17.2</td>
</tr>
<tr>
<td>MID-Fusion [81]</td>
<td>55.8</td>
<td>100</td>
<td>94.7</td>
<td>21.7</td>
<td>38.6</td>
<td>45.8</td>
<td>63.9</td>
<td>9.6</td>
<td>53.8</td>
<td>35.7</td>
<td>30.1</td>
</tr>
<tr>
<td>F2F-MaskRCNN</td>
<td>25.7</td>
<td>100</td>
<td>73.7</td>
<td>15.2</td>
<td>28.3</td>
<td>79.2</td>
<td>73.2</td>
<td>21.2</td>
<td>59.6</td>
<td>33.9</td>
<td>35.8</td>
</tr>
<tr>
<td>(no corr., no comp.)</td>
<td>39.8</td>
<td>54.5</td>
<td>22.6</td>
<td>21.8</td>
<td>27.2</td>
<td>37.5</td>
<td>49.5</td>
<td>13.8</td>
<td>60.4</td>
<td>36.7</td>
<td>29.3</td>
</tr>
<tr>
<td>(no corr.)</td>
<td>39.8</td>
<td>54.5</td>
<td>24.0</td>
<td>23.2</td>
<td>32.2</td>
<td>37.5</td>
<td>50.3</td>
<td>13.8</td>
<td>61.8</td>
<td>38.1</td>
<td>30.6</td>
</tr>
<tr>
<td>(no comp.)</td>
<td>24.9</td>
<td>45.5</td>
<td>50.0</td>
<td>26.1</td>
<td>42.3</td>
<td>66.4</td>
<td>63.3</td>
<td>18.0</td>
<td>63.2</td>
<td>38.0</td>
<td>35.6</td>
</tr>
<tr>
<td>Ours</td>
<td>24.9</td>
<td>45.5</td>
<td>50.1</td>
<td>26.1</td>
<td>51.8</td>
<td>66.4</td>
<td>63.3</td>
<td>17.3</td>
<td>67.4</td>
<td>49.0</td>
<td>42.3</td>
</tr>
</tbody>
</table>

step to alleviate occlusion, but it cannot handle non-convex surface and disconnected segments, which leads to poor object association and tracking. Additionally, the weighted surfel tracking scheme employed by MaskFusion requires a static initialization period and is unstable to handle high-dynamic sequences, as shown in Table 4.1.

MID-Fusion is a volumetric fusion method using Octrees and MaskRCNN [58]. MID-Fusion achieves better MOTA compared to MaskFusion because of its volumetric representation, which alleviates the occlusion problem and the low recall issue of MaskRCNN through back-projecting the consolidated models. However, it cannot associate with highly occluded objects, such as the qualitative examples in Figures 4.5 and 4.4.

We also implement a frame-to-frame tracking method using iterative closest point (ICP) [145, 146] with the segmentation predicted by MaskRCNN [58], named F2F-MaskRCNN. As ICP has a correspondence searching step instead of the projective mapping strategy used by real-time methods (MID-Fusion and MaskFusion), F2F-MaskRCNN performs better under fast object motion. However, it cannot resolve occlusion or weakly constrained geometry [147], such as the chair objects in Figure 4.5.

Compared to the above methods, our approach is the only method inferring the unseen geometry to perform object association and tracking, which achieves the best mean MOTA in Table 4.1.
4.3. Evaluations and Results

Table 4.2: Evaluation of object pose estimation on individual RGB-D frames. Predicting the underlying geometry of each object enables more accurate object pose estimation in each frame.

<table>
<thead>
<tr>
<th></th>
<th>DynSynth</th>
<th>ScanNet+Scan2CAD</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Median Rot. errors</td>
<td>Median Transl. errors</td>
</tr>
<tr>
<td>Ours (no comp.)</td>
<td>7.4°</td>
<td>15.4 cm</td>
</tr>
<tr>
<td>Ours</td>
<td>5.7°</td>
<td>12.3 cm</td>
</tr>
</tbody>
</table>

4.3.2 Does Object Completion Help Tracking?

We validate our key hypothesis through ablation studies in Table 4.1. When object association is calculated using 3D bounding box overlap, our completion variant (no corr.) slightly improves tracking performance by 1.3\% mean MOTA than no completion variant (no corr., no comp.). When objects are matched based on correspondence prediction without completion prior (no comp.), it further enhances tracking performance by 5\% mean MOTA. Our full model achieves the best performance via utilizing object completion and correspondence prediction to associate and track objects leading to 42.3\% mean MOTA. In addition, we evaluate our pose estimation performance on DYN SYNTH and ScanNet in Table 4.2. Our evaluation results on both datasets also confirm our hypothesis that object completion can improve object tracking and pose estimation.

4.3.3 Real-world Dynamic RGB-D Sequences

To fill the synthetic-to-real gap, we pre-train our network using our synthetic DYN SYNTH dataset. Then, we finetune our network on a static real-world dataset using ScanNet with Scan2CAD annotations. As we do not have ground truth information for real-world dynamic data, we qualitatively evaluate our method’s performance against comparing methods in Figure 4.5. Our method outputs more reasonable object trajectories and provides visually plausible object reconstruction.

4.3.4 Detection and Completion Evaluation

We evaluate our detection and completion module on both DYN SYNTH and ScanNet+Scan2CAD in Table 4.3. For detection, We follow the standard evaluation protocol [58] and report mean average precision (mAP) at a 3D bounding box IoU
4.3. Evaluations and Results

Figure 4.5: Qualitative comparison to state of the art on real-world sequences. Our method recovers full object trajectories and accurate object shapes over time. The colored lines show the estimated object trajectories.
Table 4.3: Evaluation of 3D detection and instance completion. The real-world dataset (ScanNet+Scan2CAD) is harder than the synthetic dataset (DYNSYNTH). Big objects such as bookshelves are harder to detect for our sparse convolutional network.

<table>
<thead>
<tr>
<th>Object</th>
<th>3D Detection</th>
<th>Instance Completion</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3D SYNTH</td>
<td>ScanNet+Scan2CAD</td>
</tr>
<tr>
<td>bathtub</td>
<td>49.3</td>
<td>38.7</td>
</tr>
<tr>
<td>bed</td>
<td>38.4</td>
<td>-</td>
</tr>
<tr>
<td>bookshelf</td>
<td>12.5</td>
<td>12.9</td>
</tr>
<tr>
<td>cabinet</td>
<td>6.3</td>
<td>4.6</td>
</tr>
<tr>
<td>chair</td>
<td>44.1</td>
<td>41.2</td>
</tr>
<tr>
<td>desk</td>
<td>46.8</td>
<td>-</td>
</tr>
<tr>
<td>sink</td>
<td>27.6</td>
<td>-</td>
</tr>
<tr>
<td>sofa</td>
<td>32.3</td>
<td>26.4</td>
</tr>
<tr>
<td>table</td>
<td>38.4</td>
<td>29.2</td>
</tr>
<tr>
<td>toilet</td>
<td>63.1</td>
<td>-</td>
</tr>
<tr>
<td>mAP</td>
<td>35.8</td>
<td>25.6</td>
</tr>
</tbody>
</table>

of 0.5. For completion, we also report mean average precision (mAP) at mesh IoU of 0.25. As we expected, real-world data is more challenging than synthetic data due to the variance of object appearances and geometry is larger. Our sparse detection backbone [61, 142] has an inherited limitation on detecting and regressing big objects such as beds, which is also the bottleneck of our tracking performance, as shown in Table 4.1.

4.4 Discussion and Limitations

In this chapter, we propose a joint optimization framework to track and reconstruct rigid objects in RGB-D sequences. We demonstrate that the completion and correspondence priors can help object tracking and provide more accurate information for associating objects across frames alleviating the occlusion problem caused by view changes. Our approach also has some limitations, as discussed next.

- **Reconstruction quality.** While our method improves the reconstruction quality, several open research problems remain unsolved, including capturing high-frequency geometric details, thin structures, and glass surfaces. Jointly modeling unseen geometry as well as addressing these issues is worth exploring.

- **Loop closure.** Handling long-term tracking is another issue since tracking errors will be accumulated. Integrating traditional feature descriptors or a learning-based visual tracker will further improve tracking performance.
• **Non-rigid motion.** We do not reconstruct non-rigid objects, which requires a lightweight architecture to model complex non-rigid deformation. Note that our dense backbone already consumes a large portion of the GPU memory.

• **Object association.** We employ the Hungarian algorithm [144] to solve object association. However, it is non-differentiable. Modeling object association through a transformer network or a graph attention network will be interesting.
Chapter 5

Unsupervised Dynamic Scene Reconstruction

5.1 Introduction

Acquiring dynamic indoor scenes with rigidly-moving objects and a non-static camera is an ill-posed problem due to the dependency on tracking and segmentation. Previous work commonly employs the tracking-by-detection paradigm by discovering foreground objects using an instance segmentation network [37, 38, 39] and motion residuals [14] to resolve the cyclic dependency on rigid object tracking and segmentation. However, using the segmentation prior is restricted to the known object classes, and the motion cues can be ambiguous when the camera has non-static movements.

Our key idea is to discover the foreground object through the accumulated free space information from multiple frames, which has been shown as an effective signal for outlier removal [22]. The free-space grid and the background model are used to separate moving elements using a frame-to-model back projection of the input frames. Instead of performing an expensive global optimization for solving a complete background model, we employ a delayed process, similar to the batch processing approach [11], to regularize the problem, as shown in Figure 5.2.

We segment out pixels of humans in image space to facilitate real-world scanning scenarios with an operator interacting with moving objects. The prediction
5.2. Method

Figure 5.1: RigidFusion’s 4D reconstruction results. Given a scene with rigidly moving objects (rendered with cyan and yellow), RigidFusion performs 4D reconstruction from RGB-D frames. We present a novel segmentation-by-reconstruction framework that factorizes camera motion (shown in green) and fused object geometries along with their respective motion (shown in brown/purple) over time from raw RGB-D scans. Two novel-view reconstructions from two timesteps (shown in blue/orange) are shown in the left columns.

of human segmentation may be inaccurate or missing. To purge outlier pixels, we utilize free space information in the foreground model. Aside from this off-the-shelf human segmentation method, our method requires no learning or training data, is agnostic to the underlying volumetric fusion framework, and runs at interactive frame rates after the initialization step.

In addition to real-world scanning cases for qualitative evaluation and comparisons on established datasets [14, 139], we employ our DYNSYNTH dataset, as presented in Chapter 3. Our synthetic dataset provides a training set for fine-tuning the state-of-the-art segmentation network [37] and a test set we used for quantitative evaluation. In a series of experiments, we show that the proposed method, RigidFusion, outperforms existing state-of-the-art methods by a significant margin and handles significantly more challenging real-world cases.
5.2 Method

5.2.1 Overview

Our method takes as input a sequence of RGB-D frames \( \{F_i\} \) captured using a moving camera recording a dynamic scene with rigidly moving object(s). We assume the setup to satisfy two conditions: dense recording, i.e., the input frames come as a continuous RGB-D video; and objects either remaining static or rigidly being moved, one at a time, by a human operator. Note that although we assume a single object to be moving at any time, we do not require any object segmentation priors, such as in MaskFusion [37]. As output, we produce a 4D reconstruction in the form of consolidated static background mesh, camera trajectory, and the consolidated foreground object meshes along with the respective object trajectories over time.

In order to break the cyclic dependency between rigid tracking and motion segmentation, RigidFusion proceeds in the following steps: (i) background reconstruction and camera estimation, (ii) asynchronous foreground reconstruction, (iii) optional post-processing, and (iv) mesh extraction. We now describe the individ-

---

**Figure 5.2: RigidFusion’s input buffer.** Our method runs at 1 fps with a delay of \( \Delta \) frames for foreground reconstruction. The short delay between the background and foreground module allows the system to accumulate more free space information.
Figure 5.3: RigidFusion’s system diagram. (a) The background model and input depth at the early frame. The human regions are masked out using human detection. (b) The background model and input depth at the time $i$ (equal to $j + \Delta$). (c) The foreground model and the unknown segmentation $u_j$. (d) The background model is reset (due to the status change of the tracked object triggering the model de-activation). (e) The foreground models and the unknown segmentation at the later frame.

5.2.2 Background Reconstruction and Camera Estimation

Human Detection and Masking. Since we focus on capturing background and rigidly moving objects, we first perform human detection in each frame $F_i$. We apply a DensePose [148, 149] detector to acquire a human mask. We reset the depth of the pixels under the detected mask to zero for each frame to serve as the input for subsequent processing. This masking stage not only avoids the existence of non-rigid objects interfering with the camera tracking but also removes the requirement that humans should continuously move, as commonly assumed in [22, 31].

Free-Space Aware TSDF Fusion. To reconstruct a (static) scene, we employ
volumetric fusion using sparse voxel hashing [4, 8]. Recall that in the standard implementation, only the voxels near the recorded surface are stored, i.e., a truncated signed distance field (TSDF) is progressively built. Additionally, RigidFusion maintains, using voxel hashing, free space counts to identify free space (outside) voxels, i.e., the voxels that have been frequently observed and have positive distances larger than the truncation margin. An illustration is shown in Figure 5.4, where red/blue cells denote negative/positive SDF values (underlying surface marked in green) that are within the truncation range, and gray cells denote outside free cells. We use this free-space information to prevent foreground signals from polluting the background model. The pseudo-code of our proposed fusion method is listed in Algorithm 1.

**Camera Tracking Optimization.** In our tracking formulation, we use 6D vectors $\xi \in \mathfrak{se}_3$, encoded as instantaneous velocity using Lie algebra, to represent a $4 \times 4$ rigid transformation $T$. To transform a spatial point, we convert $\xi$ to a $4 \times 4$ transformation matrix using exponential map, denoted by $exp$, as described in [6, 150]. To perform camera tracking, we employ frame-to-SDF registration [22, 151]. Given the input as an RGB-D frame $F_i$ at time $i$, we solve for the best increment transformation $\xi$ to align the input frame to the TSDF model as,

$$ T_i^{(c)} = exp(\xi) \cdot T_{i-1}^{(c)}, $$

where $T_i^{(c)}$ represents the estimated camera pose at frame $i$. Our tracking objective $E$ includes both geometry and color intensity cost functions and is defined as

$$ E(\xi) = E_{SDF}(\xi) + \alpha E_I(\xi), $$

Figure 5.4: TSDF illustration. The green curve, gray, blue, and red cells represent the surface, free space, positive and negative truncation regions, respectively.
where $\alpha$ is a scalar weight for balancing the two terms. The geometric cost term $E_{SDF}$ minimizes the signed distance value of the transformed input point set $P$, as any surface voxel has zero signed distance values, and hence a perfectly aligned point set should have a zero residual. The input points are firstly transformed into the world space using the accumulated pose $T^{(c)}_{i-1}$.

$$E_{SDF}(\xi) = \sum_{p \in P} (\psi_g(T^{(c)}_{i-1}p))^2 = \sum_{p \in P} (\psi_g(exp(\xi) \cdot T^{(c)}_{i-1}p))^2,$$

where $\psi_g$ represents a signed distance function $\psi_g : R^3 \rightarrow R$ that takes a 3D point and returns a truncated signed distance value from the TSDF model. The photometric cost term $E_I$ minimizes the difference of color intensity between the transformed input points $p \in P$ to the corresponding voxels in the model, where $I$ represents the input intensity map by converting the input RGB to relative luminance as,

$$E_I(\xi) = \sum_{p \in P} (\psi_I(T^{(c)}_{i}p) - I(p))^2.$$

Here, $\psi_I$ represents a color interpolation function $\psi_I : R^3 \rightarrow R$, which takes a point and returns an interpolated color intensity from the TSDF model. To optimize our objective (5.2), we apply a Gaussian Newton solver while linearizing the objective around the initial $\xi$.

**Background Model Update.** We take the estimated pose and transform the input frame to integrate the new information. Note that we prevent any surface (depth pixels) from being integrated into free voxels. Please refer to [4] and Algorithm 1.

### 5.2.3 Asynchronous Foreground Reconstruction

We regularize the foreground segmentation problem by assuming only one rigidly moving object at any point. Note that by freezing the previously detected object, our proposed scheme still supports capturing multiple moving objects (by detecting a new moving object). We provide two examples in Figure 5.6, where we employ a simple heuristic to deactivate previous-moving foreground objects and detect a new object.
5.2. Method

**Segmentation by Reconstruction.** We define the unknown foreground segmentation $u_j$ at the time $j$ as the depth pixels that unproject to the free space of the background model. We apply connected component filtering to remove small blobs and initialize a new active object if the area of $u_j$ is larger than a threshold. We empirically set the threshold to 1% of the image size. Note that remaining false positives are later removed in our foreground deactivation step (see later) due to the lack of correspondences detected using tracking failures. Compared to solving segmentation using color (2D) information, our approach utilizes the space information (3D) that resides in the background model and therefore is able to extract object segmentation without prior knowledge. We provide the pseudo-code of our segmentation-by-reconstruction step in Algorithm 2.

**Delayed Processing.** For motion segmentation, a frame-to-frame approach, such as [14, 31], cannot guarantee the observed motion signals are sufficient because an object may move slowly. Instead, we delay foreground tracking by a pre-defined window size $\Delta$ (60 frames), leading to a delayed reconstruction in the foreground modules at run-time to gather more background information, as shown in Figure 5.2. In other words, when the foreground module is processing the frame at the time $i$, the background module has processed the frames until the time $i+\Delta$. Hence, the extracted unknown foreground segments $u_j$ at the time $j$ can directly access the future background model during the segmentation-by-reconstruction step. This step is similar to doing scene completion during registration in the sense that the foreground module can use a completed background model. Note that our asynchronous processing does not add any extra memory overhead and only has a short delay in the beginning.

**Foreground Tracking and Reconstruction.** To estimate the pose $T_k^j$ of the active object $k$ at the time $j$, as well as reconstruct its geometry, we apply the same tracking optimization and free-space aware fusion used in the background module (see Section 5.2.2 and Algorithm 1). Note that, during foreground tracking, we still perform frame-to-SDF tracking without an instance mask since the non-foreground pixels will receive zero gradients in the foreground TSDF model.
Algorithm 1: Free-space Aware TSDF Fusion

|   | 
|---|---|
| **Input:**  | a RGBD frame, TSDF, FreeGrid, instance mask $u$, camera frustum $\eta$  |
| **for** | each voxel $v \in \text{TSDF} \cup \eta$ do  |
|   | $c \leftarrow$ the free-space count of $v$ in FreeGrid  |
|   | $C \leftarrow$ the free-space threshold  |
|   | $sdf \leftarrow$ the signed distance value of $v$  |
|   | $w \leftarrow$ the weight of $v$  |
|   | $v_{2d} \leftarrow$ projected image coordinates on the input frame  |
|   | $dist \leftarrow$ the signed distance from $v$ to the back-projected depth pixel at $v_{2d}$  |
|   | /* reject integration and denoise */  |
|   | if ($c \geq C$) then  |
|     | if $w > 0$ then  |
|     |  remove the voxel $v$  |
|     | end  |
|     | continue  |
| end  |
| /* integration */  |
| isForeground $\leftarrow u(v_{2d})$  |
| if (isForeground AND $|dist| < \text{truncation}$) then  |
| /* In truncation, do standard integration */  |
| update the $w$ and $sdf$ using running mean as in [4]  |
| else if ($dist \geq \text{truncation}$) then  |
| /* In free space */  |
| $c \leftarrow c + 1$  |
| end  |
| else  |
| /* In occluded space */  |
| continue  |
| end  |

Handling Multiple Objects by Deactivation. In RigidFusion, we assume multiple objects do not move simultaneously. We deactivate any active object under any of the following conditions: (i) an active object becomes static over a period of time (i.e., accumulated pose difference is less than $1e^{-4}$ over the last $\Delta/2$ frames); (ii) an active object moves out of the camera view; or (iii) an active object cannot be successfully tracked for more than ten frames.

Once a deactivation event is triggered, we temporarily reset the background model (i.e., allocate a new TSDF model) and freeze foreground detection for $\Delta$ frames to prevent inactive objects from being re-detected in the unknown segment.
Algorithm 2: Segmentation by Reconstruction

**Input:** an input depth frame $D$, a human detection mask $h$, background model, camera pose $T_{j}^{(c)}$

**Output:** instance mask $u_{j}$

1. $u_{j} \leftarrow$ Initialize a 2D mask with false values
2. $C \leftarrow$ the free-space threshold
3. set the human segments’ depth to zero values in $D$ using $h$
4. $d_{max} \leftarrow$ the maximum depth value
5. $\text{FreeGrid} \leftarrow$ background model’s free-space grid
6. for each pixel at $(x, y) \in$ input depth $D$ do
   /* skip invalid depth */
   7. if ($D(x, y) = 0$ or $D(x, y) > d_{max}$) then
      8. continue
   end
   /* back-project depth and transform to the world space */
   9. $p \leftarrow T_{j}^{(c)} \cdot \text{backproject}(D(x, y))$
   /* query the background model */
   10. $c \leftarrow \text{FreeGrid}(p)$
   /* valid check */
   11. if ($c > C$) then
      12. $u_{j}(x, y) \leftarrow$ true
      end
   end

proposal $u_{j}$.

### 5.2.4 Post-Processing

After processing all input frames, we can perform an optional post-process step (PS) to optimize full 4D reconstruction. First, given the captured foreground models and trajectories, we perform backward tracking for each object from its first detected frame to the first frame. This step can recover some missing frames. Second, we rebuild background reconstruction by using a high-resolution grid and re-optimize the camera trajectory $\left\{ T_{i}^{(c)} \right\}$. During this optimization, the already inferred foreground models are used to mask out foreground pixels by depth ray-casting. Thus it improves the camera tracking accuracy. We provide ablation studies of this step in Tables 5.1 and 5.2.
5.2.5 Mesh Extraction

To output surface reconstruction, we perform Marching Cube on each model and extract a 3D mesh along with its corresponding trajectory.

5.3 Evaluations and Results

We compare RigidFusion with different state-of-the-art reconstruction methods, including VoxelHashing (VH) [8], BundleFusion (BF) [11], StaticFusion (SF) [31], ReFusion (RF) [22], CoFusion (CF) [14], MaskFusion (MF) [37], EM-Fusion (EM) [39], and MID-Fusion (MID) [38] on both synthetic and real-world datasets. We use the above abbreviation of comparison throughout the rest of this chapter.

To evaluate with the prior-based approach [37], we fine-tune MaskRCNN on our DYN SYNTH dataset. Our trained model reaches 70.2 mean average precision (mAP) on instance segmentation task, which shows the model is capable of providing reasonable segmentation input. For real-world data, we use the pre-trained model on the CoCO dataset [125] and manually select the semantic class of moving objects for each testing example.

5.3.1 Computational Time and Implementation Details

We measure the computational time of RigidFusion for processing a single frame in each module, see Figure 5.3, as follows - Human detection: 0.38s. BG tracking and model updating: 0.07s. FG tracking and model updating: 0.10s. Segment-by-Reconstruction takes: 0.44s. Due to the asynchronous process, see Figure 5.2, our system has $(0.38 + 0.07) \times \Delta$ seconds delay at the beginning (including human detection and background tracking). Each frame contains an active foreground object takes around one second $(0.38 + 0.07 \times 2 + 0.44)$ to process. The post-process step in Section 5.2.4 takes 0.45 second for each frame. We summarize the system parameters in Appendix B. All the experiments are performed on a desktop machine with Intel Core i7-6700K 4.00 GHz CPU, 32GB memory, and GTX 1080Ti GPU.
5.3. Evaluations and Results

![Figure 5.5: Qualitative evaluation on DYNSYNTH.](image)

Note that the moving object is positioned at the first detected frame. In contrast to ours, both CoFusion and MaskFusion result in ghosting in the background due to delayed moving object detection.

Table 5.1: Quantitative evaluation of the background reconstruction and tracking on DYNSYNTH. Our method shows the best reconstruction quality and tracking accuracy.

<table>
<thead>
<tr>
<th>Object Priors</th>
<th>Background Reconstruction</th>
<th>Tracking</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F1</td>
<td>R</td>
</tr>
<tr>
<td>VH [8]</td>
<td>0.58</td>
<td>0.66</td>
</tr>
<tr>
<td>BF [11]</td>
<td>0.44</td>
<td>0.61</td>
</tr>
<tr>
<td>RF [22]</td>
<td>0.64</td>
<td>0.67</td>
</tr>
<tr>
<td>CF [14]</td>
<td>0.73</td>
<td>0.72</td>
</tr>
<tr>
<td>MF [37]</td>
<td>yes</td>
<td>0.67</td>
</tr>
<tr>
<td>Ours w/o PS</td>
<td>no</td>
<td>0.74</td>
</tr>
<tr>
<td>Ours</td>
<td>no</td>
<td><strong>0.86</strong></td>
</tr>
</tbody>
</table>

5.3.2 Evaluation on DYNSYNTH Test Set

In Table 5.1, Table 5.2, and Figure 5.5, we conduct a quantitative and qualitative evaluation on our synthetic dataset along the two main axes: reconstruction and tracking, as introduced in Section 3.4. The employed metrics measure the accuracy and the completeness of the reconstruction as well as the tracking performance.

**Background Tracking and Reconstruction.** In Table 5.1, we evaluate different methods’ background tracking and reconstruction performance. VoxelHashing [8] and BundleFusion [11] are designed for static scenes, which can be seen as a performance reference. Their reconstruction and tracking performance are low due to the existence of dynamic elements.

ReFusion [22] is a robust background reconstruction method and reaches a F1 of 0.63 on background reconstruction and a MOTA of 66% on camera tracking. ReFusion averages out foreground voxels in the TSDF model by additionally integrating free-space voxels and removes high residual pixels using a flood-filling
5.3. Evaluations and Results

Table 5.2: Quantitative evaluation of the foreground reconstruction and tracking on DYN SYNTH. Overall, our method achieves a significantly lower miss-detection ratio and better recall/outlier balance than the other approaches. Symbol ‘-’ denotes that the methods do not handle dynamic objects. Note that we do not consider identity switches here because it is designed for re-identification instead of acquiring a canonical model.

<table>
<thead>
<tr>
<th>Object Priors</th>
<th>F1</th>
<th>R</th>
<th>P</th>
<th>CD (m)</th>
<th>MOTA (%)</th>
<th>MISS (%)</th>
<th>MOTP (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VH [8]</td>
<td>no</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BF [11]</td>
<td>no</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RF [22]</td>
<td>no</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CF [14]</td>
<td>no</td>
<td>0.38</td>
<td>0.27</td>
<td>0.70</td>
<td>0.26</td>
<td>32</td>
<td>39</td>
</tr>
<tr>
<td>MF [37]</td>
<td>yes</td>
<td>0.53</td>
<td>0.47</td>
<td>0.70</td>
<td>0.29</td>
<td>33</td>
<td>50</td>
</tr>
<tr>
<td>Ours w/o PS</td>
<td>no</td>
<td>0.51</td>
<td>0.41</td>
<td>0.74</td>
<td>0.31</td>
<td>55</td>
<td>24</td>
</tr>
<tr>
<td>Ours</td>
<td>no</td>
<td>0.56</td>
<td>0.44</td>
<td>0.88</td>
<td>0.13</td>
<td>59</td>
<td>13</td>
</tr>
</tbody>
</table>

algorithm. However, its performance is still affected by pixels from the foreground.

CoFusion [14] and MaskFusion [37] are state-of-the-art methods based on ElasticFusion [83]. Interestingly, their MOTA for background tracking is 16-19% lower than ReFusion. We found this is due to two reasons: The false positives of the moving object detection remove many geometry signals for camera tracking. And the weighting mechanism adapted from ElasticFusion prefers not to use newly observed points.

RigidFusion achieves the best F1 (0.74) and MOTA (68%) on the background tasks without performing the post-processing step (w/o PS, Section 5.2.4). This is benefited from our aggressive outlier purging strategy (Section 5.2.2, free-space aware fusion). With the post-processing step, our system’s performance can be further enhanced by 0.12 on background reconstruction F1 and 2% on background tracking MOTA.

Foreground Detection, Tracking, and Reconstruction. In Table 5.2, we evaluate RigidFusion’s foreground reconstruction, detection, and tracking accuracy against other methods. To reconstruct foreground objects, moving object detection is essential. This step is non-trivial because too many false positive detections not only increase the computation costs but also remove the geometry signals for background
Both CoFusion [14] and MaskFusion [37] detect a new foreground object by examining the estimated camera motion residuals. This approach’s major disadvantage is being sensitive to the actual camera motion (moving direction and magnitude) and the selection of the detection threshold. This causes a high missing detection ratio (MISS) and makes these two methods require a static or slow camera setup. This observation is also reported in the previous work [31].

MaskFusion utilizes a very good segmentation prior (73 mAP) and hence achieves a higher recall and F1 on foreground reconstruction. However, semantic segmentation does not directly solve motion segmentation. When motion segmentation is the same as semantic segmentation input, such as TUM f3w sequences in Table 5.5 (major moving objects are humans), MaskFusion has better background tracking than CoFusion and achieves competitive performance as ReFusion [22]. Inversely, when motion segmentation is non-trivial, i.e., input masks may contain static objects, such as the examples shown in Table 5.2 and Figure 5.5, MaskFusion does not have performance gains on object tracking. Notably, RigidFusion’s foreground miss detection is significantly lower than the comparisons (15% and 26% lower than CoFusion and MaskFusion). This is benefited by employing the segment-by-reconstruction strategy, which is insensitive to the relative motion between the moving object and the camera. Hence, RigidFusion can detect foreground quickly when both foreground and background have non-static movements. With the post-process step (Section 5.2.4), the reconstruction F1 and MOTA can be enhanced by 0.05 and 4%, respectively. In Figure 5.5, we show a qualitative comparison.

5.3.3 Evaluation on Real-world Data

Scenes with different levels of camera motion. We recorded four dynamic scenes with different settings, including a small desktop scene with a near-static camera and medium-scale scenes with non-static camera motions, using a Structure Sensor developed by Occipital Inc. mounted to an iPad AIR2. The scene settings are summarized in Table 5.3. We rank the ambiguity of the camera motion from low to high using the estimated camera trajectories and the annotated foreground masks on
the frame where a dynamic object appears or starts to move. In addition, to achieve a fair comparison, we passed to other methods input frames with the human regions masked out.

In Figure 5.6, we compare RigidFusion against CoFusion [14] and MaskFusion [37]. In Table 5.4, we show the frame index of the foreground being detected by each method and the corresponding delay frame number. The results of Scene 1 show when the camera motion is very small, both CoFusion and MaskFusion can detect foreground objects within a short delay. When a camera has non-static motion, as in Scene 2 to Scene 4, both CoFusion and MaskFusion suffer from long-delayed detection causing severe errors in tracking and reconstruction. This observation coincides with the evaluation results in Table 5.2, where both CoFusion and MaskFusion suffer from high MISS ratios. RigidFusion produces significantly better reconstruction results in all four examples, outputs visually completed object trajectories, and maintains low detection latency, as shown in Figure 5.6.

**Camera Tracking Accuracy.** In Table 5.5, we carry out an evaluation on camera tracking in dynamic scenes using the TUM RGB-D dataset, freiburg3, and report ATE-RMSE. Note that this dataset contains many far-range pixels. RigidFusion’s camera tracking is comparable to alternate methods, especially in high-dynamic walking examples, although not the best.

**What if the camera tracking fails?** In this case, our method and all comparison [22, 14, 37, 39, 38] will fail to produce a reasonable reconstruction result since camera tracking (or reconstruction) is used for identifying foreground objects.

**Does our method mainly benefit from having the foreground movement assumption?** While the foreground movement assumption (i.e., objects do not move simultaneously) helps to simplify the instance segmentation step, the evaluation results in Section 5.3.2 and Section 5.3.3 show the major limitation of previous work is the moving object detection step in scenes with moving cameras, i.e., when to initialize the first foreground tracking and which pixels to group. The multiple object association-and-tracking problem comes after a system having more than one tracked foreground object. From Table 5.4 and Figure 5.6, we can see a long-delayed
5.3. Evaluations and Results

Figure 5.6: Qualitative evaluation on real-world data. In each scene, the first row: full 4D reconstructions; the second row: foreground trajectories and reconstruction results. The scene settings are analyzed in Table 5.3. The scenes are ordered based on camera motion and the scene size. Our method can work on both low-dynamic and high-dynamic settings and output plausible results.
5.4. Discussion and Limitations

detection, such as CoFusion’s result on Scene3, produces a noisy and rather partial reconstruction.

5.3.4 Evaluation on CoFusion Dataset

In Table 5.6, we evaluate RigidFusion on the CoFusion’s synthetic data [14] and report ATE-RMSE. For the Airship object in Room4, both MaskFusion [37] and EMFusion [39] fail to associate segments across frames and output fragmented trajectories. Compared to the other methods, RigidFusion consistently produces better camera tracking and improves foreground tracking in Room4 sequence without using object priors.

Tracking methods versus moving object detection methods. Table 5.6 shows that the accuracy of the moving object detection step is more important than the underlay tracking methods. Note that both CoFusion and MaskFusion use surfel tracking [83], while EMFusion uses volumetric-based tracking but employs a similar moving object detection step as MaskFusion using semantic priors. EMFusion utilizes its volumetric model to improve semantic segmentation. Therefore its object tracking is better than MaskFusion. However, EMFusion and MaskFusion still have similar error patterns in camera tracking, and both have higher ATE-RMSE than CoFusion. In contrast, RigidFusion employs the segment-by-reconstruction strategy and volumetric-based tracking and achieves the best-in-class camera tracking performance.

What if our foreground movement assumption does not hold? RigidFusion assumes foreground objects are not moving simultaneously. Our method will only track the dominant foreground object if this assumption does not hold. One example is the ToyCar3 sequence in Table 5.6. where the two cars move simultaneously, and Car1 is the dominant foreground object.

5.4 Discussion and Limitations

In this Chapter, we develop an unsupervised reconstruction system for acquiring both rigidly-moving objects and background. We demonstrate that our system can successfully reconstruct objects and scene geometry in highly-dynamic scenes, which was particularly challenging for the state-of-the-art methods.
Table 5.3: The analysis of our real-world examples. Each scene contains a different level of dynamic, ordered from low to high. The motion residuals is calculated using the estimated camera trajectory by running ReFusion [22]. When the camera motion is large, the motion residuals become unreliable for segmenting moving objects.

<table>
<thead>
<tr>
<th>Scene size</th>
<th>Scene1 (two objects)</th>
<th>Scene2 (two objects)</th>
<th>Scene3 (one object)</th>
<th>Scene4 (one object)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4x0.6 m²</td>
<td>4.5x3.0 m²</td>
<td>4.3x2.2 m²</td>
<td>5.9x4.5 m²</td>
<td></td>
</tr>
<tr>
<td>Camera motion</td>
<td>2.7 cm/s</td>
<td>25.3 cm/s</td>
<td>10.5 cm/s</td>
<td>39.2 cm/s</td>
</tr>
<tr>
<td>BG’s median motion residuals</td>
<td>4.32E-04</td>
<td>9.65E-05</td>
<td>1.22E-04</td>
<td>4.85E-04</td>
</tr>
<tr>
<td>FG’s median motion residuals</td>
<td>9.70E-01</td>
<td>1.01E-04</td>
<td>1.43E-04</td>
<td>7.87E-04</td>
</tr>
<tr>
<td>Motion ambiguity</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>First dynamic frame / total frame #</td>
<td>11 / 350</td>
<td>134 / 635</td>
<td>26 / 270</td>
<td>1 / 209</td>
</tr>
</tbody>
</table>

Table 5.4: The evaluation of moving object detection. We show the number of delayed detection frames (Delay #) using the output trajectories. Clearly, RigidFusion has the lowest detection latency. Symbol ‘n/a’ represents the object is not detected, and '-' indicates there are no object 2.

| FG ID | Detection Methods | Object1 | | Object2 | |
|-------|-------------------|---------|-----------------|-----------------|
|       | Scene1 (two objects) | Scene2 (two objects) | Scene3 (one object) | Scene4 (one object) |
|       | Delay # | Delay # | Delay # | Delay # | Delay # |
|       |       |       |       |       |       |
|       |       |       |       |       |       |
|       |       |       |       |       |       |
|       | CF [14] | 11 | 27 | 145 | 144 |
|       | MF [37] | 11 | 12 | 120 | 21 |
|       | Ours w/o PS | 3 | 10 | 5 | 0 |
|       | Ours | 3 | 0 | 0 | 0 |
|       | CF [14] | 3 | n/a | - | - |
|       | MF [37] | 7 | 0 | - | - |
|       | Ours w/o PS | 4 | 6 | - | - |
|       | Ours | 3 | 0 | - | - |
5.4. Discussion and Limitations

Table 5.5: Background tracking evaluation on TUM RGB-D dataset. Our method has a slight performance drop due to our free-space aware fusion step remove some geometry features in the latter frames. f3s and f3w represent sitting and walking cases in freiburg3 respectively.

<table>
<thead>
<tr>
<th>Methods</th>
<th>Human Priors</th>
<th>Object Priors</th>
<th>f3s static</th>
<th>f3s xyz</th>
<th>f3s halfsphere</th>
<th>f3w static</th>
<th>f3w xyz</th>
<th>f3w halfsphere</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF [31]</td>
<td>no</td>
<td>no</td>
<td>1.3</td>
<td>4.0</td>
<td>4.0</td>
<td>1.4</td>
<td>12.7</td>
<td>39.1</td>
</tr>
<tr>
<td>RF [22]</td>
<td>no</td>
<td>no</td>
<td>0.9</td>
<td>4.0</td>
<td>11.0</td>
<td>1.7</td>
<td>9.9</td>
<td>10.4</td>
</tr>
<tr>
<td>CF [14]</td>
<td>no</td>
<td>no</td>
<td>1.1</td>
<td>2.7</td>
<td>3.7</td>
<td>55.1</td>
<td>69.6</td>
<td>80.3</td>
</tr>
<tr>
<td>MF [37]</td>
<td>yes</td>
<td>yes</td>
<td>2.1</td>
<td>3.1</td>
<td>5.2</td>
<td>3.5</td>
<td>10.4</td>
<td>10.6</td>
</tr>
<tr>
<td>MID [38]</td>
<td>yes</td>
<td>yes</td>
<td>1.0</td>
<td>6.2</td>
<td>3.1</td>
<td>2.3</td>
<td>6.8</td>
<td>3.8</td>
</tr>
<tr>
<td>EM [39]</td>
<td>yes</td>
<td>yes</td>
<td>0.9</td>
<td>3.7</td>
<td>3.2</td>
<td>1.4</td>
<td>6.6</td>
<td>5.1</td>
</tr>
<tr>
<td>Ours</td>
<td>yes</td>
<td>no</td>
<td>1.9</td>
<td>5.4</td>
<td>12.9</td>
<td>1.8</td>
<td>9.0</td>
<td>7.6</td>
</tr>
</tbody>
</table>

Table 5.6: Foreground tracking evaluation on CoFusion's dataset. Symbol ‘†’ represents the corresponding moving object is not detected. MaskFusion does not detect Horse due to the delay detection and camera tracking drift, as reported in [39]. Our method does not detect Car2 and Horse due to they simultaneously move with other objects, but our approach achieves the best tracking accuracy on Airship, Car, and Camera.

<table>
<thead>
<tr>
<th>AT-RMSE (in cm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToyCar3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object Priors</th>
<th>Camera</th>
<th>Car1</th>
<th>Car2</th>
<th>Camera</th>
<th>Car</th>
<th>Airship</th>
<th>Horse</th>
</tr>
</thead>
<tbody>
<tr>
<td>CF [14]</td>
<td>no</td>
<td>0.61</td>
<td>7.78</td>
<td>1.44</td>
<td>0.93</td>
<td>0.29</td>
<td>0.96</td>
</tr>
<tr>
<td>MF [37]</td>
<td>yes</td>
<td>20.6</td>
<td>1.53</td>
<td>0.58</td>
<td>1.41</td>
<td>2.66</td>
<td>6.46</td>
</tr>
<tr>
<td>EM [39]</td>
<td>yes</td>
<td>0.95</td>
<td>0.77</td>
<td>0.18</td>
<td>1.37</td>
<td>2.1</td>
<td>0.91</td>
</tr>
<tr>
<td>Ours</td>
<td>no</td>
<td>0.46</td>
<td>1.9</td>
<td>†</td>
<td>0.58</td>
<td>1.0</td>
<td>0.55</td>
</tr>
</tbody>
</table>
5.4. Discussion and Limitations

Compared to Chapter 4, our method does not require pre-trained object priors, and our segmentation-by-reconstruction strategy is more reliable for unseen objects. On the other hand, for the common objects with lower variance in appearance and structure, such as books and boxes, the supervised approach may exhibit higher performance.

Our approach also has some limitations, as discussed in the following:

• **Only allows one active object at a time.** Although our method requires this assumption to simplify the foreground detection step, we demonstrated in Section 5.3.3 that our method can still successfully handle several real-world examples since many dynamic events only have a single moving object at a time. We believe this assumption can be relaxed by incorporating dense tracking priors, a visual tracker, or a global optimization step to estimate the number of objects. These ideas can be combined with TSDF representation and formulate a robust multi-object SLAM solution.

• **Object re-identification.** Similar to Chapter 4, we do not handle object re-identification, where an object may re-enter the scene or switch its motion status (static or dynamic) multiple times. We believe this requires an efficient scene representation and possibly can be combed with online learning methods to build object identification at run-time.

• **Large run time.** As mentioned in Section 5.3, the major bottlenecks of our system are the human detector [148], which we believe can be improved by adapting a real-time human detector, and the segment-by-reconstruction step, which can be sped up by using parallel processing on GPU as the volumetric fusion step.

• **Model size.** We employ sparse voxel hashing to represent background and object geometry on a commodity Nvidia GeForce 1080Ti GPU. The maximum number of objects we can support is four in a medium size room. The tricky part for memory management is that the object association step requires all
active foreground models to be alive on GPU. Further compressing model size using neural networks or designing a novel object association algorithm will be interesting future work.
Chapter 6

Neural Dynamic Scene Representation

6.1 Introduction

Dynamic scene reconstruction and understanding from video capture have a long history in content creation; it subsequently enables editing by replaying the content from novel viewpoints and allowing object-level modifications. The task is particularly challenging in the dynamic context of moving and deforming objects when observed through a moving (monocular) camera due to occlusion and the large parameter space of non-rigid motions. Traditional approaches make simplifications by assuming the scene to be static [4], focusing on a single deformable object [15, 16, 18, 36], or requiring access to a variety of priors in the form of object templates [46, 47], or deformable object models [43, 72, 152].

Neural Radiance Field (NeRF) [103], a recently developed neural representation, has provided a breakthrough in terms of producing highly photo-realistic (static) representation, simultaneously capturing geometry and appearance from only a set of posed images. A substantial body of work has rapidly emerged to extend the formulation to dynamic settings [86, 87, 88, 89, 94, 153, 154], work with localized representations for real-time inference [155, 156, 157, 158, 159, 160, 161, 162], support fast training [163, 159, 160, 161, 164], and investigate applications in the context of generative models [165]. However, the representations often lack inter-
Figure 6.1: Factored neural representation. We present an algorithm that directly factorizes raw RGB-D monocular video [25] to produce object-level neural representation with explicit motion trajectories and possibly deformation information. The decoupling subsequently enables different object manipulation and novel view synthesis applications to produce authored videos. We do not use any template but instead use an end-to-end optimization to enable factorization. Note that the human deforms/moves in this sequence.

pretability, require multi-view input, fail to provide scene understanding, and do not provide object-level factorization or enable object-level scene manipulation.

We introduce factored neural representation (FactoredNeRF), a novel scene representation using neural volumetric rendering that supports object-level geometry reconstruction (including both rigid and non-rigid objects), interpretability, and editability while still capturing appearance details under object movement and viewpoint changes. Compared to the recently developed neural dynamic scene representations [166, 167, 92] that support appearance and view changes, our scene representation supports not only novel view rendering but also outputs individual object geometry instead of a single time-dependent foreground model. Further, our approach does not require any object template, deformation prior, or pre-training object NeRFs. Starting from an RGB-D monocular video of a dynamic scene, we demonstrate how such a factored neural representation can be robustly extracted via joint optimization by leveraging off-the-shelf image-space segmentation and tracking information. Factorization is provided in the form of object-level neural representations, as well as object trajectory and/or deformations.

Technically, we formulate a global optimization to simultaneously build and track per-object neural representations along with a background model while solving for object trajectories and a camera path. Further, we model deformable bodies
6.2. Image Formation Model

Our proposed representation combines the advantages of object-centric representations and motion tracking, thereby allowing per-object manipulation, without having to pay the overhead of separately building object priors or requiring 3D supervision, and naturally integrates information from a monocular input over time across the neural representations to recover from occlusion. For example, Figure 6.1 shows a factored representation obtained by our method, which operates on a monocular RGB-D sequence [25] of 60 frames, along with some edits.

6.2 Image Formation Model

Before introducing the optimization formulation in Section 6.3, we present our image formation model to produce a rendered image from a factored neural representation.

6.2.1 Volume Rendering

To render an image $I$ from the given camera parameters $\Pi$ (i.e., intrinsic parameters), volume rendering [103, 122] maps each image pixel $uv$ to form a camera ray $r$. Points are sampled on each ray and sorted based on their depth values to produce a rendered color $C(r)$ as the integration of the sampled point colors $\{c_i\}$ weighted by the corresponding point opacity $\{\alpha_i\}$ and transmittance $\{T_i\}$. Note that samples along a ray $r := (o,d)$, going through point $o$ along the direction $d$, are parameterized as $p(s_i, r) := o + s_i d$ for increasing scalar samples $s_i \in \mathbb{R}^+$. Using the samples, we discretize the continuous formulation using the quadrature approximation as:

$$C(r) := \sum_i T_i \alpha_i c_i$$
$$T_i := \prod_{j=1}^{i-1} (1 - \alpha_i)$$
$$\alpha_i := 1 - \exp(-\sigma_i \delta_i),$$

(6.1)

where $\sigma_i$ is the point density, and $\delta_i$ is the depth distance between two adjacent samples. Recall that point opacity $\alpha_i$ represents the probability of a ray stopping at the point position $p(s_i)$, while the transmittance $T_i$ indicates the cumulative transmittance.
before a ray hits the $i$-th sample point. Looping over all image pixels $\{uv\}$, we obtain 
\[ I := \mathcal{R}(\Pi, f_\theta, \{r_{uv}\}), \]
where the function $f_\theta$, typically modeled by an MLP [103], can be probed to produce density and color samples as $f_\theta(p(s_i), r) := (\sigma_i, c_i)$. Note that only the color values are view-dependent.

### 6.2.2 Volume Rendering with Implicit Surface

Implicit surface representation, such as occupancy or signed distance field, can also be used with volume rendering [123, 124, 168] and provides an inductive bias for modeling surface geometry. We found this to be more suitable for object-level factored representation as we can easily regularize the optimization to encode object surfaces, instead of producing volumetric clouds. Here, we employ the signed distance field formulation proposed by Wang et al. [123] and convert the signed distance value $\psi$ to the opacity values by assigning non-zero values near the zero level set of the modeled surface geometry as:

\[ \alpha_j := \max \left( \frac{\Phi(\psi_j) - \Phi(\psi_{j+1})}{\Phi(\psi_j)}, 0 \right), \tag{6.2} \]

**Figure 6.2: Rendering neural factored representation.** Given a factored representation $\mathcal{F}\{ (f^i, \psi^i, B_i, T_i) \}_{i=0}^k$ and any query ray $r$ from the current camera, we first intersect each object’s bounding box $B_i$ to obtain a sampling range and then compute a uniform sampling for each of the intervals. For each such sample $p$, we lookup feature attributes by re-indexing using local coordinate $T_j^{-1} p$, resort the samples across the different objects based on (sample) depth values, and then volume render to get a rendered attribute. Background is modeled as the 0-th object. See Section 6.2 for details. For objects with active non-rigid flags, we also invoke the corresponding deformation block (see Section 6.3). The neural representations and the volume rendering functions are jointly trained.
6.2. Image Formation Model

Figure 6.3: Pipeline. Starting from a monocular RGB-D sequence \( \{I(t)\} \), we extract a factored neural representation \( F \) that contains separate neural models for the background and each of the moving objects along with their trajectories. For any object tagged as non-rigid, we also optimize a corresponding deformation block (e.g., human). First, in an initialization phase, we assume access to keyframe annotations (segmentation and AABBs) over time, propagate them to neighboring frames via dense visual tracking and optical flow, and estimate object trajectories. Then, we perform end-to-end optimization using a customized neural volume rendering block. The factored representation enables a variety of applications involving novel view synthesis and object manipulations.

where we use a shorthand \( \psi_j := \psi(p(s_j)) \) for the \( j \)-th sample and \( \Phi \) is the Sigmoid function. Here, we represent the rendering function as \( I := R(\Pi, f_\theta, \psi, \{r_{uv}\}) \), where the function \( f_\theta \) again can be probed to produce only view dependent color samples \( f_\theta(p(s_i), \Pi) := c_i \) and \( \psi \) represents the learned SDF function.

6.2.3 Attributes Rendering

By replacing point color \( c_i \) with any other attribute \( a_i \), such as depth [88, 34] or semantic labels [169], volumetric rendering can be generalized to render depth or semantic segmentation, respectively. Specifically, for any attribute \( a_i \) and ray \( r \), we simply compute an attribute at a pixel as \( A(r) := \sum_i T_i a_i \).
6.2.4 Volume Rendering with Factored Neural Representation

Our proposed factored representation $F := \{(f^i, \psi^i, B_i, T_i)_{i=0}^k\}$ for a background model $f^0$ and the foreground objects $\{f^i, i \in [1,k]\}$, which can be probed to output density and color attributes. Each model, the background or any foreground object, can be probed to output color attributes with corresponding AABB (axis-aligned bounding boxes) $\{B_i\}$, transformations $\{T_i\}$ to map the AABB local coordinates to the global coordinate system, and implicit SDF functions $\{\psi^i\}$ to produce density samples. We now define the rendering function $I := R(\Pi, F, \{r_{uw}\})$ using our factored representation $F$. Figure 6.2 illustrates the process. For each ray $r$, for each intersected model, computed using its AABB $B_i$, we obtain SDF density values using uniform samples and perform inverse transform sampling to generate 128 samples per ray. For each background ($i = 0$) or foreground ($i \in [1,k]$) sample, we obtain sampled color $f^i(T_i^{-1}p^i(s_j), r) := c_j^i$ and opacity $\alpha_j^i$ using Equation 6.2 with $\psi^i$ and the remapped sample $T_i^{-1}p^i(s_j)$, expressed in the local coordinate system of the object. We collect the samples across the background and all the intersecting objects, sort the samples based on their depth values, and render the colors/attributes as described earlier (see Equation 6.1).

6.3 Method

As input, we take in RGB-D frames, denoted by $\{I(t) := (C_t, D_t)\}$ with color $C$ and depth $D$ frames at time $t$, of scenes with one or more moving objects, where objects can be moving rigidly or non-rigidly (e.g., humans). We assume access to keyframe annotations over time, containing instance segmentation and rigid/non-rigid flags. This information is used, in an initialization step, to extract initial camera and object trajectories, instance masks over time, and axis-aligned bounding boxes (AABBs) in the camera space. As output, we produce a factored neural representation $F$ of the scene, where for each object we produce a neural representation along with its estimated object trajectory, and for each non-rigid object also an associated deformation function. In Section 6.4, we use these inferred factored representations to directly render novel view synthesis or perform object-level manipulations.
In order to obtain such a factored representation, we have to address several challenges. First, the extracted segmentation information from the RGB-D frames is imperfect, and hence any information or supervision (e.g., segmentation loss) derived from them leads to error accumulation. Second, we need to recover from artifacts in initial pose estimation, especially in scenes with insufficient amount of textures to guide the camera calibration stage. Auto-focus, color corrections, and error accumulation in real captures pose further challenges. Third, since we only use monocular input, the input provides partial information in the presence of occlusion, both in terms of shape and appearance, and, in the absence of any priors, we have to recover from the missing information by fusing information across the (available) frames. Finally, we allow objects to exhibit non-rigid motion (e.g., human walking) and have to factorize object deformation from object motion. In the following, we present how to set up a joint optimization, with suitable initialization and regularizes, involving object deformation, neural representations, and volume rendering to solve these challenges.

### 6.3.1 Initialization

We first use an off-the-shelf visual tracker [170] with keyframe annotations, including instance segmentation and AABBs, to propagate the segmentation across the frames. The keyframe annotations are annotated by users. Segmentation is annotated at every ten frames because the visual tracker may fail for long-term tracking. AABBs are only annotated at the first frame and can be overlapped between objects or be slightly larger than the actual object size. In order to get an initial registration, we run a state-of-the-art optical flow network [171] to find initial correspondences and solve for frame-to-frame rigid alignment using the iterated closest point (ICP) approach [85]. The registration information across frames provides object trajectory \( \{T_i(t)\} \) estimates.

### 6.3.2 Joint Optimization

We now introduce the main loss terms to capture reconstruction quality and additional regularizers to get a desired factored representation. We first perform balanced
6.3. Method

sampling for each object using the inferred segmentation from the initialization step and sample $P$ pixels from $N$ RGB-D frames. This prevents the background object from dominating the whole loss term.

**Reconstruction loss:** We then render color and depth attributes using the current (multi-object) neural factored representation as described in Section 6.2. Note that the ground truth attributes are indexed by the sampled ray $r$ and frame time $t$. We compare the sampled color $C$ and depth $D$ attributes against the estimated attributes using the L1 reconstruction loss, i.e.,

$$L_{\text{color}}(\mathcal{F}) := \sum_{(r, t) \in P} |C_t(r) - R_C(\Pi, \mathcal{F}(t), r)| / |P|$$

and

$$L_{\text{depth}}(\mathcal{F}) := \sum_{(r, t) \in P} |D_t(r) - R_D(\Pi, \mathcal{F}(t), r)| / |P|,$$

where render the current background and foreground neural objects at time $t$, i.e.,

$$\mathcal{F}(t) := \{(f^i, \psi^i, B_i, T_i(t))_{k=0}^k\},$$

to produce RGB and depth attributes, and sum up over the sampled pixels.

**Free-space loss:** In order to check the factorization quality, one approach is to compare the predicted object segmentation, computed using the current re-projection of objects’ transmission, against the input segmentation. However, we found this approach leads to poor results as segmentation estimates are noisy. Instead, we focus on the complement space and define a free-space loss to penalize density values in regions that are indicated to be free according to the raw depth information. For any point sample derived from $P$, we identify free-space samples using depth $I^D$. Then, we constrain the integrated weights of each free-space sample $p \in P_{\text{free}}$, before reaching the object surface, to be zero using L1 loss. Specifically,

$$L_{\text{free}}(\mathcal{F}) := \sum_{p \in P_{\text{free}}} |T_p \alpha_p| / |P_{\text{free}}|$$

where $P_{\text{free}} = \{p(s,r) | s < D_t(r)\}$.  

(6.4)

**Non-rigid Deformation:** In order to handle non-rigid objects, we additionally incorporate a deformation block for objects marked with non-rigid flags. To achieve
Figure 6.4: Dataset. We test on a mix of synthetic and real RGB-D monocular captures from the BEHAVE [25] dataset. Here we show RGB (top) and depth (bottom) representative frames.

this, we employ a state-of-the-art bijective deformation network proposed by Cai et al. [36], which consists of three sub-networks, each predicting a low-dimensional deformation. Given an input 3D point, each sub-network selects one axis, predicts a 1D displacement, and infers a 2D translation and rotation for the other axes. These sub-networks are sequentially invoked in the XYZ axis order. Note that this block gets directly optimized via the reconstruction loss and is not supervised with any intermediate data.

**Surface Regularizers:** In order to regularize our network to learn object geometry as a shared model between frames (i.e., a canonical model) instead of arbitrary reconstruction, we employ auxiliary losses to constrain our geometry models to be actual surfaces. We achieve this by penalizing the implicit functions $\psi^i$ to (i) be a true signed distance field (i.e., Eikonal loss) by constraining random points in the bounding box $B_i$; (ii) requiring the surface samples to have normals in the direction of normals $\mathbf{n}(\mathbf{x})$ estimated from the input RGB-D frames; and (iii) surface samples to have zero implicit values. These auxiliary losses do not slow down the optimization since surface samples can be obtained from input depth, and these losses can be directly calculated without performing volumetric rendering. Putting them together,
we get,

\[
L_{\text{surface}}(\mathcal{F}) := \sum_{i \in [0, k]} \frac{1}{(k+1)} \left[ \sum_{x \in P_{B_i}} \| \nabla \psi^i(x) - 1 \| / |P_{B_i}| + \sum_{x \in P_{\Omega_i}} (1 - \langle \nabla \psi^i(x), n(x) \rangle) / |P_{\Omega_i}| + \sum_{x \in P_{\Omega_i}} |\psi^i| / |P_{\Omega_i}| \right], \quad (6.5)
\]

where \( P_{B_i} \) and \( P_{\Omega_i} \) denote the randomly sampled spatial points and surface samples in the object bounding box, respectively. Finally, we arrive at the full optimization problem as,

\[
\min_{\mathcal{F}} L_{\text{total}}(\mathcal{F}) := L_{\text{color}}(\mathcal{F}) + \lambda_1 L_{\text{depth}}(\mathcal{F}) + \lambda_2 L_{\text{free}}(\mathcal{F}) + \lambda_3 L_{\text{surface}}(\mathcal{F}), \quad (6.6)
\]

where \( \lambda_1 < 1 \) due to noisy depth input. We use \( \lambda_1 = 0.1, \lambda_2 = 1.0, \) and \( \lambda_3 = 0.1 \) in our experiments.

### 6.4 Evaluation

We evaluated Factored Neural Representations on a variety of synthetic and real scenes in the presence of rigid and non-rigid objects. In each case, we start with only RGB-D sequences without access to any geometry or motion prior.

#### 6.4.1 Datasets

We tested on two types of datasets, synthetic and real. As *synthetic data*, we propose a new dataset using publicly available CAD models [41, 172] and render RGB-D sequences using Blender [172, 173]. To inject motion, we manually edit camera motion and rigid object motion, and we combine non-rigid motion from the DeformingThings4D [174] dataset. As representative examples, we present three sequences, SYN-SCENE-A, SYN-SCENE-B, and SYN-SCENE-C, each spanning for 90-100 frames, and simulate sensor noises using the noise model proposed.
6.4. Evaluation

Figure 6.5: Comparisons on our synthetic dataset. Visually comparing our results against iMAP [33] and NiceSLAM [34] on our synthetic dataset. See Table 6.1 for the quantitative evaluation and Figure 6.6 for object factorization. Please note that since both iMAP and NiceSLAM work for static scenes, for a comparison purpose, we provide the same segmentation initialization as ours (ground truth keyframes segmentation and the inferred segmentation using SiamMask [170] for the other frames); and we run the methods multiple times, once for each object, and generate full rendering using their predicted depth. Our results capture finer object geometry and do not affect by the imperfect object segmentation (e.g., the handle of the green bag). In the validation view of Scene C, we observed that large non-rigid deformation is quite challenging for the network to learn accurate geometry showing some over-fitting happened.
6.4. Evaluation

Figure 6.6: Comparisons of the object reconstruction on our synthetic dataset. Visually comparing our results against iMAP [33] and NiceSLAM [34] on our synthetic dataset. See Table 6.1 for quantitative evaluation. For comparison, we provide the same segmentation initialization as ours. Note that the other methods do not support reconstructing the non-rigidly moving human. Our object reconstruction is not affected by the noisy segmentation boundary and captures appearance details (e.g., shading on the yellow monkey face). Our results also show a better visual quality of novel view rendering, although our method doesn’t factorize out some floor regions in the Human objects due to the textureless issue.

by [175]. For these sequences, we have access to ground truth data (e.g., object segmentation, validation views) for conducting the per-object evaluation.

As real dataset, we use the BEHAVE [25] dataset, which provides human object interaction RGB-D videos with keyframe annotation. We crop and evaluate the first non-occlusion sequence in each scene to avoid the object re-identification issue. Figure 6.4 shows some representative frames.
6.4. Comparison

We compare our approach against different competing alternatives. Existing monocular approaches can be categorized as either employing an MLP (e.g., iMAP [33]), or using multi-resolution feature grids (e.g., NiceSLAM [34]). Since these competing methods do not support jointly optimizing multiple objects, we use the segmentation predicted by the visual tracker [170] and manually run them multiple times to reconstruct background and dynamic objects. Note that we modified the ray sample step of iMAP and NiceSLAM to accept object segmentation input, and we use L1 segmentation loss [123, 168, 36] when training foreground models. For both iMAP and NiceSLAM, we employ the open-source network implementation [34] in our training framework instead of their multi-threads SLAM framework, which contains several optimizations (e.g., view-purging) for real-time applications. Furthermore, we do not optimize both camera and object poses for all comparison methods and focus on evaluating the reconstruction quality. We will discuss joint pose estimation and reconstruction in the future work section.

6.4.3 Evaluation Metrics

We compare different methods across a range of metrics. We evaluate reconstruction quality using PSNR, L1, and SSIM in Table 6.1 and Table 6.2. We also qualitatively evaluate object trajectory estimates and re-synthesis quality under the authoring of updated object trajectory and the visibility states of objects from the factored scenes in Figure 6.8 and Figure 6.9.

6.4.4 Quantitative Evaluation

We present quantitative comparisons using both synthetic and real-world data. In Table 6.1, we evaluate novel view rendering quality on our synthetic dataset, separately for RGB and depth channels. Notably, our method consistently outputs better reconstruction than others (iMAP and NiceSLAM), indicating that our method extracts a proper factorization and avoids overfitting to training views. In the absence of validation views and per-frame annotations, we cannot run a quantitative evaluation for real sequences. Therefore, we evaluate reconstruction quality using the training
Table 6.1: Reconstruction error on our synthetic dataset. (Top/Bottom) Quantitative color/depth results on validation poses. We evaluate novel view rendering using a validation camera. Ours largely produces better reconstruction, validating that our joint optimization captures better scene geometry. See Figure 6.5 and Figure 6.6 for qualitative evaluation.

<table>
<thead>
<tr>
<th></th>
<th>SYN-Scene A</th>
<th>SYN-Scene B</th>
<th>SYN-Scene C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PSNR↑/SSIM↑</td>
<td>PSNR↑/SSIM↑</td>
<td>PSNR↑/SSIM↑</td>
</tr>
<tr>
<td>iMAP</td>
<td>20.32/0.79</td>
<td>17.02/0.73</td>
<td>17.02/0.75</td>
</tr>
<tr>
<td></td>
<td>13.52/0.38</td>
<td>16.89/0.90</td>
<td>14.34/0.88</td>
</tr>
<tr>
<td></td>
<td>14.89/0.80</td>
<td>26.38/0.91</td>
<td>18.11/0.93</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>15.45/0.78</td>
<td>17.00/0.92</td>
</tr>
<tr>
<td>NiceSLAM</td>
<td>24.48/0.89</td>
<td>23.16/0.90</td>
<td>21.71/0.82</td>
</tr>
<tr>
<td></td>
<td>19.31/0.93</td>
<td>20.71/0.94</td>
<td>18.73/0.85</td>
</tr>
<tr>
<td></td>
<td>17.98/0.78</td>
<td>16.75/0.88</td>
<td>17.05/0.98</td>
</tr>
<tr>
<td></td>
<td>21.87/0.90</td>
<td>20.04/0.88</td>
<td>20.39/0.95</td>
</tr>
<tr>
<td></td>
<td>18.78/0.93</td>
<td>22.12/0.83</td>
<td>15.24/0.84</td>
</tr>
</tbody>
</table>

Figure 6.7: Comparisons of scene reconstruction on the BEHAVE dataset. Visually comparing our results against iMAP [33] and NiceSLAM [34] on the BEHAVE dataset using the training camera. For comparison, we provide the same segmentation initialization as ours. See Table 6.2 for quantitative evaluation. Notably, our method generalizes better when the scene contains large missing depth areas, showing the learned geometry model is constrained well (see the wall in the training views). Our method successfully captures non-rigid motion.
Table 6.2: Reconstruction error on BEHAVE using the training camera. We crop and evaluate the first non-occlusion sequence in each scene to avoid the object re-identification issue. We report total scene reconstruction errors using PSNR and SSIM due to the lack of per-frame annotation. Our method consistently produces better reconstruction quality benefiting from the proposed joint optimization and the deformation module. See Figure 6.7 for qualitative evaluation.

<table>
<thead>
<tr>
<th></th>
<th>PSNR ↑ / SSIM ↑</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Color Reconstruction, Training camera,</strong></td>
<td></td>
</tr>
<tr>
<td>tablesquare_move</td>
<td>14.68/0.66</td>
</tr>
<tr>
<td>trashbin</td>
<td>13.46/0.65</td>
</tr>
<tr>
<td>yogaball_play</td>
<td>12.42/0.64</td>
</tr>
<tr>
<td>chairblack_lift</td>
<td>13.28/0.64</td>
</tr>
<tr>
<td>iMAP</td>
<td>11.35/0.54</td>
</tr>
<tr>
<td>NiceSLAM</td>
<td>26.48/0.85</td>
</tr>
<tr>
<td>Our</td>
<td>27.75/0.87</td>
</tr>
<tr>
<td></td>
<td>28.03/0.87</td>
</tr>
<tr>
<td></td>
<td>26.71/0.85</td>
</tr>
</tbody>
</table>

| **Depth Reconstruction, Training camera,** |                  |
| tablesquare_move    | 24.84/0.31      |
| trashbin            | 21.67/0.49      |
| yogaball_play       | 22.25/0.43      |
| chairblack_lift     | 26.21/0.28      |
| iMAP                | 25.48/0.23      |
| NiceSLAM            | 30.06/0.14      |
| Our                 | 30.07/0.15      |
|                     | 30.29/0.13      |
|                     | 30.39/0.14      |

camera against the related methods in Table 6.2 and Figure 6.7. Again, our method achieves the best quality.

**Does joint optimization help reconstruction?** Our joint optimization allows the network to adjust object segmentation and sidestep the inaccurate segmentation issue introduced by the visual tracker [170]. This advantage is clearly demonstrated in Figures 6.5 and 6.6. The non-joint optimized methods (iMAP and NiceSLAM) failed to recover from imperfect segmentation. We also observed that a limitation of our method is to factorize the texture-less floor area in the object reconstruction, which may require additional priors, and left as future work.

### 6.4.5 Qualitative Evaluation

In Figure 6.5 and Figure 6.6, we qualitatively compare our method against alternative approaches (iMAP and NiceSLAM) using our synthetic data and the real-world BEHAVE dataset. Note that these approaches jointly optimize for scene geometry and appearance but assume the scenes to be static. In other words, these methods provide only partial factorization into rigid models and camera trajectories, and cannot be used for scene manipulation applications as supported by ours. Although these methods perform better in terms of modeling the background, they produce worse foreground object reconstruction and have a weaker generalization ability as
Table 6.3: Ablation study on our synthetic dataset. We evaluate total scene reconstruction errors using the validation camera on our synthetic dataset. Our setting (the last row) achieves the best performance. Segment. Loss: supervise the rendered masks (weights of each sampled ray) using the input segmentation [123, 168, 36]. Recon. Loss: color and depth reconstruction loss. Surface Reg. and Freespace Loss: the surface regularizer and the loss described in Section 6.3.

<table>
<thead>
<tr>
<th>Ablation Settings</th>
<th>Scene Reconstruction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Color (PSNR↑/ SSIM↑)</td>
</tr>
<tr>
<td>✓</td>
<td>18.59 / 0.81</td>
</tr>
<tr>
<td>✓</td>
<td>18.87 / 0.82</td>
</tr>
<tr>
<td>✓</td>
<td>20.08 / 0.79</td>
</tr>
<tr>
<td>✓</td>
<td>14.47 / 0.74</td>
</tr>
<tr>
<td>✓</td>
<td>17.85 / 0.76</td>
</tr>
<tr>
<td>✓</td>
<td>22.78 / 0.84</td>
</tr>
</tbody>
</table>

compared to ours. They also fail to reconstruct the deforming human models. In Figure 6.5 and Figure 6.6, we also present the extracted object motion trajectories in $\mathbb{R}^3$ as recovered by our initialization step (Section 6.3.1). Note that since we do not perform any loop closure, the trajectory estimates degrade over a longer distance due to error accumulation.

6.4.6 Ablation study

In Table 6.3, we conduct an ablation study using our synthetic dataset. While the commonly employed segmentation loss [123, 168, 36] can constrain the object shape through the rendered mask (weights of each sampled ray), it blocks the foreground reconstruction in joint optimization. The surface regularizers can stabilize the geometry models and improve both color and depth reconstruction. Our final setting (with surface regularizers and freespace loss) has the best full-scene reconstruction quality.

6.4.7 Applications

We support different editing modes: (i) novel view synthesis by changing the extracted camera trajectory; (ii) object-level manipulation by changing one or more object trajectories; (iii) deleting objects by removing them from the factored representations. Note that the scene-specific learned renders are held fixed during any of
Figure 6.8: Reconstruction and applications on our synthetic dataset. Here we show frames for the output RGB, depth, and underlying recovered geometries (extracted by performing diffuse shading using the estimated implicit representations $\psi_i$). We also show the recovered trajectories, along with corresponding ground truth trajectories. Recall that the 0-th object is the background, and $\{T_0(t)\}$ represents the camera path. Any stationary object gets reconstructed in the background layer in our factorization.

the edits. Figure 6.8 and Figure 6.9 show different examples. These edit modes are applied separately or in parallel, and test the quality of the scene understanding (i.e., factorization) by revealing unseen object parts and configurations.
Figure 6.9: Reconstruction and applications on the BEHAVE dataset. Here we also show frames for the output RGB, depth, underlying geometry rendering, and the recovered trajectories. Real-world data is harder than synthetic one due to the missing depth and sensor noises.
6.4.8 Memory and Implementation Details

We report the model size of our method and comparisons. iMAP uses 0.9MB (FG/BG); NiceSLAM uses 76MB (FG) and 135MB (BG) with $32^3$ and $64^3$ grid resolutions for the foreground and $32^3$ and $80^3$ for the background. In contrast, our model takes 5.7MB. We train all methods using our training framework on a single Nvidia RTX 3090 GPU. We do not use input depth to guide ray sampling for any of the methods as we observed that this reduces models’ generalization ability on novel view rendering. Instead, at each training iteration, we perform inverse transform sampling and sample 256 rays with 128 points per ray.

Architecture. We provide our network architecture in Figure 6.10. For the SDF MLP networks, we use geometric initialization [176], weighted normalization [177], Softplus activations, and a skip-connection at the 4-th layer. The input coordinates and view directions are lifted to a high dimensional space using positional encoding [90]. For rigid objects, we use $\mathbb{SE}(3)$ representation, i.e., a quaternion and a translation vector. For non-rigid objects, we use bijective deformation blocks [36] with weighted normalization and Softplus activations as well. For the color MLP network, we use ReLU activation.

6.5 Discussion and Limitations

We have presented a factored neural representation along with a joint optimization formulation that allows the separation of a monocular RGB-D video into object-level encodings, without requiring access to additional shape or motion priors. We demonstrated how to directly obtain object-level coupled geometry and appearance encoding, along with object trajectories and deformations. The factorized representation directly supports novel view synthesis along with authoring edits on object trajectories. Our work has limitations that we would like to address in future works, as discussed next.

- **Joint camera and object tracking.** We do not optimize the camera obtained during the initialization phase. As a next step, it would be interesting to jointly fine-tune the initial estimates, possibly by loop closing and locally
linearizing the transformation estimates to simplify the resultant optimization, while computing the factorization.

- **Inter object interactions.** We do not model object-object or object-background effects. For example, we do not explicitly model shadows or object interactions arising from human affordance considerations. In the future, it would be a possibility to model these in the volume rendering step, possibly by allowing rays to look up features before and after current pixels (e.g., using a transformer architecture).
• **Time-dependent appearance.** While our model accepts the time-dependent latent input (time codes), our employed datasets (both synthetic and real) only contain static lighting. Therefore, it will be interesting to apply our method to the scenes containing dynamic lighting and explore time-dependent effects.

• **Better architecture.** At present, we modeled object functions of the form $f_\theta$ simply using MLPs. More recent alternatives and localized versions [179, 160] or sparse points [115] can be explored alternatively. However, the challenge would then be effectively integrating information across multiple frames to model deformations, possibly by dynamically re-indexing the local grid-based representations or using an attention module.

• **Other priors.** We do not employ geometry or tracking priors in volume rendering. Several interesting choices can be considered, including depth and normal priors [180] for resolving sensor noises, optical flow [86, 128] for regularizing occlusion and poses, and layout estimation models [181, 182] for background modeling.
Chapter 7

Conclusions

7.1 Summary

In this thesis, we investigated the problem of 4D reconstruction of scenes with moving objects, as recorded from a moving camera, and address several sub-problems in this field. Here we summarize our contributions in each chapter.

In Chapter 3, we developed a data generation toolkit and established an evaluation protocol for benchmarking the 4D reconstruction task and resolving the lack of training data issue. We leveraged our generated data as a training set and a test set in Chapter 4 and Chapter 5 to solving object tracking and reconstruction problem.

In Chapter 4, we explored the idea of incorporating intra-category priors for tracking objects and reported performance improvement against the state-of-the-art methods. Our key hypothesis is that completion and dense correspondence priors can help to resolve heavy occlusion and provides robust object tracking. We presented an end-to-end pipeline that jointly solves detection, completion, and correspondence mapping for object pose estimation and tracking. We verify our hypothesis by training our network on our DYNSYNTH dataset and ScanNet [12] using Scan2CAD [42] annotations. Both qualitative and quantitative evaluations demonstrate performance improvement than comparison methods.

In Chapter 5, we proposed RigidFusion to simultaneously solve tracking and segmentation. Our key observation is that free-space information is more reliable in detecting moving objects under large camera or object motions than the commonly
used instance segmentation priors [37, 81, 39] and motion residuals [14]. We quantitatively evaluate our method against comparisons on our DYN SYNTH dataset and reported systematic improvement in terms of tracking accuracy and reconstruction quality. We also provide qualitative evaluations using real-world scans, and the results again verified our system outputs more accurate 4D reconstruction in high-dynamic environments. Our approach, being non-learning based, is not restricted to objects with semantic labels. Hence, it can be used to collect training data for typical object movements under real-world interactions.

Finally, in Chapter 6, we addressed the question of full dynamic scene reconstruction and proposed FactoredNeRF to jointly optimize both rigid and non-rigid scene elements using volume rendering. Our hypothesis is that neural volume rendering can be used to achieve full dynamic scene reconstruction and provide a factorized scene representation using monocular RGB-D video input. We verify our approach on a new synthetic dataset created by adding non-rigid motion from DeformingThings4D [174] dataset and real-world scans as well. Our representation can be trained on a single commodity GPU and acquire non-rigid objects, which verifies our hypothesis. We provide qualitative evaluations against other neural scene representations demonstrating better generalization ability and finer object reconstruction. Further, our scene representation supports scene editing operations, such as object re-positioning and deleting objects, without requiring re-training, which serves as the building block for the downstream applications.

7.2 Future Work

Understanding dynamic environments involving several vision and graphics challenges. In this section, we elaborate on some future avenues of dynamic scene reconstruction.

- Generative modeling. Synthetic data have become a rapidly developed field in recent years because of its usages on learning data priors and benchmarking performance, as we demonstrated in Chapters 4 to 6. A useful addition in this direction is to synthesize human-object interaction and generate a large-scale
dataset. There are several challenges reminding: the variance and the randomness of the synthesized data, the scale of the dataset, the synthetic-to-realistic gap in terms of object motion, environment lighting, sensor noises, and shading. A potential solution is to employ generative modeling techniques. Particularly, the recent success on language-driven generative modeling, which has received enormous success in various modalities, such as image synthesis [183, 184] meshes [185, 186] and human motions [187, 188]. Adapting a language model to synthesize human-object or object-to-object interactions with motion styles, e.g., slow steady and fast sweeping, will be worth exploring.

- **Static priors.** Another less explored topic is applying static data priors on dynamic tasks. Static priors are relatively easier to access than dynamic priors because of the availability of large-scale datasets [12, 40, 41]. There are still a lot of unknowns regarding utilizing static pre-training on dynamic tasks. Researchers have employed a similar idea on learning a weight initialization for achieving fast training on dynamic scene modeling [189]. Other potential usages are partial shape completion and correspondence searching. One example is the self-supervised vision transformer [190, 191], which demonstrates its ability on visual object tracking and segmentation without accessing dynamic training data. This direction has the potential to break the chicken-and-eggs situation in dynamic scene understandings such that we don’t have a mature solution to capture a large-scale dynamic dataset and the lack of dynamic data priors to create a robust capture system using neural networks.

- **Continuous learning.** Current dynamic reconstruction systems primarily focus on short sequences, typically less than a minute. Extending current approaches ([14, 37, 38, 39, 178, 34, 33] and our FactoredNeRF) to handle a daily-long video bring several new challenges, including object re-identification, memory consumption, scene change detection, global optimization, and loop closing. Several very recent works have proposed solutions for these research problems, such as continuous learning [192, 193, 194] and
localization with neural fields [195, 196]. However, they are focused on static environments. Adapting the state-of-the-arts to dynamic environments will be worth exploring.

- **Object state management.** Managing the state of the detected objects is not a trivial problem due to the view change and self-occlusion. This is particularly difficult under a highly dynamic environment with monocular input because a moving object can sometimes become a static item during a recording or disappear from the camera view. Efficiently solving object detection, full scene reconstruction, and object re-identification will be an important step for dynamic scene understanding.

### 7.3 Remark

In this thesis, we have addressed the problem of dynamic scene understating from four perspectives, synthetic data, learning geometry priors, free-space as segmentation signals, and representing a dynamic scene. To evaluate the effectiveness of our systems, we tested them on the challenging sequences containing large camera and object motions, using both synthetic and real-world data. Further, we highlighted several future directions, and we expect the continuous explorations of these research questions can accelerate the development of home assistant robots, augmented reality, and other exciting artificial intelligence applications.
Appendix A

A Glossary of the Terminology

Table A.1: The terminology used in this thesis.

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canonical Model</td>
<td>The reference model represents the reconstructed geometry in the reference coordinate space.</td>
</tr>
<tr>
<td>Correspondences</td>
<td>The same points observed in different input frames.</td>
</tr>
<tr>
<td>Fusion</td>
<td>Merging input scans into a model in the reference coordinate space.</td>
</tr>
<tr>
<td>Geometric Features</td>
<td>Features driven from an object’s surface points or normals.</td>
</tr>
<tr>
<td>Instance Segmentation</td>
<td>Image pixels are grouped according to individual objects. Each pixel can be assigned to multiple object groups.</td>
</tr>
<tr>
<td>Non-Rigid Motion</td>
<td>A type of motion that does not preserve angle and distance during movement. It is usually approximated by multiple rigid transformations.</td>
</tr>
<tr>
<td>Object Priors</td>
<td>The prior knowledge of the dynamic foreground object, including object geometry, appearance, and segmentation.</td>
</tr>
<tr>
<td>Occlusion, Occluded Geometry</td>
<td>Only a part of the target object/scene is observed during scanning.</td>
</tr>
<tr>
<td>Rigid Motion, Rigid Transformation</td>
<td>A type of motion can be modeled by an angle and distance preserved 3D transformation, which consists of a 3D rotation and translation.</td>
</tr>
<tr>
<td>Semantic Segmentation</td>
<td>Image pixels are labeled using a set of pre-defined semantic classes. In the context of indoor understanding, the semantic classes contain the categories of common indoor objects.</td>
</tr>
<tr>
<td>Surfel</td>
<td>Surface element. A disk-like data structure. Each surfel consists of a point, a radius, and a normal vector.</td>
</tr>
<tr>
<td>Truncated Signed Distance Function (TSDF)</td>
<td>A volumetric grid records the distance from a voxel to the closest surface. The distance is truncated to focus on the close-surface regions.</td>
</tr>
<tr>
<td>Template Mesh</td>
<td>A reference geometry for object tracking and detection.</td>
</tr>
<tr>
<td>Tracking Drift</td>
<td>The tracking errors accumulated in the system result in a large misalignment.</td>
</tr>
</tbody>
</table>
Appendix B

RigidFusion’s Experiment Details

We list all system parameters used in our quantitative evaluation in Chapter 5, including RigidFusion, and the two major comparing methods, CoFusion [14] and MaskFusion [37].

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>∆</td>
<td>60</td>
<td>the size of delay window</td>
</tr>
<tr>
<td>min_size</td>
<td>4500</td>
<td>the minimum size of new object segments</td>
</tr>
<tr>
<td>segth</td>
<td>10</td>
<td>freespace count threshold</td>
</tr>
<tr>
<td>detth</td>
<td>1.00E-04</td>
<td>foreground de-activation threshold</td>
</tr>
<tr>
<td>chnum</td>
<td>0.5Δ</td>
<td>foreground de-activation check frame numbers</td>
</tr>
<tr>
<td>bvxsize</td>
<td>0.03</td>
<td>background TSDF voxel size</td>
</tr>
<tr>
<td>fvxsize</td>
<td>0.01</td>
<td>foreground TSDF voxel size</td>
</tr>
<tr>
<td>btrunc</td>
<td>10 • voxel size</td>
<td>background truncation</td>
</tr>
<tr>
<td>ftrunc</td>
<td>15 • voxel size</td>
<td>foreground truncation</td>
</tr>
</tbody>
</table>
Table B.2: CoFusion’s experimental setting

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>confO</td>
<td>0.01</td>
<td>initial surfel confidence threshold for objects</td>
</tr>
<tr>
<td>confG</td>
<td>1.0</td>
<td>initial surfel confidence threshold for scene</td>
</tr>
<tr>
<td>segMinNew</td>
<td>0.015</td>
<td>the minimum size of new object segments</td>
</tr>
<tr>
<td>segMaxNew</td>
<td>0.4</td>
<td>the maximum size of new object segments</td>
</tr>
<tr>
<td>thNew</td>
<td>5.5</td>
<td>the threshold of initializing a new model</td>
</tr>
<tr>
<td>offset</td>
<td>22</td>
<td>offset between creating models</td>
</tr>
<tr>
<td>or</td>
<td>1</td>
<td>outlier rejection level</td>
</tr>
<tr>
<td>crfRGB</td>
<td>10</td>
<td>the parameters for the conditional random field</td>
</tr>
<tr>
<td>crfDepth</td>
<td>0.9</td>
<td>the parameters for the conditional random field</td>
</tr>
<tr>
<td>crfPos</td>
<td>1.8</td>
<td>the parameters for the conditional random field</td>
</tr>
<tr>
<td>crfAppearance</td>
<td>15</td>
<td>the parameters for the conditional random field</td>
</tr>
<tr>
<td>crfSmooth</td>
<td>4</td>
<td>the parameters for the conditional random field</td>
</tr>
<tr>
<td>icpWeight</td>
<td>10</td>
<td>ICP weight</td>
</tr>
</tbody>
</table>

Table B.3: MaskFusion’s experimental setting

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Values</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>confO</td>
<td>0.01</td>
<td>initial surfel confidence threshold for objects</td>
</tr>
<tr>
<td>confG</td>
<td>1.0</td>
<td>initial surfel confidence threshold for scene</td>
</tr>
<tr>
<td>segMinNew</td>
<td>0.015</td>
<td>the minimum size of new object segments</td>
</tr>
<tr>
<td>segMaxNew</td>
<td>0.4</td>
<td>the maximum size of new object segments</td>
</tr>
<tr>
<td>thNew</td>
<td>5.5</td>
<td>the threshold of initializing a new model</td>
</tr>
<tr>
<td>offset</td>
<td>22</td>
<td>offset between creating models</td>
</tr>
<tr>
<td>or</td>
<td>1</td>
<td>outlier rejection level</td>
</tr>
<tr>
<td>filter_classes</td>
<td>-</td>
<td>filter instance segmentation by semantic labels</td>
</tr>
<tr>
<td>icpWeight</td>
<td>20</td>
<td>ICP weight</td>
</tr>
<tr>
<td>frameQ</td>
<td>30</td>
<td>the size of frame-queue</td>
</tr>
</tbody>
</table>
Bibliography


[44] Maik Keller, Damien Lefloch, Martin Lambers, Shahram Izadi, Tim Weyrich, and Andreas Kolb. Real-time 3d reconstruction in dynamic scenes using


Bibliography


[120] Shaohui Liu, Yinda Zhang, Songyou Peng, Boxin Shi, Marc Pollefeys, and Zhaopeng Cui. Dist: Rendering deep implicit signed distance function with


[127] Matt Deitke, Winson Han, Alvaro Herrasti, Aniruddha Kembhavi, Eric Kolve, Roozbeh Mottaghi, Jordi Salvador, Dustin Schwenk, Eli VanderBilt, Matthew


