A 3D photo-realistic environment simulator for mobile robots

ISBN: 978-94-6423-139-7

Published by Proefschriftmaken
http://www.Proefschriftmaken.nl

Cover design by Honglin Yuan

Copyright © 2021 Honglin Yuan.

All rights reserved.

A 3D photo-realistic environment simulator for mobile robots

Een 3D-fotorealistische omgevingssimulator voor mobiele robots
(met een samenvatting in het Nederlands)

Proefschrift

ter verkrijging van de graad van doctor aan de Universiteit Utrecht op gezag van de rector magnificus, prof. dr. H. R. B. M. Kummeling, ingevolge het besluit van het college voor promoties in het openbaar te verdedigen op maandag 15 februari 2021 des ochtends te 10.30 uur

door

Honglin Yuan

geboren op May 13, 1990
te Shandong, China

Promotor:	Prof. dr. R.C. (Remco) Veltkamp
Promotor:	Prof. dr. Alexandru C. Telea

SUMMARY

Recent years have witnessed great advancement in visual artificial intelligence (AI) research based on deep learning. To take advantage of deep learning, we need to collect a large amount of data in various environments and conditions. However, collecting such data is a time-consuming and labor-intensive task. Apart from that, developing and testing visual AI algorithms for robots are expensive and in some cases dangerous processes in the real world. To address these challenges, in this thesis we investigate algorithms to design a high-quality simulator for mobile robots. We aim to narrow the gap between simulation and reality, generate infinitely many photo-realistic color-and-depth image pairs from arbitrary locations and allow transferring algorithms that are developed and tested in simulation to physical platforms without domain constraints.

To achieve our goals, we design a view synthesis module used for our simulator to synthesize free-viewpoint photo-realistic color-and-depth image pairs. Our approach combines depth refinement, adaptive view selection and layered 3D warping to lower the rendering complexity and improve the quality of synthesized images. We also design controller, recorder, and visualizer modules for our simulator. These modules are designed to work together, providing a variety of data including real-time camera poses, synthesized color-and-depth image pairs, trajectories of the robot for training robotic tasks.

Based on our simulator, we build a 3D dataset for benchmarking 6D object pose estimation which pays an important role in robotic grasping and manipulation research. The dataset consists of different objects that cover a variety of shapes, rigidity, sizes, weight and textures. For our simulator can seamlessly integrate robots with virtual scenes, we generate a large number of photo-realistic color-and-depth image pairs with ground truth 6D poses for training data-driven pose estimation approaches. Our dataset is freely distributed to research groups worldwide by the Shape Retrieval Challenge (SHREC) benchmark on 6D pose estimation.

We conduct a variety of experiments to investigate the performance of different pose estimation approaches proposed from our benchmark using different evaluation metrics. We learn important lessons from the current pose estimation algorithms. This gives insight into where researchers' attention should be paid to make progress on pose estimation. Apart from that, we propose a novel approach to further improve the performance of 6D object pose estimation by effectively computing hidden representations from color and depth images, and then fusing them properly with a graph attention network which fully exploits the relationship between visual and geometric features.

Overall, we propose a 3D photo-realistic virtual environment simulator to develop vision-based algorithms for AI research. Experiments demonstrate our simulator narrows the reality gap between the virtual environment and the real scene. Thus, computer vision-based algorithms including depth estimation, object recognition and 6D object pose estimation developed in simulation can be transferred to the real world without domain adaption.

SAMENVATTING

De laatste jaren is er grote vooruitgang geboekt in het onderzoek naar visuele kunstmatige intelligentie (AI), gebaseerd op diepgaand leren. Om gebruik te maken van deep learning moeten we een grote hoeveelheid data verzamelen in verschillende omgevingen en omstandigheden. Het verzamelen van dergelijke data is echter een tijdrovende en arbeidsintensieve taak. Daarnaast zijn het ontwikkelen en testen van visuele AI-algoritmen voor robots dure en in sommige gevallen gevaarlijke processen in de echte wereld. Om deze uitdagingen aan te gaan, onderzoeken we in dit proefschrift algoritmes om een hoogwaardige simulator voor mobiele robots te ontwerpen. We streven ernaar de kloof tussen simulatie en werkelijkheid te verkleinen, oneindig veel fotorealistische beeldparen in kleur en diepte te genereren vanaf willekeurige locaties en algoritmen die in de simulatie zijn ontwikkeld en getest over te brengen naar fysieke platformen zonder domeinbeperkingen.

Om onze doelen te bereiken, ontwerpen we een beeldsynthesemodule voor onze simulator om fotorealistische kleuren- en dieptebeeldparen met een vrij gezichtspunt te synthetiseren. Onze aanpak combineert diepteverfijning, adaptieve weergaveselectie en gelaagde 3D vervorming om de renderingcomplexiteit te verlagen en de kwaliteit van de gesynthetiseerde beelden te verbeteren. We ontwerpen ook controller-, recorder- en visualisatiemodules voor onze simulator. Deze modules zijn ontworpen om samen te werken en bieden een verscheidenheid aan gegevens, waaronder realtime camerahoudingen, gesynthetiseerde kleuren- en dieptebeeldparen, trajecten van de robot voor het trainen van robottaken.

Op basis van onze simulator bouwen we een 3D-dataset voor het benchmarken van 6D-object houdingen, wat een belangrijke rol speelt in het robotgrijpen en manipulatie-onderzoek. De dataset bestaat uit verschillende objecten die een verscheidenheid aan vormen, stijfheid, afmetingen, gewicht en texturen omvatten. Omdat onze simulator robots naadloos kan integreren met virtuele scènes, genereren we een groot aantal fotorealistische kleuren- en dieptebeeldparen met ground truth 6D poses voor het trainen van data-gedreven houdingschatting benaderingen. Onze dataset wordt vrijelijk verspreid onder onderzoeksgroepen wereldwijd door de Shape Retrieval Challenge (SHREC) benchmark op 6D-poseschatting.

We voeren een verscheidenheid aan experimenten uit om de prestaties van verschillende houdingschattingsmethoden die vanuit onze benchmark worden voorgesteld, te onderzoeken aan de hand van verschillende evaluatiemetingen. We leren belangrijke lessen uit de huidige houdingschattingsalgoritmen. Dit geeft inzicht in waar de aandacht van de onderzoekers naartoe moet gaan om vooruitgang te boeken op het gebied van houdingschatting. Daarnaast stellen we een nieuwe aanpak voor om de prestaties van 6D object houdingschatting verder te verbeteren door het effectief berekenen van verborgen weergaven van kleur- en dieptebeelden, en deze vervolgens goed te versmelten met een grafisch attentienetwerk dat de relatie tussen visuele en geometrische kenmerken volledig benut.

In het algemeen stellen we een 3D fotorealistische virtuele omgevingssimulator voor om op visie gebaseerde algoritmen voor AI-onderzoek te ontwikkelen. Experimenten tonen aan dat onze simulator de realiteitskloof tussen de virtuele omgeving en de echte scène verkleint. Zo kunnen computervisie-gebaseerde algoritmen, waaronder dieptesschatting, objectherkenning en 6D-oorsprongschatting, ontwikkeld in de simulatie, naar de echte wereld worden overgebracht zonder domeinaanpassing.

ACKNOWLEDGEMENTS

I would like to thank my supervisor Prof. Remco C. Veltkamp for inspiring and supporting me throughout my PhD study. It was Remco who welcomed me with open arms when I applied for a PhD position in UU, telling me he would like to supervise me. It was the first day of 2017, a special date that I will always remember. During my PhD, I had many ups and downs, just like every PhD student experienced. Rejected papers, failed experiments, unsolved problems made me disappointed. His incredible kindness gives me courage and inspiration to go through these dark hours. Remco trusts his students to be self-motivated and to find their own path. I really appreciate this large amount of freedom and I believe it will benefit me for a lifetime. Thanks also to Prof. Alexandru C. Telea for agreeing to be my second supervisor, reviewing my thesis and giving valuable feedback.

Thanks to my colleagues Tao Ku, Ju Zhang, Ilja Gubins. Tao as my longest roommate helps me solve many programming problems with his incredible patience. Many bugs have been spotted and fixed with his help. I also receive many comforts from deep conversations about life, study or simply mindless talking. Ju is a successful researcher, being a role model: one who maintains a good work-life balance. Her invaluable help on revising my papers gives me a lot of inspiration on how to write a research paper. I am grateful to Ilja. Even though we are in different projects, his creative ideas and incredible kindness help me a lot.

I am incredibly thankful to my friends Na Li, Chunlei Liu, Xuexia Zhang, Fuqing Li, Nanjing Chen, Cui Li, Qingwu Liu. I am touched by their firm support and warm talks. I am also grateful to all my other colleagues and friends.

Many thanks to China Scholarship Council for providing me funding support for my PhD study.

Finally, I owe my doctoral work to my parents, relatives, and Rongfeng Dong. Without their selfless love and support, I would never have been able to finish my thesis.

CONTENTS

1 INTRODUCTION 1

1.1 Motivation and goals 2
1.2 Context 3
1.3 Contributions 5
1.4 Structure 8
1.5 Publications 9

2 BACKGROUND 11

2.1 Image based rendering 11
2.2 3D reconstruction 20
2.3 Robot Operating System 33
2.4 Deep learning 37

3 FREE-VIEWPOINT IMAGE BASED RENDERING 45

3.1 Introduction 46
3.2 Related work 47
3.3 Overview 49
3.4 Free-viewpoint image based rendering 50
3.5 Experimental results 57
3.6 Conclusion and future work 67

4 PRESIM: A 3D PHOTO-REALISTIC ENVIRONMENT SIMULATOR 69

4.1 Introduction 70
4.2 Related work 71
4.3 Photo-realistic virtual environment 73
4.4 Tasks 82
4.5 Experimental results 84
4.6 Conclusion and future work 91

5 SIM-TO-REAL 6D OBJECT POSE ESTIMATION DATASET CONSTRUCTION 93

5.1 Introduction 94

5.2 Related work 96
5.3 The RobotP dataset 97
5.4 Experimental results 112
5.5 6D object pose estimation challenge 117
5.6 Conclusion and future work 118

6 6D OBJECT POSE ESTIMATION 119
6.1 Introduction 120
6.2 Related work 122
6.3 Analysis of benchmarking approaches for 6D object pose estimation . . 123
6.4 Object pose estimation with color/geometry attention fusion 133
6.5 Conclusion and future work 143

7 CONCLUSIONS AND FUTURE WORK 145
7.1 Reaching our goals 145
7.2 Beyond our goals 147
7.3 Concluding remarks 149

CURRICULUM VITAE 151

1

INTRODUCTION

FIGURE 1.1: *Over time, we have strived to equip robots with thoughts and movement. The term "robot", is first used in the science fiction play named R.U.R. (left) and the first physical robot, ELEKTRO, is displayed at the World's fair (middle). In this thesis, we aim to develop simulation environment and artificial algorithms for humanoid robots (right).*

Imagine asking a robot to pick up a banana from a table and bring it back to you. To accomplish such a task, the robot would need a range of skills, such as language understanding, object recognition, visual navigation and object pose estimation. This goal is something we have been working towards for centuries. Already in ancient mythologies, artificial people such as the talking mechanical handmaidens built by the Greek god Hephaestus (Vulcan to the Romans) out of gold [Ger03], and clay golems of Jewish legend had appeared. In a 1920 science fiction play, written by the Czech writer Karel Čapek, the term ”robot”, was first used, where R.U.R. (Rossum’s Universal Robots) was a robot which could think and behave like humans [Rob16] (Figure 1.1(a)). By 1939, the first physical robot, ELEKTRO (Figure 1.1(b)), went on display at the World’s fair. It could walk by voice command, speak about 700 words, smoke cigarettes, blow up balloons, and move its head and arms [RS10]. In 1948 the first autonomous robot was created by William Grey Walter [FIS14]. The robot could find its way around obstacles by making use of a bump sensor. Later, with the importance of simulation being recognized, in 1997 IBM created Deep Blue, a computer which defeated world champion Garry Kasparov [CHJH02]. Since then, simulation, specifically 3D simulation has started to attract much attention. With the remarkable breakthroughs in simulation, vision and language communities, more and more robots (e.g., humanoid Pepper ((Figure 1.1 (c)) manufactured by SoftBank Robotics.) are created to achieve human-like abilities. Various robots have been widely used in manufacturing, transport, space exploration, surgery, industrial goods and laboratory research [KK17]. While the advent of deep learning has led to significant progress in computer vision and natural language processing [Gir15, DCLT18], today’s robots still lack the ability to achieve the aforementioned goals.

1.1 MOTIVATION AND GOALS

Training robots in the real world presents many challenges: (1) preparing training environments is a time-consuming process, and battery drainage or hardware failures can delay the training process; (2) testing in the real world cannot run any faster than real time and cannot to be parallelized; (3) training robots for tasks like bomb-detection, search-and-rescue and underwater exploration is resource intensive and in some cases dangerous. Furthermore, poorly-trained robots may injure themselves or others; (4) replicating or controlling experiment conditions like temperature, pressure, and air flow are difficult. Gradually, the simulation environment is becoming an effective way to solve these problems, for developing and testing algorithms in simulation environments is fast, flexible, safe and reproducible.

A major current focus of the simulation environment is to reproduce high-quality free-viewpoint rendering of real senses. There are a number of open source simulators (e.g., Gazebo [KH04] and Chalet [YMB$^+$18]) to achieve this goal by parameter settings of scene details, including geometry, textures, lighting and 3D modeling of static objects. However, parameter setting is time-consuming and labor-intensive. Even with precise modeling and suitable parameter settings, the simulated world still lacks richness and diversity of the real world. This disadvantage may result in the failure of transferring algorithms that are developed and tested in simulation to physical platforms for many vision-based tasks, such as object recognition, obstacle avoidance, and visual navigation.

In this thesis, we design and implement a photo-realistic environment simulator for robots and strive toward the following goals:

1. provide free-viewpoint photo-realistic rendering of real scenes, using a collection of RGB-D images,

2. allow developing robotics applications and seamlessly interfacing with Robot Operating System (ROS),

3. easily control the movement of robots, and provide real-time positions and whole trajectories of the moving robot, and a global 3D map,

4. generate representative datasets with rich data for training and evaluating algorithms designed for robotic vision tasks, and

5. enable robots to learn artificial intelligence algorithms (e.g., object recognition and pose estimation) in simulation and allow transforming knowledge learned from simulation to the real world without domain adaptation.

In order to accomplish these objectives, we look for cross-disciplinary approaches that coverage robotics, computer vision and deep learning.

1.2 CONTEXT

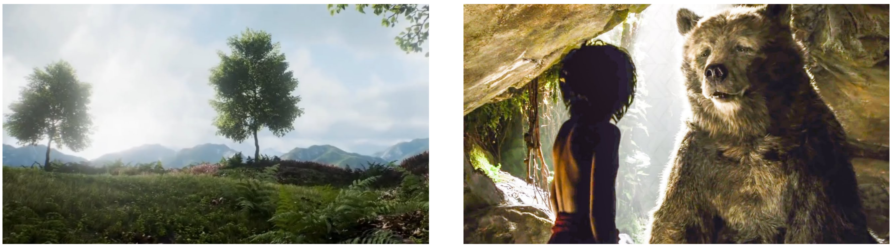

(a) *Video game based on Unreal Engine.* (b) *”The Jungle Book” film.*

FIGURE 1.2: *Snapshots of rendering in the video game and film.*

In this thesis, we make the best use of advances in the fields of view synthesis, 3D reconstruction, robotics frameworks and deep learning.

Rendering or image synthesis refers to the automatic process of generating 2D images from a 3D model or a set of 2D images of scenes. It has been used widely in video games, film industry, simulators and free-viewpoint TV [KDW$^+$17, BE01]. Nowadays, with exhaustive detailed modeling of 3D models, rendering algorithms are very close to achieve one of the most important goals of computer graphics: realism (see Figure 1.2).

However, achieving realism requires a large amount of laborious work for generating 3D geometry models that contain lights, viewpoints, material properties (e.g., textures and normal maps) to describe a scene. One promising approach that avoids time-consuming content creation and expensive lighting simulation is image based rendering (IBR) [SCK08]. It gives users real-time free-viewpoint rendering of the real scene, using a sparse collection of captured images. In the rendering process, instead of using 3D geometry, 2D images are directly used to synthesize virtual images.

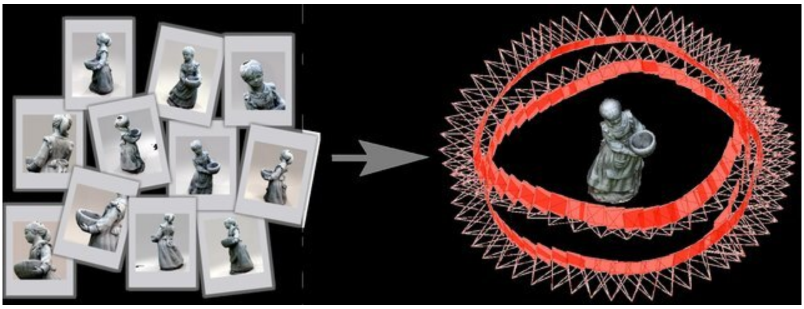


FIGURE 1.3: *A 3D model generated by 3D reconstruction [BCM18].*

3D reconstruction is the creation of 3D models or appearance of real objects from a set of images (see Figure 1.3). Semantically rich and geometrically accurate 3D reconstruction can provide rich information about the object including 3D shapes, textures, and 3D coordinates of points on the object. It has been widely used in various vision-based applications, such as virtual reality, visual navigation, computer animation and building maintenance [IZU13, CT19]. For example, with 3D reconstructed scenes, people can stay at home to take virtual tours for real estate, which allows freely walking around the room and viewing the scene from vastly different perspectives. With the development of sensors (e.g., 3D cameras and laser scanners), high-quality 3D reconstruction results already exist for indoor and out door environments.

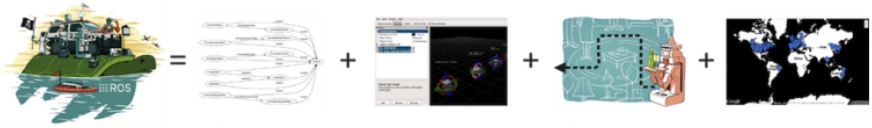
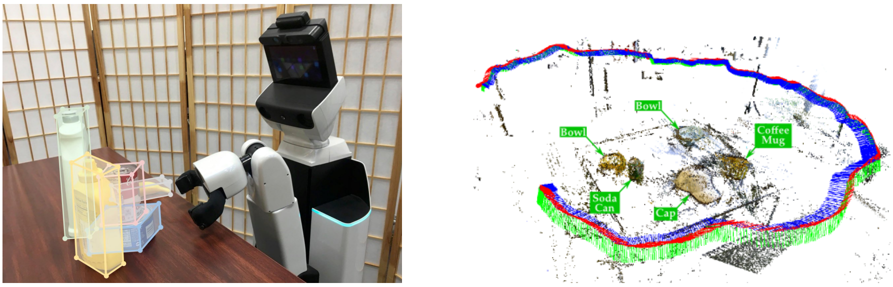


FIGURE 1.4: *An image showing the ”ROS equation”: ROS = Plumbing + Tools + Capabilities + Ecosystem.*

The Robot Operating System (ROS) [QCG^{+}09], a flexible framework for writing robot software has simplified the implementation of tasks such as hardware abstraction, visual perception, and behavior generation. The primary goal of ROS is sharing and collaboration, which supports code reuse in robotics research and development. It now consists of tens of thousands of users worldwide, working in domains ranging from tabletop hobby projects to large industrial automation systems. It provides libraries and tools for software developers to create their robotics applications using an existing foundation rather than doing everything themselves. The philosophy behind ROS is described in Figure 1.4.



(a) *6D object pose estimation [WXZ+19]* (b) *Object recognition [PL15]*

FIGURE 1.5: *Robotics applications.*

In recent years, deep learning has revolutionized the computer vision, natural language processing, machine translation and speech recognition fields. Inspired by information processing and distributed communication nodes in the human brain, deep learning models are based on artificial neural networks that consist of multiple hierarchical layers to progressively extract higher level features from the raw input. Deep learning approaches have achieved state-of-the-art performance on various computer vision benchmarks, such as face recognition benchmark [HMBLM08], object tracking benchmark [RDS+15] and ImageNet [RDS+15]. With a large quantity of data, they have also been used successfully in robotics applications, such as object recognition [PL15], visual scene understanding [SZZJ17] and 6D object pose estimation [WXZ+19] (see Figure 1.5).

1.3 CONTRIBUTIONS

In this thesis, we focus on developing a 3D photo-realistic simulator for designing artificial intelligence (AI) algorithms for robots. Our main contribution are as follows:

Chapter 3 focuses on free-viewpoint view synthesis with a sparse set of RGB-D images. Our main contributions are summarized as follows:

- A novel depth refinement algorithm that respects photo-consistency and edge preservation to correct misalignment between color-and-depth image pairs and fill missing depth information.

- A novel adaptive view selection approach that effectively avoids selecting redundant and useless input views to improve the quality of synthesized images and the rendering speed.

• A novel rendering algorithm providing high-quality free-viewpoint synthesized images, which is based on layered 3D warping to handle occluded elements and lower the rendering complexity.

The content of this chapter is based on the following paper:

- Honglin Yuan, Remco C. Veltkamp. Free-viewpoint image based rendering with layered depth maps. *Submitted for publication.*

Chapter 4 presents PreSim, a 3D environment simulator for training and testing vision-based algorithms. Our main contributions are summarized as follows:

• A photo-realistic 3D virtual environment that provides users with ground truth poses of the multisensory model and free-viewpoint color-and-depth image pairs, even in regions where a global 3D reconstruction of the scene has inaccurate or missing data.

• A global visualizer providing real-time positions and whole trajectories of moving robots, and a global 3D map.

• A sequence controller and recorder components to control the movement of robots and store all the required information for developing AI algorithms.

• A novel view synthesis module providing free-viewpoint rendering that combines depth refinement, adaptive view selection and layered 3D warping to lower the rendering complexity and improve the quality of synthesized images.

The content of this chapter is based on the following paper:

- Honglin Yuan, Remco C. Veltkamp. PreSim: A 3D photo-realistic environment simulator for visual AI. *Submitted for publication.*

Chapter 5 presents the RobotP dataset designed for benchmarking in 6D object pose estimation which plays an important role in robotic grasping and manipulation research. The awareness of the 3D rotation and 3D translation matrices of objects in a scene is referred to as 6D, where the D stands for the degree of freedom of the pose. Our main contributions are summarized as follows:

• A representative dataset providing high-quality RGB-D images, ground truth poses, object segmentation masks, 2D bounding boxes and accurate 3D models for daily used objects with different sizes, shapes, and textures, which covers a wide range of pose estimation challenges.

• A novel pose refinement approach that uses a local-to-global optimization strategy to achieve the improved accuracy of each pose and global pose alignment.

• A novel depth generation algorithm producing high-quality depth images, which is able to accurately align the depth image to its corresponding color image and fill missing depth information.

• Careful merging of multi-modal sensor data for object reconstruction, followed by an algorithm to produce the segmentation mask and 2D bounding box for each object automatically.

• A training dataset generated by a free-viewpoint image based rendering approach in a simulated environment. It provides a large amount of high-resolution and photo-realistic color-and-depth image pairs which have plausible physical locations, lighting conditions, and scales.

• The Shape Retrieval Challenge benchmark on 6D object pose estimation. The benchmark allows evaluating and comparing pose estimation algorithms under the same standard. Evaluation results indicate that there is considerable room for improvement in 6D object pose estimation, particularly for objects which have dark colors or reflective characteristics, and photo-realistic images are helpful to increase the accuracy of pose estimation algorithms.

The content of this chapter is based on the following paper:

- Honglin Yuan, Remco C. Veltkamp, Georgios Albanis, Nikolaos Zioulis, Dimitrios Zarpalas and Petros Daras. SHREC 2020 track: 6D object pose estimation. *Eurographics Workshop on 3D Object Retrieval.* The Eurographics Association, 2020.
- Honglin Yuan, Remco C. Veltkamp. RobotP: A Benchmark Dataset for 6D Object Pose Estimation. *Submitted for publication.*

In **Chapter 6** we conduct a variety of experiments to investigate the performance of different pose estimation approaches proposed from the Shape Retrieval Challenge benchmark on 6D pose estimation in Chapter 5. Apart from that, we propose a novel approach to predict the 6D pose of a given object. Our main contributions are summarized as follows:

• A comprehensive evaluation of 6D object pose estimation approaches. We use different evaluation metrics to compare the proposed methods on our benchmark. Evaluation results indicate that approaches that fully exploit the color and geometric features are more robust for 6D pose estimation of reflective objects and occlusion. Besides, methods that estimate the 6D pose in a single and consecutive network are more robust to texture-less objects and run faster.

• An efficient feature extraction method extracting representative local and global geometric features from point clouds, which makes it robust to handle heavy occlusion, low texture and sensor noise for 6D object pose estimation.

- A new multi-feature fusion network that improves 6D pose prediction performance by applying a graph attention network to fully exploit the relationship between visual and geometric features and compute hidden feature representations between these features.

The content of this chapter is based on the following paper:

- Honglin Yuan, Remco C. Veltkamp. 6D Object Pose Estimation With Color/Geometry Attention Fusion. *2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV), pp. 529-535, 2020.*

1.4 STRUCTURE

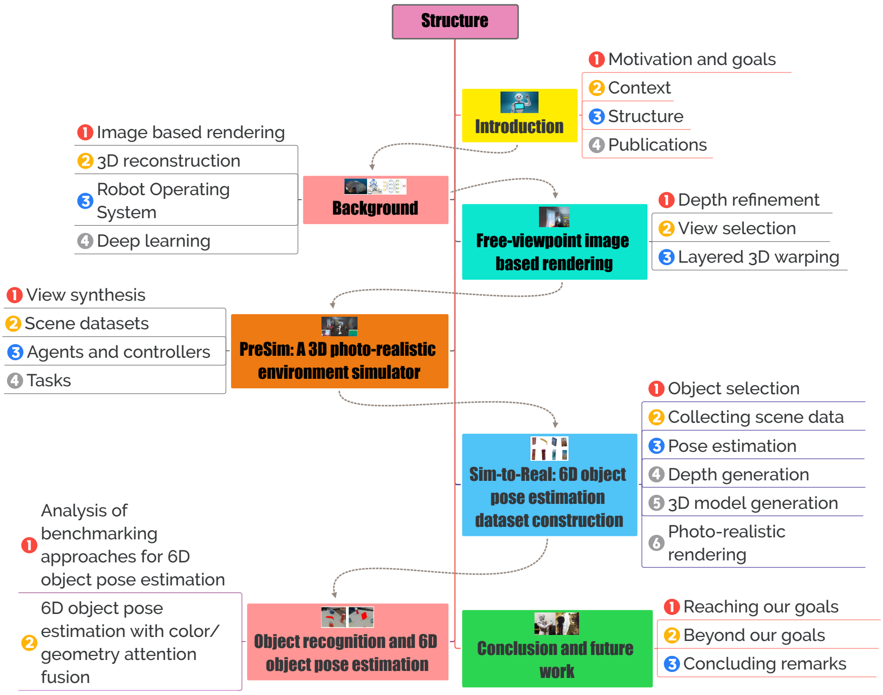


FIGURE 1.6: *PhD thesis structure.*

Figure 1.6 shows the structure of this thesis. The first chapter states the motivation, goals and contributions for our work. In Chapter 2, we give a brief overview of basic techniques and ideas on image-based rendering, 3D reconstruction, ROS, and deep learning that are relevant to the methods presented in this thesis.

In Chapter 3, we develop a free-viewpoint image based rendering approach to produce photo-realistic synthesized images. Based on the view synthesis approach proposed in Chapter 3, in Chapter 4 we design a 3D photo-realistic environment simulator to develop and test visual AI algorithms for mobile robots. In Chapter 5, we build a 3D dataset designed for the 6D object pose estimation challenge organized by us. Taking the advantage of our simulator described in Chapter 4, we produce a large number of color-and-depth image pairs with ground truth 6D poses for our dataset. In Chapter 6, we conduct a variety of experiments based on our benchmark organized in Chapter 5 to investigate how different state-of-the-art pose estimation approaches perform in terms of various object properties. Apart from that, we propose a novel approach to handle heavy occlusion, low texture and sensor noise for 6D object pose estimation.

In Chapter 7, we conclude the thesis and present the contributions, limitations and suggest some possible research directions for future work.

1.5 PUBLICATIONS

This thesis is based on the following publications:

1. Chapter 3: Honglin Yuan, Remco C. Veltkamp. Free-viewpoint image based rendering with layered depth maps. *Submitted for publication.*

2. Chapter 4: Honglin Yuan, Remco C. Veltkamp. PreSim: A 3D photo-realistic environment simulator for visual AI. *Submitted for publication.*

3. Chapter 5: Honglin Yuan, Remco C. Veltkamp, Georgios Albanis, Nikolaos Zioulis, Dimitrios Zarpalas and Petros Daras. SHREC 2020 track: 6D object pose estimation. *Eurographics Workshop on 3D Object Retrieval.* The Eurographics Association, 2020.

4. Honglin Yuan, Remco C. Veltkamp. RobotP: A Benchmark Dataset for 6D Object Pose Estimation. *Submitted for publication.*

5. Chapter 6: Honglin Yuan, Remco C. Veltkamp. 6D Object Pose Estimation With Color/Geometry Attention Fusion. *2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV), pp. 529-535, 2020.*

2

BACKGROUND

This chapter gives a brief overview of image based rendering, 3D reconstruction, ROS and deep learning techniques and ideas. The purpose of this chapter is to provide background knowledge required to understand the rest of the thesis, for readers who are not familiar with computer vision, robotics and deep learning. For a more extensive overview, we refer to [SCK08, HZ03, QCG$^+$09, GBC16].

2.1 IMAGE BASED RENDERING

Image based rendering (IBR) technology generates photo-realistic images which are comparable to those produced by conventional computer graphics methods, directly from 2D images without a full 3D geometric representation. It can be classified according to the accuracy and amount of geometric information required, as shown in Figure 2.1. The spectrum of IBR methods varies from algorithms that do not require any geometry, to use image correspondences (implicit geometry) to require detailed explicit geometry [SK00].

Another way to classify IBR approaches is based on capture setup and range of motion. Capture plays an important role on IBR. It determines the freedom of IBR approaches and has a great impact on the quality of synthesized images. An exhaustive capture process is helpful to avoid visual artifacts and increase the navigation freedom.

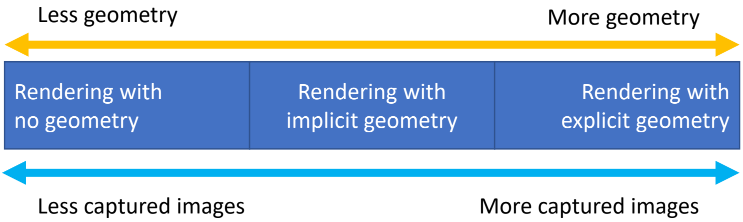

Less geometry
More geometry
Rendering with no geometry
Rendering with implicit geometry
Rendering with explicit geometry
Less captured images
More captured images

FIGURE 2.1: *Spectrum of IBR algorithms. Approaches on the left extreme of this spectrum use no geometry information and a small number of images as input, while on the right explicit geometry information and exhaustively capturing a dense set of images in a scene are required.*

2.1.1 PINHOLE CAMERA MODEL

We use *pinhole camera* model illustrated in Figure 2.2 to explain the process of capturing an image. The pinhole camera model describes the mathematical relationship between the coordinates of a point in 3D space and its projection onto an image plane. The pinhole camera model has no lenses to focus light, so that it is able to avoid geometric distortions and blurring of unfocused objects caused by lenses.

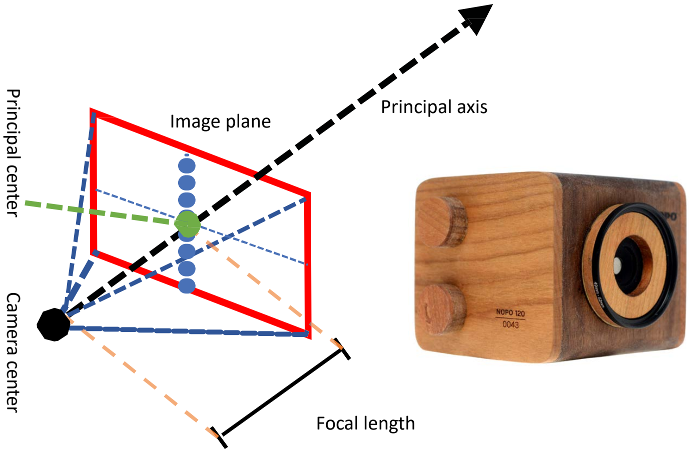
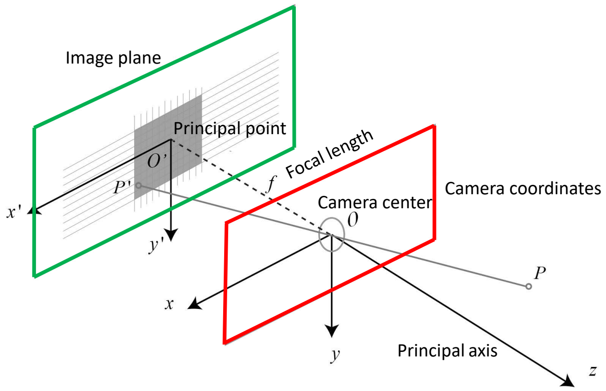


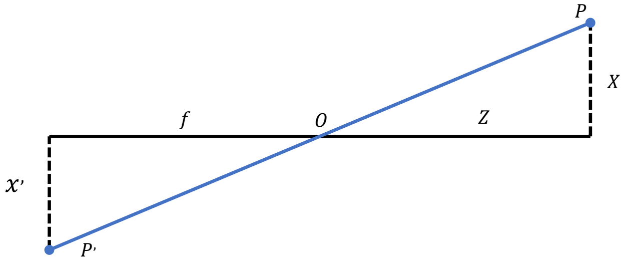
FIGURE 2.2: *The pinhole camera model.*

2.1.1.1 PINHOLE CAMERA GEOMETRY

The geometry of mapping a point in 3D space to an image plane is illustrated in Figure 2.3. Let the center O of the projection be the origin of a 3D Euclidean coordinate system, which is called the *camera center*. The three axes of the coordinate system are referred to as x, y, z. Axis z is pointing in the viewing direction of the camera and is referred to as the *optical axis*. The plane, $z = f$, which is parallel to axes x and y is called the *image plane*, where f is the focal length of the pinhole camera. Suppose that a 2D coordinate system is defined in the image plane, allowing each point on it to be identified by an image coordinate. The two axes of the coordinate system are referred to as x' and y'. The point where the principal axis meets the image plane is called the *principal point*. Under the pinhole camera model, a 3D point, $P = [X, Y, Z]^{\mathrm{T}}$, in the 3D Euclidean space is projected to the point $P' = [x', y']^{\mathrm{T}}$ on the image plane with a ray that originates from the Euclidean space origin, O.



(a) *The geometry of a pinhole camera as seen in 3D dimensions.*



(b) *The geometry of a pinhole camera as seen from the y axis.*

FIGURE 2.3: *The geometry of mapping a point in 3D space to an image plane. A 3D point P is mapped to P' in the image plane by camera intrinsic matrix.*

By similar triangles, the mapping function from 3D Euclidean space to the image plane is able to be derived. In Figure 2.3(b) which is generated by looking down in the negative direction of the y-axis, there are two right triangles, having congruent angles. Based on the similarity theorem, the two triangles are similar. The catheti of the left triangle are x' and f, and the catheti of the right triangle are X and Z. Since the two triangles are similar it follows that

$$\frac{Z}{f} = -\frac{X}{x'} \; or \; x' = -f\frac{X}{Z}. \tag{2.1}$$

A similar investigation, looking in the negative direction of the x-axis gives

$$\frac{Z}{f} = -\frac{Y}{y'} \; or \; y' = -f\frac{Y}{Z}. \tag{2.2}$$

This can be summarized as

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = -\frac{f}{Z} \begin{bmatrix} X \\ Y \end{bmatrix}. \tag{2.3}$$

The negative sign means the obtained image is an inverted image, while the image captured by a real pinhole camera is rotated by 180°. In order to produce a non-rotating image, which is the expected image from a camera, the image plane is placed in front of the camera, intersecting the Z axis at f instead of at $-f$, as shown in Figure 2.2. This would generate a virtual (or front) image plane which removes the negative sign in the formula. Therefore,

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \frac{f}{Z} \begin{bmatrix} X \\ Y \end{bmatrix}. \tag{2.4}$$

The mapping process can also be expressed as a linear mapping in homogeneous coordinates. Let $P = [X, Y, Z, 1]^{\mathrm{T}}$ be a point in homogeneous coordinates (a 4-dimensional vector) and $Q = [uz, uz, z]^{\mathrm{T}}$ be the image point represented by a homogeneous 3D vector projected by P. Then 5.4 can be written in terms of matrix multiplication as

$$\begin{bmatrix} uz \\ vz \\ z \end{bmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \tag{2.5}$$

Furthermore, 5.4 can be written compactly as

$$Q = HP, \tag{2.6}$$

where H is the mapping matrix for the pinhole model.

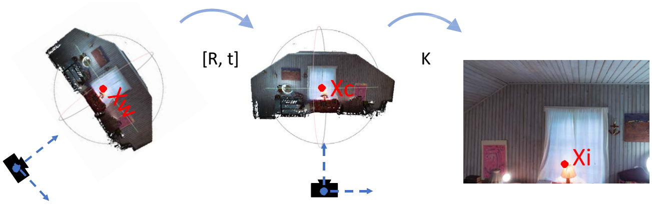


FIGURE 2.4: *The geometry of image formation. A 3D point $X_w$in the world coordinates is mapped to X_c in the camera coordinates by rotation matrix R and translation matrix t. Then X_c is projected to X_i in the image plane by camera intrinsic matrix K.*

The formula 5.4 describes the spatial relationship between 3D points and their corresponding 2D points in the image coordinates. However, for the image captured by a real camera, it is made up of pixels. Thus, it needs to sample and quantize the pixels on the image plane. In order to describe the process that converts the ray into pixels in the image plane, we define a *pixel coordinate system* that is attached on the physical image plane. Without loss of generality, it assumes that the origin of image space lies at top left corner of the bounded image plane. The two axes of the pixel coordinate system are referred to as U and V. The axes U and V are parallel to axes x and y, respectively.

In Equation 5.4, it assumes that the origin of coordinates in the image plane is at the principal point. Between the pixel coordinate system and the image plane, there is a translation of the origin. Therefore, the mapping process from point q in the image plane to point $q' = [u, v]^{\mathrm{T}}$ in the pixel coordinates is described as

$$\begin{cases} u = x' + cx \\ v = y' + cy \end{cases}, \tag{2.7}$$

where $[cx, cy]^{\mathrm{T}}$ are the coordinates of the principal point. Put 5.4 into 2.7, then we can get:

$$\begin{cases} u = f\dfrac{X}{Z} + cx \\ \\ v = f\dfrac{Y}{Z} + cy \end{cases}. \tag{2.8}$$

This equation can also be expressed in homogeneous coordinates as:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \frac{1}{Z} \begin{bmatrix} f & & cx \\ & f & cy \\ & & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}. \tag{2.9}$$

Put Z to the left side and rewrite 2.9

$$\begin{bmatrix} Z*u \\ Z*v \\ Z \end{bmatrix} = \begin{bmatrix} f & & cx \\ & f & cy \\ & & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \triangleq KP', \tag{2.10}$$

where matrix K,

$$K = \begin{bmatrix} f & & cx \\ & f & cy \\ & & 1 \end{bmatrix}, \tag{2.11}$$

is called *camera calibration matrix* and $P' = [X, Y, Z]^{\mathrm{T}}$ is defined in a Euclidean coordinate system with principal axis of the camera pointing straight down the z-axis. Such a coordinate system is called the *camera coordinate frame*.

Camera rotation and translation. Let 3D points be represented in a Euclidean coordinate system called the *world coordinate system*. There is a Euclidean 3D transformation including a rotation and translation, between the world and camera coordinate systems.

Let $X_W = [X, Y, Z]^{\mathrm{T}}$ be a 3D point in the world coordinate system and X_C be the same point in the camera coordinate system. Then we write

$$X_C = RX_W + t, \tag{2.12}$$

where R is a 3×3 rotation matrix and t is a translation matrix. R and t are the extrinsic parameters which denote the coordinate system transformation from 3D world coordinates to 3D camera coordinates. The combination of the 3D rotation and 3D translation matrices is called 6D camera pose. Putting this together with 2.10 leads to

$$Z \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K(RX_W + t). \tag{2.13}$$

This is the general mapping given by a pinhole camera, as described in Figure 2.4. To obtain the image-space 2D coordinates, a divide is necessary. We define the division function *s* as:

$$p_{u,v} = s(X_{x,y,z}) = \begin{bmatrix} \frac{x}{z} \\ \\ \frac{y}{z} \end{bmatrix}, \tag{2.14}$$

where $p_{u,v}$ is 2D point in the image plane converted from its representation $P_{x,y,z}$ in homogeneous coordinates.

Generalized 3D transformations can be represented as 4×4 matrices. Given a 3D point $P_w = [x_w, y_w, z_w, 1]^{\mathrm{T}}$ (a 4-dimensional vector) in the world coordinate system, we can apply the extrinsic matrix H, a 4×4 transformation matrix,

$$H = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}, \tag{2.15}$$

to transform P_w into the camera coordinate system:

$$P_c = HP_w. \tag{2.16}$$

2.1.1.2 IMAGE

Cameras convert the scene information from the 3D world into an image composed of pixels. It determines which parts of the scene will be observed by the camera. To a computer, an image which consists of rectangular pixels is just a grid of numbers and occupies a continuous disk.

Based on the pixel coordinate system we defined in Section 2.1.1.1, the width of the image is defined by the number of columns in the image along U axis; The height of the image is defined by the number of rows in the image along V axis. Let $P = (x, y)$ be a pixel in the pixel coordinate system. It can be recorded as a 8-bit unsigned integer with a range [0, 255]. For a color image, it has three channels and each of them is composed by such pixels. These three channels are associated with three primary colors (red, green and blue), and the value of each primary color indicates how many photons land on the pixel. A depth image which is generated by RGB-D sensors or 3D scanners often uses 16-bit integers ([0, 65535]) to record distance information.

2.1.1.3 DEPTH AND DISPARITY

In 3D computer graphics and computer vision, the depth of a 3D point is obtained by measuring its distance to the plane defined by camera sensors. This is equivalent to the z coordinate of the point in the camera coordinate system. Depth is a fundamental concept used in the scene reconstruction. When reasoning about the image-space motion of 3D points, it is common to reason about their inverse depths, or disparities. *Disparity* is inversely proportional to depth z and is often used for depth calculation.

2.1.2 IMAGE BASED RENDERING WITHOUT GEOMETRY

Image based rendering approaches without the use of geometry project image-space points in reference images to a target image plane to synthesize a new image, which avoids the requirement of scene geometry information. These methods are often based on the *plenoptic function* [AB+91] that is used to describe the intensity of each light ray passing through a certain viewpoint in a scene.

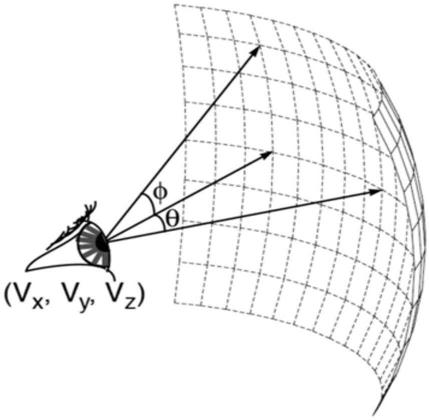


FIGURE 2.5: *Geometric elements of the plenoptic function.*

Let $X = (V_x, V_y, V_z)$ be the position of an idealized eye. From there, a viewing direction can be chosen by an angle (θ, φ) and a band of wavelengths λ. In the case of a dynamic scene, we are also able to choose the viewing time t. Then the plenoptic function is written as :

$$P = \textit{plenoptic}(V_x, V_y, V_z, \theta, \varphi, \lambda, t). \tag{2.17}$$

Figure 2.5 describes the geometric elements of this relationship. If we only consider static scenes with fixed light conditions, we are able to drop out time t and light wavelength λ. We rewrite 2.17:

$$P = \textit{plenoptic}(V_x, V_y, V_z, \theta, \varphi). \tag{2.18}$$

Thus, image based rendering approaches work by explicitly sampling the plenoptic function. McMillan et al. [MB95] use cylindrical panoramas as samples. They provide rendering at discrete locations by stitching images. Similarly, QuickTime VR [Che95] uses a series of cylindrical panoramic images to build a virtual environment. The panoramic image is digitally warped on-the-fly to mimic camera panning and zooming. However, these systems are not able to provide free-viewpoint rendering. Another disadvantage of these systems is that if users change their view positions, the translation between two synthesized images is not smooth. It leads to the fact that the synthesized image often jumps from one position to another just like the google street view.

Light fields [LH96] and lumigraph [GGSC96] systems further reduce the parameters of the plenoptic function to 4. The light rays of the scene are encoded by their intersections with two parallel planes. In the light fields system, densely sampled images are obtained by a capturing rig. It produces new images by querying and interpolating rays.

The lumigraph uses an electronic setup to capture images and allows rendering from arbitrary positions.

Concentric Mosaics [SH99] is a 3D parameterization of the plenoptic function. It can be created by compositing slit images captured at different locations of each circle, as its rendering is constrained along concentric circles on a plane. Compared with light fields and lumigraph systems, synthesizing new images by concentric mosaics is faster, due to the less parameters in the plenoptic function. To overcome the drawback of concentric mosaics caused by vertical distortion, spherical light field [DDB+15] is introduced. It uses fisheye camera to capture outward-looking spherical light fields on a programmable pan/tilt head, as shown in Figure 2.6. The system uses 1728 images to achieve free-viewpoint rendering within the 35cm radius sphere outlined by the cameras. However, the capturing process is complicated, which limits its practical application.

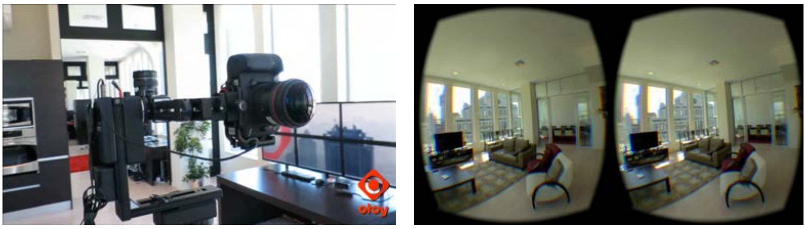


FIGURE 2.6: *The capture system for spherical light field and an example of synthesized images [DDB+15].*

2.1.3 IMAGE BASED RENDERING WITH GEOMETRY

It is well known that image based rendering algorithms that use geometric information can provide better rendering. The technologies from 3D reconstruction are often introduced to estimate feature correspondences or reconstruct point clouds, meshes or depth maps which are used as geometric information for rendering. We will give a brief overview of 3D reconstruction in the next section.

Since geometric information can be represented by correspondences between image pairs, the first step of IBR using implicit geometry is to find these pixel correspondences [CW93]. Based on pixel correspondences, the synthesized image is produced by view interpolation approaches. However, this method is not able to deal with cases when input images have big differences with each other, for the lack of full geometric information. To address this issue, depth information calculated by stereo reconstruction is used to project the centers of input cameras into the new image. Then these projections are triangulated and mapped with textures, which is helpful to improve the quality of the synthesized image. Another solution [BBM+01] is to use a number of weights to overcome limitations caused by the lack of full geometric information. It can achieve some degree of free-viewpoint navigation. The weight function is defined as:

$$w(i) = w_{ang}(i) + w_{dis}(i) + w_{fov}(i), \tag{2.19}$$

where the angle similarity $w_{ang}(i)$ and the distance term $w_{dis}(i)$ are used to select angularly close cameras, and the visibility term $w_{fov}(i)$ avoids choosing rays that fall outside the field of view of the selected camera.

To further improve the quality of the synthesized image, the full depth image is created with the consideration of local geometry information [CDSHD13, HRDB16]. However, such a depth image can reduce artifacts caused by depth discontinuities. Another solution is to use a global point cloud which is obtained by multi-view stereo. The point cloud allows high-quality rendering using the view-specific depth information. Recently, deep learning-based approaches are also used for IBR algorithms. Convolutional neural network(CNN) [FNPS16] is introduced to perform new view synthesis directly from pixels. However, this method is only suitable for narrow-baseline capture. SynSin [WGSJ20] uses geometric information which are estimated by deep learning to further improves the quality of the synthesized image, as shown in Figure 2.7. However, in the current state, learning-based approaches still suffer from blurring and high computational costs.

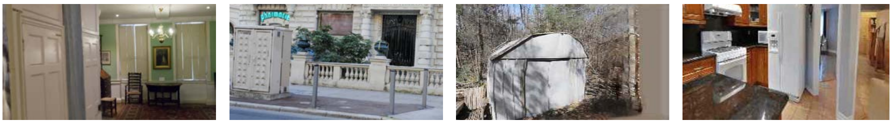


FIGURE 2.7: *Examples of synthesized images from deep learning-based IBR. Left two images are synthesized by learned blending weights [HPP$^+$18] and the right two images are generated by learned point clouds [WGSJ20].*

2.2 3D RECONSTRUCTION

3D reconstruction is the process of creating 3D models or appearance of real objects. It is the inverse process of generating 2D images from 3D scenes. The reconstruction process can be accomplished either by active methods that require special devices, such as laser scanners, rangefinders to capture geometry information or by passive methods that are based on a collection of images. In this thesis, we use passive methods, for they do not require special devices or equipment, which are easily applied in different fields. The passive algorithm combines *Structure from Motion* and *Multi-view stereo*.

2.2.1 STRUCTURE FROM MOTION

Structure from Motion (SfM) is a pipeline that allows 3D reconstruction starting from unstructured image collections. It includes four main stages: feature extraction and matching, pose estimation, triangulation, and bundle adjustment. Here, we give a brief overview of each stage.

2.2.1.1 FEATURE EXTRACTION AND MATCHING.

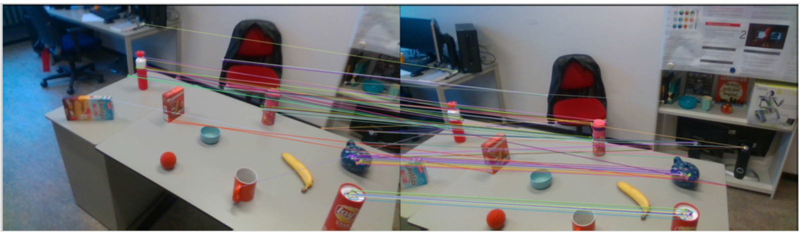


FIGURE 2.8: *The example results of feature extraction and matching.*

Features are used as starting points for SfM algorithms. Good features are repeatable and invariant to rotations, scales, and intensity changes. They are extracted from some interesting points (2D key points) in the image which provide useful information for camera pose estimation and scene reconstruction. Feature descriptors such as SIFT [Low04], FAST [RPD08] and ORB [MAMT15] are often used to describe them. There are many features such as edges, corners, ridges and blobs from which the feature descriptors are computed.

Once we obtain these key points and their descriptions, we can match key points to each other and find correspondences of these points in different images using distance calculation. During this step, some tests and checks are often used to reduce spurious correspondences. For example, the ratio test used in [Low04] avoids poor matches by computing the ratio between the best and second-best match. If the ratio is below some threshold, the match is discarded as being low-quality. The output of this stage is a set of corresponding key points (see Figure 2.8).

2.2.1.2 POSE ESTIMATION

The camera pose consists of 6 degrees-of-freedom (DOF) which is made up of a 3D rotation (roll, pitch, and yaw) and 3D translation of the camera with respect to the world. The camera pose estimation can be classified into three types, including 2D-to-2D pose estimation, 3D-to-2D pose estimation and 3D-to-3D pose estimation.

2D-to-2D pose estimation. Assume that we have a set of 2D point correspondences between two images, we want to estimate the relative pose of the cameras capturing the two images. Figure 2.9 shows a pair of correspondence points, p_1 and p_2, and their sharing 3D point $P = [X, Y, Z]^\mathrm{T}$ in the world coordinate system. p_1 is projected by the left camera represented by its center O_1 and p_2 is projected by the right camera represented by the center O_2. As we can see, the image points p_1 and p_2, 3D point P,

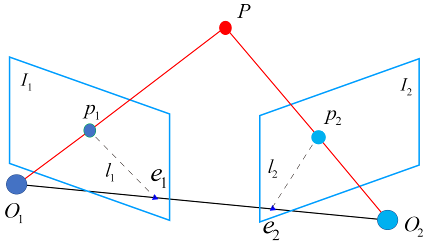


FIGURE 2.9: *Epipolar geometry.*

and camera centers O_1 and O_2 are coplanar. This plane is called *epipolar plane*. The line joining the camera centers O_1 and O_2 is called *baseline* and it intersects image plane I_1 at point e_1 and image plane I_2 at point e_2. The points e_1 and e_2 are called *epipoles*. The line l_1 or l_2 which is the intersection of an epipolar plane with the image plane is called *epipole line*.

Since we do not know anything about the camera positions, without loss of generality, we can set the left camera coordinate system as the world coordinate system. Given camera intrinsic matrix K, the relative rotation matrix R and translation matrix t between two cameras, we get:

$$d_1 p_1 = KP,\ d_2 p_2 = K(RP + t), \tag{2.20}$$

where d_1 and $d_2$are the depth values of points p_1 and p_2. Then set $x_1 = K^{-1} d_1 p_1$ and $x_2 = K^{-1} d_2 p_2$, and put them into 2.20. The relationship between x_1 and x_2 is

$$x_2 = Rx_1 + t. \tag{2.21}$$

To annihilate t on the right hand side, we write 2.21 as

$$[t]_\times x_2 = [t]_\times Rx_1, \tag{2.22}$$

where $[t]_\times$ is skew symmetric corresponding to the cross product with t. Taking the dot product of both sides with x_2^T yields

$$x_2^\mathrm{T} [t]_\times x_2 = x_2^\mathrm{T} [t]_\times Rx_1. \tag{2.23}$$

Since the left hand side is a triple product with two identical entries, the left hand side is 0. Then we rewrite 2.23 to derive the epipolar constraint

$$x_2^\mathrm{T} E x_1 = 0, \tag{2.24}$$

where $E = [t]_\times R$ is a 3×3 matrix called *essential matrix*.

Let $x_1 = [u_1, v_1, 1]^{\mathrm{T}}$ and $x_2 = [u_2, v_2, 1]^{\mathrm{T}}$ be a pair of corresponding points. Based on 2.24 we get :

$$\begin{bmatrix} u_1 & v_1 & 1 \end{bmatrix} \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} \begin{bmatrix} u_2 & v_2 & 1 \end{bmatrix} = 0. \tag{2.25}$$

Given a set of corresponding image points, it is possible to estimate an essential matrix which satisfies the epipolar constraint for all the points in the set. The simplest way to obtain the essential matrix is to find the solution of the least squares problem, commonly known as the eight-point algorithm [Har97]. Now writing

$$E = [e_{11}, e_{12}, e_{13}, e_{21}, e_{22}, e_{23}, e_{31}, e_{32}, e_{33}]^{\mathrm{T}},$$

with 8 correspondences, we can form

$$\begin{bmatrix} u_1u'_1 & u_1v'_1 & u_1 & u_1u'_1 & u_1v'_1 & u_1u'_1 & v'_1 & 1 \\ u_2u'_2 & u_2v'_2 & u_2 & u_2u'_2 & u_2v'_2 & u_2u'_2 & v'_2 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_8u'_8 & u_8v'_8 & u_8 & u_8u'_8 & u_8v'_8 & u_8u'_8 & v'_8 & 1 \end{bmatrix} \begin{bmatrix} e_{11} \\ e_{12} \\ e_{13} \\ e_{21} \\ e_{22} \\ e_{23} \\ e_{31} \\ e_{32} \\ e_{33} \end{bmatrix} = 0, \tag{2.26}$$

where $x_i = [u_i, v_i, 1]^{\mathrm{T}}$ is a key point and $x'_i = [u'_i, v'_i, 1]^{\mathrm{T}}$ is its corresponding point.

Once the essential matrix E has been computed, we are able to determine R and t by performing the singular value decomposition (SVD) [GR71].

3D-to-2D pose estimation. Assume that we already know 3D points in the world coordinate system and their corresponding 2D projections in the image, the problem of estimating the pose of a calibrated camera is called Perspective-n-Point (PnP) pose problem. There are plenty of approaches proposed to solve the PnP problem and the least number of points required to solve this problem is $n = 3$. These methods can be divided into direct and iterative solutions. One of the direct solutions is P3P [GR71], which only uses three pairs of corresponding points to solve the PnP problem.

Let A, B and C be the 3D points in the world, a, b and c be the corresponding 2D points in the image, O be the center of the camera, $\alpha = \angle OAB$, $\beta = \angle OAC$, and $\gamma = \angle OBC$, as shown in Figure 2.10. For triangles $\triangle OAB$, $\triangle OAC$, $\triangle OBC$, we get:

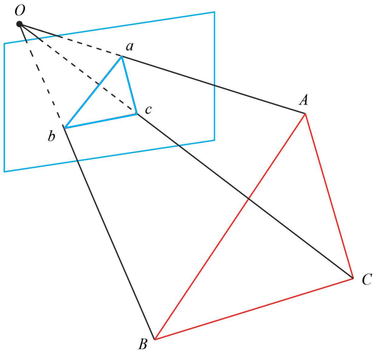


FIGURE 2.10: *The geometry of the P3P problem.*

$$\begin{cases} OA^2 + \ OB^2 - 2OA \cdot OB \cdot \cos\alpha = AB^2 \\ OA^2 + \ OC^2 - 2OA \cdot OC \cdot \cos\beta = AC^2 \\ OB^2 + \ OC^2 - 2OB \cdot OC \cdot \cos\gamma = BC^2 \end{cases} . \tag{2.27}$$

To simplify the equation system, let $x = OA/OC, y = OB/OC, v = AB^2/OC^2$, $uv = BC^2/OC^2$ and $wv = AC^2/OC^2$. 2.27 becomes

$$\begin{cases} x^2 + \ y^2 - 2xy\cos\alpha - v = 0 \\ x^2 + \ 1 - 2x\cos\beta - wv = 0 \\ y^2 + \ 1 - 2y\cos\gamma - uv = 0 \end{cases} . \tag{2.28}$$

Now putting $v = x^2 + \ y^2 - 2xy\cos\alpha$ into 2.28 leads to

$$\begin{cases} (1-u)y^2 - ux^2 - y\cos\gamma + 2uxy\cos\alpha + 1 = 0 \\ (1-w)x^2 - wy^2 - x\cos\beta + 2wxy\cos\alpha + 1 = 0 \end{cases} . \tag{2.29}$$

This equation has two unknown parameters x and y, which can be calculated by solving the two quadratic equations. There are many methods to find the positive solutions for x and y (e.g., Wu-Ritt's zero decomposition method [GHTC03]). After obtaining the solutions, we can get the 3D positions of a, b and c in the camera coordinate system. Then, the PnP problem is converted into the iterative closest point (ICP) problem [Zha94], from which the rotation matrix R and translation matrix t are estimated. In fact, the number of the corresponding points is often more than three. There are also a large number of approaches [LMNF09, LXX12] to solve the PnP problem with more than three points.

Iterative solutions often first estimate an initial pose and then use a Gauss–Newton or Levenberg–Marquardt optimal algorithm to iteratively refine the initial pose to obtain the final one. The most popular algorithm is bundle adjustment which will be described in Section 2.2.1.4.

3D-to-3D pose estimation. Given a set of 3D-to-3D point correspondences, $X = X_1, X_2, ..., X_N$ and $X' = X'_1, X'_2, ..., X'_N$, $X_i \leftrightarrow X'_i$ (the number of these points is N), the transformation is defined by:

$$X'_i = RX_i + t, \tag{2.30}$$

where R is a rotation matrix and t is a translation matrix. To calculate R and t, ICP is widely used, which minimizes the difference between two sets of points.

From 2.30, we get the error function of ICP:

$$e_i = X'_i - (RX_i + t). \tag{2.31}$$

It is possible to solve for R and t by the linear least squares which is defined by

$$\min_{R,t} J = \frac{1}{2}\sum_{N}^{i=1} ||X'_i - (RX_i + t)||^2. \tag{2.32}$$

The SVD provides a convenient way to find the solution for the linear least squares. Let o and o' be the centroids:

$$o = \frac{1}{N}\sum(X_i),\ o' = \frac{1}{N}\sum(X'_i). \tag{2.33}$$

Then, rewriting the right side of 2.33 with o and o' leads to

$$\begin{aligned}
&\frac{1}{2}\textstyle\sum_{N}^{i=1} ||X'_i - (RX_i + t)||^2 \\
&= \frac{1}{2}\textstyle\sum_{N}^{i=1} ||X'_i - RX_i - t - o' + Ro + o' - Ro||^2 \\
&= \frac{1}{2}\textstyle\sum_{N}^{i=1} ||(X'_i - o' - R(X_i - o)) + (o' - Ro - t)||^2 \\
&= \frac{1}{2}\textstyle\sum_{N}^{i=1} ||(X'_i - o' - R(X_i - o))||^2 + +||(o' - Ro - t)||^2 \\
&+2(X'_i - o' - R(X_i - o))^T(o' - Ro - t)
\end{aligned} \quad . \tag{2.34}$$

Since $2(X'_i - o' - R(X_i - o))^T(o' - Ro - t) = 0$, we can get

$$\min_{R,t} J = \frac{1}{2}\sum_{N}^{i=1} ||(X'_i - o' - R(X_i - o))||^2 + ||(o' - Ro - t)||^2. \tag{2.35}$$

From 2.35, we obtain R that satisfies

$$R^* = \arg\min_R \frac{1}{2} \sum_N^{i=1} ||(X_i' - o' - R(X_i - o))||^2. \tag{2.36}$$

Then, SVD can be used to solve2.36. Based on R, t^* is given by

$$t^* = o' - Ro. \tag{2.37}$$

Besides, ICP can also be solved by nonlinear optimization methods. More detailed descriptions of these approaches can be found in [PCS+15, GIRL03, RL01].

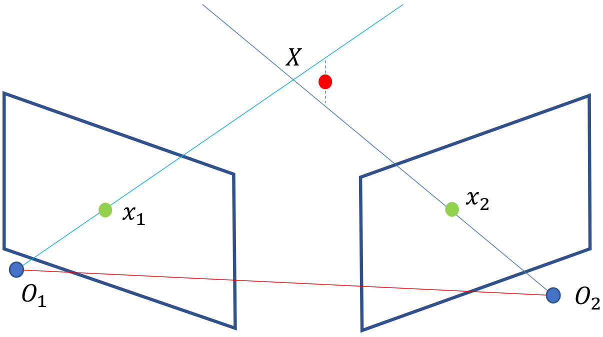


FIGURE 2.11: *Triangulation. Two rays passing camera centers O_1 and O_2, and 2D points x_1 and x_2 will intersect in a 3D point X in ideal case. However, due to errors, the intersection point is not X.*

2.2.1.3 TRIANGULATION

The problem of estimating the 3D location of a point from a set of corresponding 2D points in images is known as *triangulation*. Let x_1 and x_2 be two 2D points in two matching images, respectively. According to the epipolar constraint, the two rays back-projected from image points x_1 and x_2 are in a common epipolar plane, that is, a plane passing through the two centers of camera O_1 and O_2. For the two rays lie in a plane, they will intersect in point X in ideal case. However, due to sensor noises and projection errors, the positions of 2D points are inaccurate, which results in the fact the two rays do not intersect with each other, as shown in Figure 2.11.

One stable approach to solve this problem is the direct linear transform method (DLT) [HZ03] by finding the 3D point X that lies closest to all the rays back-projected from 2D image points. For points x_1 and x_2, this amounts to minimize the distance

$$||x_1 - s(K_1 H_1 X)||^2 + ||x_2 - s(K_2 H_2 X)||^2, \tag{2.38}$$

where K_1 and H_1 are the intrinsic and extrinsic matrices of camera O_1, K_2 and H_2 are the intrinsic and extrinsic matrices of camera O_2, and *s* is the division function (see 2.14).

2.2.1.4 BUNDLE ADJUSTMENT

Given a set of 3D points X_i captured by a set of cameras and their corresponding 2D projections x_i in the images, we are able to calculate accurate poses P_i for the cameras. However, if image measurements are noisy, we are not able to get accurate feature points. Furthermore, matched features often extend over a sequence of images. This means that the pose of a single camera may have many estimates by using different 2D to 3D correspondences, and the 3D points calculated by triangulation also have different values. As a result, with the camera intrinsic matrix K_i, the equations $x_i = s(K_i P_i X_i)$ will not be satisfied exactly. To address this issue, pose estimation approaches often perform *bundle adjustment* [TMHF99] which is a robust non-linear minimization of measurement errors. It is able to refine the 3D points, the camera poses and the intrinsic parameters of the cameras used to capture images.

Bundle adjustment can be summarized as minimizing reprojection errors between 2D projections of 3D points and 2D locations of their corresponding features. For this reason, it is expressed as a function containing motion and structure parameters:

$$\operatorname*{arg\,min}_{(P_i', K_i', X_j')} \sum_{i,j} \varphi(x_j^i - s(K_i' P_i' X_j')), \tag{2.39}$$

where P_i' is the estimated projection matrix, K_i' is the estimated camera intrinsic matrix, X_j' is the estimated 3D point, x_j^i is the 2D location of i in image j, *s* is the division function 2.14, and $\varphi(x, y)$ is the geometric image distance between the image points represented by x and y.

It attempts to fit a nonlinear model to the point correspondences. Its solution can be achieved using nonlinear least-squares algorithms, such as Levenberg–Marquardt [Mor78] which has proven to be one of the most successful algorithms.

2.2.2 MULTI-VIEW STEREO

Multi-view stereo (MVS) is often applied for a dense reconstruction from images with estimated 6D poses and spare 3D points estimated from SfM. In the section, we give a brief insight into three main stages of MVS: Stereo matching, Depth estimation, and Fusion.

2.2.2.1 STEREO MATCHING

Stereo matching is the process of finding dense correspondences in different images that correspond to the same 3D point in the scene. It uses the epipolar constraint that reduces the search regions to find pixels with similar appearance on the epipolar lines.

The most widely used matching method that allows finding the conjugate points is block matching. Correspondences are found by comparing photo-consistency which estimates the likelihood of two pixels (or groups of pixels) being similar. The photo-consistency measurement operates by comparing a small region centered at a pixel with congruent regions extracted from neighboring images. Given a 3D point X and a set of images containing the projected point $\pi_i(X)$ from X, the photo-consistency is defined by

$$C_{i,j} = \phi(P_i(\Omega(\pi_i(X))), P_j(\Omega(\pi_j(X)))), \tag{2.40}$$

where $\phi(x, y)$ is a similarity measurement that compares the two vectors x and y, $\Omega(x)$ is a patch center at point x, P_i is the image intensities of the patch Ω, i and j are a pair of images.

One of the key factors of photo-consistency measurement is how to define the metric ϕ and patch Ω. The straightforward way to define Ω for each pixel is to use a square grid of pixels centered at that pixel. 3×3 or 5×5 is the most widely used patch size. For more complicated scenarios, the size and shape of the patch are not constant [ZPQL04, FP09].

photo-consistency metrics. The metrics often used for photo-consistency measurement includes Sum of Square Differences (SSD), Sum of Absolute Difference (SAD), Zero-mean normalized cross correlation (NCC), Rank and so on [FH15]. In this thesis, one of the metrics we use is SAD. Give a patch centered at pixel x in an image A, we project x_i in the patch by camera's intrinsic and extrinsic matrices to x' in another image B. Then we compute the photo-consistency by SAD:

$$C_{SAD} = \sum_{1}^{N} |\rho_A(x_i) - \rho_B(x_i')|, \tag{2.41}$$

where functions ρ_A and ρ_B extract colors from the images A and B, x_i' is the corresponding point of x_i in the image B, and N is the number of pixel in the patch. It achieves good performance for applications that can guarantee similar capture conditions for different images (e.g., real-time or mobile applications).

The photo-consistency metrics described above assume all the points be visible in all views when compute photo-consistency. However, in the general case some points may be occluded by other points, so that one does not know which points are visible in which images (See Figure 2.12). In order to select images that are able to capture the 3D geometry for the photo-consistency computation, we need the correct 3D geometry. However, this model is unknown and is what we want to produce. Techniques breaking this loop can be divided into geometric and outlier based approaches.

Geometric approaches explicitly model occlusion to determine which scene structures are visible in which images. One common approach is to use the current reconstructed geometry to compute occlusion, select which views see which parts of the geometry, and iterate visibility estimation and reconstruction [Kut00, DP11]. To improve the performance, images which have similar view angles, small baselines and large overlaps are often clustered. The view clustering process can be achieved by using the 3D

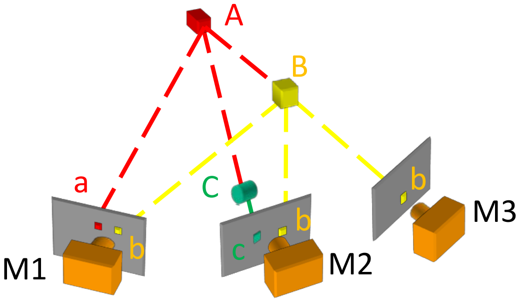


FIGURE 2.12: *Visibility problem. Since point A is occluded by point B and C, it cannot be captured by cameras $M2$ and $M3$.*

points and 6D camera poses estimated from SfM. The 3D points are used to compute the number of shared matches between any two views as an indication of the overlap between them; The 6D pose is used to compute the angle and distance between any two views.

Instead of modeling occlusion geometrically, the outlier based methods treat occlusion as outliers [GS05]. The intuition behind it is that dissimilar images would yield poor photo-consistency scores. In order to select views that may have high similarity, the outlier rejection [Ste99] is often applied to increase the percentage of possible inliers.

2.2.2.2 DEPTH ESTIMATION

Depth estimation in MVS from a pair of images that are often captured by a two-camera rig has the same formulation with the traditional two view stereo. The main advantage of capturing images by a two-camera rig is that it provides more accurate depth values. This is because the camera intrinsic and extrinsic parameters can be carefully calibrated in advance. Besides, for we only need to compute photo-consistency between two images, it achieves high speed.

In order to compute the depth value for every pixel, we use epiploar constraint which can reduce the search regions to find corresponding pixels in two images. Let x_1 be a pixel in the left image, l_2 be the epipolar line in the right image, as shown in Figure 2.13. Based on the epiploar constraint, we can find the projection of x_1 which has the highest photo-consistency cost in the right image along the epioplar line l_2. Then, we use triangulation to estimate the depth d for x_1.

Depth estimation from multi-view images is more complicated than two-view approaches, as there are often more images, resulting in more redundancy. Given a collection of images and their corresponding camera poses, traditional depth estimation approaches first compute photo-consistency through a possible depth range. And then the depth value with the highest photo-consistency score is chosen as the estimated depth.

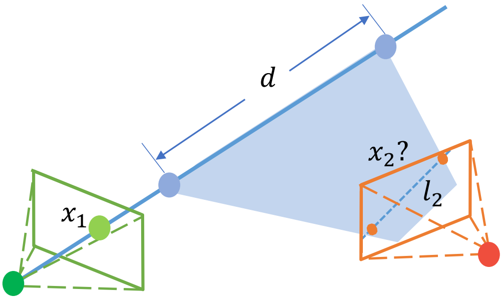


FIGURE 2.13: *Epipolar searching. The projection x_2 of point x_1 can be found along epipolar line l_2*

A well known method to compute depth values from multiple images is *plane sweep stereo*.

The plane sweep computes photo-consistency at a discrete integer depth for each pixel in a reference image. This is equivalent to computing photo-consistency on fronto-parallel or oriented planes with respect to a reference image. For the fronto-parallel plane, a patch centered at a pixel shares the same depth value. However, if the patch contains slanted surface, the depth values in such a patch are different. The oriented plane is introduced to deal with this issue. To speed up depth estimation, recent approaches [BRR11, GLS15] find good depths by iteratively propagate the current estimate of a pixel to its neighboring pixels rather than initializing depth with a range of depth values and exhaustively computing photo-consistency at every possible depth.

Smoothness. The depth estimated by photo-consistency may be incorrect, due to the noise caused by occlusion, large texture-less regions and non-Lambertian reflectance. To address these challenges, many approaches consider the spatial consistency of all the pixels in the image and enforce smoothness during the geometry reconstruction. The Markov Random Field (MRF) based approach is one of the most successful approaches, which works under the assumption that neighboring pixels have similar depth values. It works by finding a depth value d_i for each pixel i, while minimizing the following cost function:

$$E(d_i) = \sum_i E_c(d_i) + \sum_{i,j \in N} E_s(d_i, d_j), \tag{2.42}$$

where E_c is the inverse cost of photo-consistency for pixel i, E_s is used to enforce smoothness, and N is the number of all pairs of neighboring pixels i and j.

The definition of the smoothness function has various forms, e.g., the Potts function:

$$E_s(d_i, d_j) = \begin{cases} 1, & if\ d_i = d_j \\ 0, & else \end{cases}, \tag{2.43}$$

which only encourages to exact label matches, or a truncated linear function:

$$E_s(d_i, d_j) = min(|d_i - d_j|, W), \tag{2.44}$$

where $W > 0$ is a truncation factor.

There are many efficient approaches to solve 2.42. If the smoothness cost at every set of neighboring pixels satisfies the modularity condition [KZ04], the alpha-expansion algorithm [KZ04] is the most popular choice to minimize the cost function. Besides, Semi-Global Matching method [Hir06] is also a good choice, which achieves good performance for dense reconstruction. Apart from MRF based methods, the estimated depth can be smoothed by edge-preserving filters, such as bilateral and guide filters. These filters can remove noise while respecting edges and structures by replacing the depth value d_j of each pixel with a weighted average valued d'_i calculated from depth values of nearby pixels:

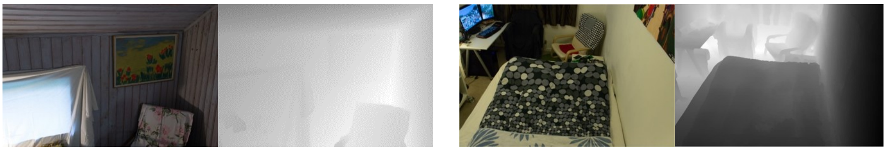


FIGURE 2.14: *Examples of estimated depth images after smoothness from [HRDB16].*

$$d'_i = \sum_{j \in N(i)} W_{i,j}(I) d_j, \tag{2.45}$$

where the weight function $W_{i,j}$ depends on the image I, i and j are pixel indexes, and $N(i)$ is a local window around the pixel i.

The weight function $W_{i,j}$ has different types when it is used in different edge-preserving filters. For example, the joint bilateral filter [ED04] uses the weight function consisting two Gaussian kernels

$$W_{i,j} = exp(\frac{-||i - j||^2}{\sigma_s^2}) \cdot exp(\frac{-||I_i - I_j||^2}{\sigma_c^2}), \tag{2.46}$$

where i and j are the pixel coordinates, I_i and I_j are color values, σ_s adjusts the sensitivity of spatial similarity and σ_c determines the sensitivity of the filter to image edges. The guide filter further improves the running time of the smooth process by using a constant time weight function

$$x'_i = k_j x_i + b_j, \forall i \in w_j, \tag{2.47}$$

where x'_i is the filtered value, x_i is the initial value, and k_j and b_j are some linear coefficients which are constant in the local window w_j centered at the pixel j. They can be calculated by the linear ridge regression model [FHT01, DS98].

After smoothness, we obtain the depth map which is a very popular scene representation, as shown in Figure 2.14. It can be used for vision-based tasks such as object recognition, pose estimation, and scene analysis.

2.2.2.3 FUSION

Even though depth maps contain the geometric information of the scene, a global representation for geometry, e.g., a global point cloud, is also useful. It can be generated by merging multiple depth maps and in return, the global geometry is helpful to remove noise in the depth map. Figure 2.15 shows a 3D point cloud reconstructed by [SZFP16].

Point cloud reconstruction approaches often rely on the geometric consistency assumption. It reconstructs the point cloud with the consideration of neighboring pixels rather than reconstructing each pixel independently. In order to make the point cloud to be representative, when merging depth maps, we often use filters to discarding pixels whose: (1) depth value is much bigger than the depth range, (2) estimated 3D location is only visible in one image, and (3) estimated 3D location has no neighboring points.

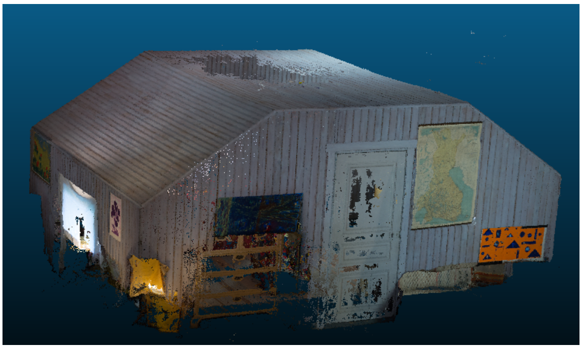


FIGURE 2.15: *The point cloud generated based on depth images [SZFP16].*

For IBR approaches, they also convert the point cloud to polygon mesh or triangle mesh models instead of rendering it directly. The converting process is commonly referred to as surface reconstruction.

Here, we briefly introduce the popular volumetric surface reconstruction method which is robust against noisy point clouds. The key factor of surface reconstruction is how to represent each 3D point in the scene. One way is to label each 3D point as exterior or interior. If the space surrounding a 3D pixel is empty, the pixel is called exterior and vice versa. The problem to produce a surface model can be considered as a binary segmentation problem, where the boundary between exterior and interior can be extracted as a surface model. A 3D MRF based method is often used for the label assignment, where the space behind the depth map pixel is encouraged to be interior and camera positions are forced to be exterior. To compute the segmentation, some methods discretize the scene with a regular voxel gird, and some approaches are based on a Delauney tetrahedralization of the scene [LPK07, JP11].

2.3 ROBOT OPERATING SYSTEM

Robotics is hard to learn as the scale and scope of it continues to grow, and it takes plenty of time for the software developer to write a software for a robot. Even though many people work in the robot software development, their robotics frameworks are only suitable for their own robots. As a result, these softwares eventually become non-reusable and quickly forgotten. To address these issues, the Robot Operating System (ROS) is proposed. The primary goal of ROS is sharing and collaboration, which supports code reuse in robotics research and development. Apart from that, there are some other main goals of ROS [QCG+09]:

•Thin: ROS encourages algorithms designed for robotics applications to occur in standalone libraries. Thus, codes written for ROS can be extracted easily and reused with other robot software frameworks. On the other hand, ROS reuses codes from many other open source projects, e.g., image processing algorithms from OpenCV [BK08] by only exposing configuration options.

•Language independence: ROS supports different modern programming languages. It has already been implemented in Python, C++, Octave and Lisp with other language ports in various states of completion. It allows user to write codes with their preferred programming languages, which is helpful to reduce programming and debugging time, and improve running-time efficiency.

In fact, ROS is a middleware rather than an operating system, which is responsible for handling communication between programs in an existing operating system for robotics applications. ROS uses a Unix-like system, (e.g., Ubuntu) as its main operating system. It provides libraries and tools for software developers to create their robotics applications using an existing foundation rather than doing everything themselves.

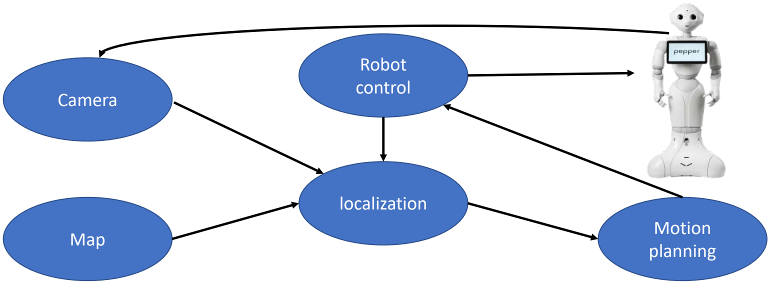


FIGURE 2.16: *An example of the ROS graph structure showing a robot application with various nodes.*

2.3.1 ROS GRAPH STRUCTURE

The basic computation graph concepts of ROS are nodes, topics, services and parameter servers.

Nodes. ROS processes are represented as nodes in a graph structure [ros]. Figure 2.16 shows an example of such a graph structure. A *node* is an executable that performs computation and communication with each other by communication tools including topics and services.

Topics. *Topics* are used for sending data streams between two or more nodes. The data steam is called message which has various types and can be defined by users. If a node interests a certain message, it can subscribe to the message's corresponding topic. For example, if the position of a robot is required, the node (subscriber) can subscribe the topic which sends position information. The node that contains the position topic is called the publisher. Publishers and subscribers on a ROS topic are anonymous. That means no node knows which node is sending a topic, only if it is receiving that topic and vice versa. Besides, a node can publish or subscribe to multiple topics.

Services. Nodes can also communicate with each other by services. A *service* allows creating a client or server system which has a defined request or response. A node offers a service under a string name, and a client calls the service by sending the request message and waiting for the reply. This process is synchronous, where the client sends a request, and blocks until it receives a response. It is useful to obtain specific data such as capturing a single-frame image from a sensor. It can also be used for quick actions, e.g., enable or disable an actuator. Besides, a service server can only exist once, but can have many clients.

Parameter server. ROS provides users a *parameter server* to create global settings. The parameter server is a collection of configuration information that can be accessed through nodes or launch files which are XML configuration files. It is a shared, multivariate dictionary that is able to be accessed at anytime in anywhere in the current ROS environment (See Figure 2.17). Nodes use this server to store and retrieve parameters at running time and launch files use this server to set parameters. The parameter server allows a variety of global settings including the name of a robot, the weight of a robot, the frequency of sensors, and the simulation flag which is used to tell the robot is running in the real mode or simulation mode. As the parameters are not used for high performance, they are designed to be static and globally available values which can be integers, floats and non-binary values. The benefit of the parameter serve is clear. If the user does not have a way to save global settings, it would take plenty of time for the user to hardcode the settings in every node.

Based on nodes, messages, topics, services, and parameters, robotics applications are able to be developed. Application in ROS is organized in *packages* which contain ROS nodes, configuration files, a third-party piece of software, or anything else that logically constitutes a useful module. The goal of designing a package is similar with the aim of ROS, which is to provide useful functionality in an easy-to-consume manner, so that software can be easily reused.

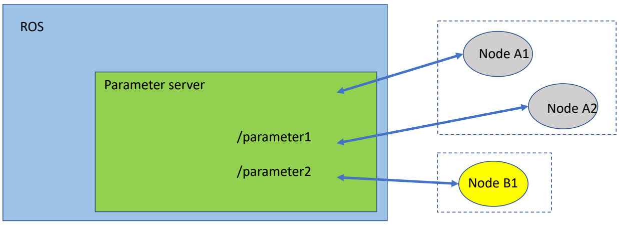


FIGURE 2.17: *The parameter server. It contains two parameters which can be accessed at anytime by the ROS nodes A1, A2 and B1.*

2.3.2 LIBRARIES AND PACKAGES

Apart from those features mentioned above, ROS has plenty of existing ROS libraries and packages that can be directly added to the user's own codes.

For example, to develop a virtual environment for a mobile robot, we need to design many modules, including:

- robot modeling,
- driver for the motors and sensors,
- path and motion planning, and
- pose estimation.

Since writing codes for every module takes a large amount of time, developing such a software is challenging. However, the problem can be solved with the help of ROS. To develop such a virtual environment, there are many libraries or packages in ROS can be used:

- URDF library which contains a parser for Unified Robot Description Format (URDF) which is an XML file to represent a 3D model of the robot,
- motor and sensor driver packages that have various drives,
- Moveit providing different algorithms for path and motion planning and collision checking, and
- tf which is a package that provides multiple coordinate frames over time, such as a world frame, head frame, and camera frame, as described in Figure 2.18.

Apart from the libraries and packages mentioned above, there are other commonly used packages include rosbridge, amcl, slam toolbox, and navigation. The functionalities of these packages cover a wide range of applications including object recognition, visual navigation, image processing, simulation and motion control.

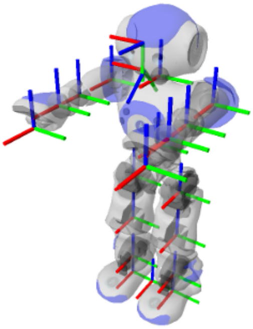


FIGURE 2.18: *The tf transformation tree of Nao robot in ROS.*

2.3.3 TOOLS

The main functionalities of ROS can be augmented by a variety of tools which allow developers to focus on the key features of their applications rather than doing everything. Like any other algorithms used for robotics applications, these tools are accessible in packages, and they work together with the core functionalities of ROS to help developers efficiently create robotics applications. In the following, we describe the most commonly used tools.

Catkin. Catkin is the ROS build system based on CMake, and uses CMake macros and Python scripts to build ROS packages. It has replaced rosbuild which is used for old ROS versions, for it can provide better distribution of packages, better cross-compiling support and is language-independent. The workflow of catkin is similar with that of CMake, but allows building multiple and dependent projects simultaneously.

Roslaunch. Roslaunch is used to launch different remote or local ROS nodes at the same time. Besides, it can be used to set parameters on the ROS parameter server and restart processes which have been dead during execution. Roslaunch uses XML format files to specify the nodes that should be run, parameters that should be set, and the machines which they should run on. It is useful to convert a complicated startup and configuration process into a single command.

Rviz. Rviz is a 3D visualizer for ROS. It allows users to visualize robot models, the global environments, the positions of robots, and the robot's sensor data. It is helpful to debug a robot application by visualizing what the robot is seeing and doing. Rviz displays 3D sensor data from stereo cameras, lasers, and other 3D devices in the form of point clouds or depth images. 2D sensor data from RGB cameras and 2D laser rangefinders can be viewed in Rviz as image data. This can be useful to develop and debug computer vision tasks for robots.

Bag. Bag is a format for saving and playing back ROS data. For example, bags can be used for storing color and depth information captured by color and depth sensors. To exhaustively capture a dense grid of viewpoints in a scene takes a large amount of time and is a labor-intensive task, while a mobile robot can complete this work easily. With the information saved in the bag file, it saves plenty of time for users to develop and test algorithms.

2.3.4 THE ROBOT OPERATING SYSTEM COMMUNITY

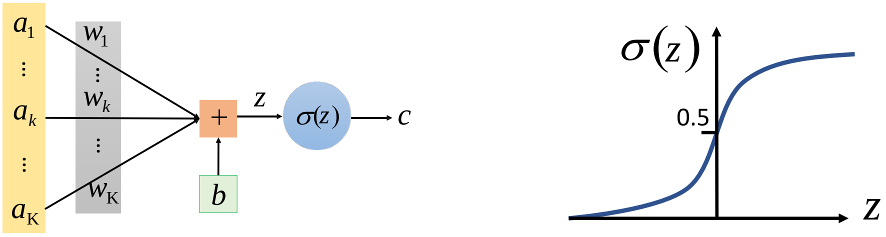
ROS has a huge and growing community including various users all over the world. It not only attracts users who are in research labs but also those working in companies. Many companies are sponsoring some open source projects related to ROS, especially in industrial and service robotics. That is a great guarantee which reduces the user's worries about the project not being supported in a few months. The ROS community enables developers to exchange software and knowledge and benefit from each other. These main online communities include:

- ROS Wiki. It is the main forum to find most of the tutorials, concept explanations, and guides for various robotics applications. Anyone can sign up for an account, write tutorials for their packages, and provide documentation of their algorithms.

- ROS Answers. It is a Q & A site. When users have ROS-related technical questions, they can ask questions in this site. Besides, it also provides a large number of answered questions that may be helpful.

- GitHub. GitHub provides a large amount of ROS packages. From it, users are able to browse source codes, download and make contributions to them.

2.4 DEEP LEARNING



(a) (b)

FIGURE 2.19: *(a): A neuron takes inputs a_i that are multiplied by weights w_i and a bias b to generate the output c through an activation function $\sigma(z)$. (b): The activate function for (a) is sigmoid function.*

2.4.1 NEURAL NETWORKS

Neural networks are computational models that are inspired by biological neural networks to process information.

2.4.1.1 NEURONS

The basic computational unit in a neural network is the *neuron*. The neuron multiples each input x_i with its corresponding weight w_i, adds an overall bias b, and then produces output c using an activation function f, as described in Figure 2.19(a):

$$c = f(\sum_{i}^{N} x_i + b). \tag{2.48}$$

The activation function is nonlinear, and allows computing nontrivial problems using only a few neurons. The sigmoid function is the commonly used activation function, as shown in Figure 2.19(b):

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{2.49}$$

There are other two commonly nonlinear activation functions: Rectified linear unit [NH10],

$$ReLU(z) = \begin{cases} 0, & z \leq 0 \\ z, & z > 0 \end{cases}, \tag{2.50}$$

and Leaky rectified linear unit (LeakyReLU) [MHN13],

$$LeakyReLU(z) = \begin{cases} \alpha z, & z \leq 0 \\ z, & z > 0 \end{cases}, \tag{2.51}$$

where α is constant gradient, which is often set to be 0.01.

2.4.1.2 NEURAL NETWORK ARCHITECTURES

Neural networks often consist of distinct *layers* that are organized by neurons. The most commonly used layer is the fully connected layer where neurons between two adjacent layers have fully pairwise connections, but neurons within a single layer share no connections. Figure 2.20 shows an example of the fully connected network.

It consists of an input layer, several hidden layers and an output layer. The neurons in the input layer pass external data, such as images to the network and no computation is performed in this layer; The hidden layers compute and transfer information to the neurons in the next hidden layer; The output layer is used for computing and transferring information from the network to the external world. In this neural network, the output

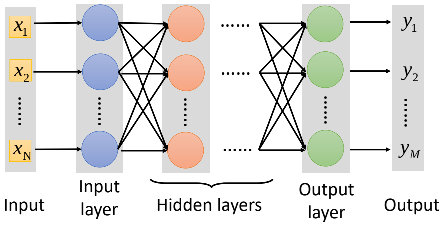


FIGURE 2.20: *A fully connected neural network with input, hidden and output layers.*

from one layer is used as input to the next layer. Such neural networks are called *feed-forward neural networks* and sometimes referred to as *Multi-Layer Perceptrons (MLPs)*. Feedforward means there are no loops in the network and information always moves forward from the input neurons, through the hidden neurons to the output neurons.

Besides, the word ”deep” in deep learning means the number of hidden layers. Training a deep neural network is to find a set of weights and biases for the neurons that minimize a loss function.

2.4.2 LOSS FUNCTIONS

The loss function is a differentiable function that optimizes network weights in a neural network model. The loss function produces a single scalar non-negative value indicating how well the network weights accomplish the task which the network is designed for.

There are mainly two classes of losses. One is the classification loss which is often used for training a neural network to predict a categorical variable (e.g., a class label). One of the most popular classification losses is the cross-entropy loss used for multi-class classification. It measures the performance of a classification model whose output is a probability value between 0 and 1. To ensure the sum of all probability values z_i is equals to 1, the softmax function is used, which is defined by:

$$softmax(z)_i = \frac{e^{z_i}}{\sum_{j=0}^{k} e^{z_j}}, \tag{2.52}$$

where k is the number of classes and i is some class. Its outputs are positive values representing the estimated class probabilities. The cross-entropy loss is the negative logarithm of the estimated class probability for the true class i:

$$P = \text{-}log(\frac{e^{z_i}}{\sum_{j=0}^{k} e^{z_j}}). \tag{2.53}$$

The other one is the regression loss. Regression is used to predict a continuous value (e.g., object depth) rather than a categorical one (e.g., object class). Regression losses are computed by performing direct comparisons between the predicted value and the true value. The L_1 loss and L_2 loss are often used to measure the differences.

The L_1 loss is formulated by summing the absolute value between network output z' and the true value z:

$$C = |z' - z|. \tag{2.54}$$

The L_2 loss computes the squared difference:

$$C = \|z' - z\|_2^2. \tag{2.55}$$

The loss function plays an important role on deep learning-based tasks. Other specialized loss functions have been designed for object segmentation [XSNF17, RSEG20], pose estimation [WXZ+19] and depth estimation problems [FGW+18].

2.4.3 TRAINING NEURAL NETWORKS

The aim of training a neural network is to find parameters $P^* = w_1, w_2, ..., b_1, b_2, ...$ that minimize the loss function:

$$P^* = \text{arg min} \sum_i C_i, \tag{2.56}$$

where C_i is the loss function for input i.

One simple way to find the set of parameters P^* that minimize the loss function, is random search: try out many sets of parameters, calculate the loss and keep updating the best set of parameters. Even though this strategy may take plenty of time and computation, a set of weights with a small loss can be eventually found. In fact, a better result can be obtained by iterative refinement. That is the training is started with a randomly selected set of weights and then iteratively refine these parameters over time to get lower loss. Apart from above approaches, the gradient descent method achieves much better performance.

2.4.3.1 GRADIENT DESCENT

Gradient descent is an iterative optimization approach that finds a local minimum of the loss function by evaluating the gradient. The gradient can be expressed as the partial derivative of the loss function $f(x)$ with respect to its input x. If the loss function is one dimension, its mathematical expression is defined as

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}, \tag{2.57}$$

where h is a small change near x.

Consider a weight w, the w with a positive gradient will lead to an increased loss while w with a negative gradient results in the decreased loss. The gradient descent algorithm is to repeatedly evaluate the gradient and then to update the weight in the negative gradient direction:

$$w \longrightarrow w' = w - \beta \frac{\partial C}{\partial w}, \tag{2.58}$$

where w' is the updated weight, C is the loss function, β is a small and positive parameter which is called the learning rate. This procedure is repeated until $\dfrac{\partial C}{\partial w}$ is approximately small.

To make gradient descent work correctly, the learning rate should be chosen carefully. If it is too large, training will become unstable and even never converge. However, if it is too small, training will converge very slowly. Another important factor is the gradient computation. The most commonly used approach is the backpropagation algorithm [RHW86].

2.4.3.2 BACKPROPAGATION

The *backpropagation* approach works by computing gradients of loss functions through recursive application of chain rule. The chain rule states that the derivative of a composite function $f(u(v(x)))$ can be represented by multiplication:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial u} \frac{\partial u}{\partial v} \frac{\partial v}{\partial x}. \tag{2.59}$$

Backpropagation uses this rule to calculate partial derivatives of the loss functin C with respect to weights w_l in layer l. The loss function passes through N layers with activations a_l. The calculation is defined by:

$$\frac{\partial C}{\partial w_l} = \frac{\partial C}{\partial a_N} \frac{\partial a_N}{\partial a_{N-1}}, \ \ldots, \ \frac{\partial a_l}{\partial w_l}. \tag{2.60}$$

The main steps of the backpropagation algorithm are summarized below:

(1) Forward propagation: The activations in each layer are calculated and stored by forward propagation.

(2) Backpropagation and weight update: The output errors are calculated and propagated back through the network using backpropagation to calculate gradients. Then the gradient descent is used to optimize all weights in the network with an aim of reducing the error at the output layer.

2.4.4 VARIANTS OF NEURAL NETWORKS

In this section we give a brief overview of the most popular neural network architectures for computer vision-based tasks.

2.4.4.1 CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (CNNs) are a class of neural networks that have been successfully used in many computer vision-based tasks, such as object segmentation, object segmentation and image classification. A CNN mainly consists of three types of layers: *convolutional layers*, *pooling layers*, and fully connected layers. We have described the fully connected layer in Section 2.4.1.2.

Convolutional layers. A convolutional layer is a locally connected layer in which a neuron is only connected with neurons that are in the same spatial neighborhood with shared weights in the next layer. It is often used to extract features from input images. The main advantage of the convolution layer is that it preserves the spatial relationship between pixels by learning image features using small squares of input data. Apart from that, it reduces the possibility of overfitting by using sharing weights to reduce the number of parameters involved in a convolutional network.

Pooling layers. Pooling layers are usually used between two convolutional layers. Their functionality is to simplify the information in the output obtained from the convolutional layers by taking an average, minimum or maximum of activations over a small spatial region.

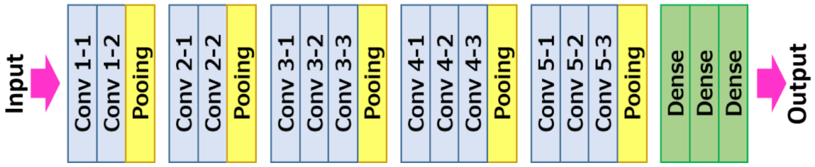


FIGURE 2.21: *VGG*

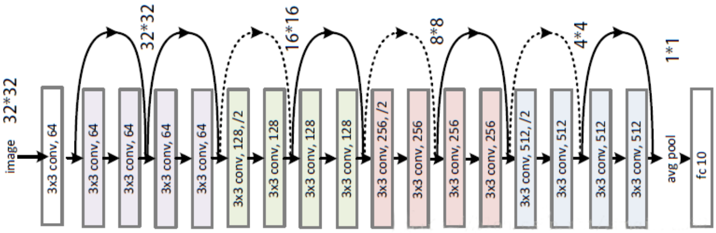


FIGURE 2.22: *Resnet*

There are a large number of architectures in the field of CNNs. Here we give two examples:

1. VGGNet. VGGNet is developed by Simonyan et al. [SZ14]. Their best network consists of 16 convolutional layers, max-pooling layers and fully connected layers, as shown in Figure 2.21. The convolutional layer only contains 3×3 filters to keep the

architecture homogeneous. The main contribution of their work is that they show increasing the depth of the network and the training time are able to significantly improve the performance.

2. ResNet. The main idea of ResNets [HZRS16] is that it introduces shortcut connections to the network, as described in Figure 2.22. With these shortcut connections, the layers in the network only need to learn a residual mapping function rather than a unreferenced function. Experiments show that these residual networks are easier to optimize and allow training deeper networks with lower complexity.

2.4.4.2 GRAPH NEURAL NETWORKS

Graph Neural Networks (GNNs) are designed to process graph structured data, such as social networks, physical systems, molecular structures, web graphs and knowledge graphs, as shown in Figure 2.23. Even though CNNs are able to extract meaningful features from Euclidean data, such as images, they are not able to handle the graph input properly. Besides, they are not able to learn the reasoning graph from large experimental data. GNNs are proposed to deal with these issues.

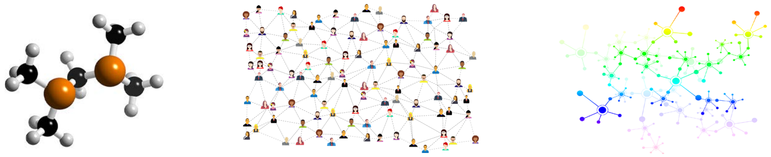


FIGURE 2.23: *Examples of graph structured data*

Graphs are a kind of data structure consisting of nodes which are connected by edges [SGT$^+$08]. The nodes represent objects and the edges represent relationships between these nodes. The aim of GNN is to learn a node embedding for each node, which contains the neighborhood information. Let f and g be parametric functions. f is called *local transition function* which is used to learn a node embedding state m_v for each node, representing the dependence of a node on its neighborhood. g is called *local output function* which is used to obtain output embedding o_v. The relationship between nodes and edges is defined by:

$$m_v = f(x_v, x_{co[v]}, m_{ne[v]}, x_{ne[v]}), \tag{2.61}$$

$$o_v = g(m_v, x_v), \tag{2.62}$$

where x_v, $x_{co[v]}$, and $x_{ne[v]}$ are the features of node v, edges connected to v, and v's neighborhood nodes, respectively. $m_{ne[v]}$ are the embedding states.

Let M, O, X, and X_n be the matrices constructed by stacking all the states, outputs, nodes, and node features. Then we have a compact form as:

$$M = F(M, X), \tag{2.63}$$

$$O = G(M, X_n), \tag{2.64}$$

where M is the global transition function, G is the global output function and n is the number of nodes in the graph. Based on Banach's Fixed Point theorem [KK11], the node state is updated by:

$$M^{l+1} = F(M^l, X), \tag{2.65}$$

where M^l is the l^{th} iteration of M. With the GNN model, the parameters of f and g can be learned by feeding these states into loss functions and running gradient descent to train the these parameters. Here we give two commonly used variants of GNNs which extend the representation capability of the GNN model. For a more extensive overview, we refer to [ZTXM19, ZCZ+18, ZCZ+18].

Graph Convolutional Networks. Graph Convolutional Networks (GCNs) aggregate node information from its neighboring nodes by the convolution operation on graphs [KW16]. Based on the types of convolutions, they can be categorized as spectral approaches and spatial approaches. For spectral methods, the graph convolutions are defined in the fourier domain and can be computed by taking the inverse fourier transform of the multiplication between two fourier transformed graph signals. The limitation of spectral methods is that learned filters depend on the graph structure. Therefore, a model trained on some graph structure is not able to be directly applied to another different graph. Unlike spectral methods, spatial approaches directly define convolutions on the graph, aggregating node information from its spatial neighborhood nodes. Thus, these approaches do not depend on the graph structure.

Graph Attention Network. Graph Attention Network (GAT) is developed by Veličković et al. [VCC+17]. It incorporates the attention mechanism into the propagation steps. Veličković et al. compute the hidden state of each node by attending over its neighbors, based on a *self attention* strategy. The core layer of GAT is the graph attentional layer which allows every node to attend on its neighboring nodes. The main contribution of their work is that their networks allow a node to assign different weights to other nodes in its neighborhood, without requiring any kind of costly matrix operation (such as inversion) or depending on knowing the graph structure.

3

FREE-VIEWPOINT IMAGE BASED RENDERING

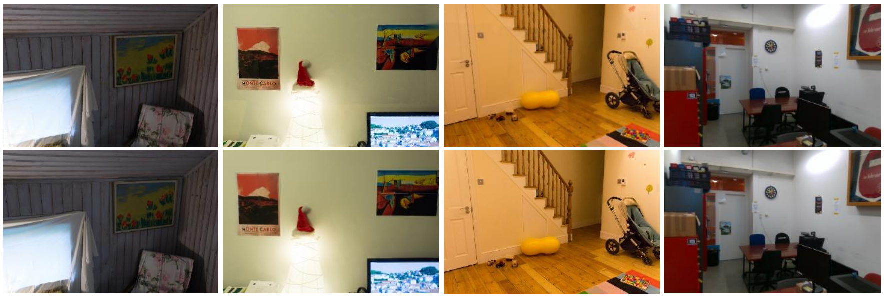


FIGURE 3.1: *Qualitative comparison between simulated images (first column) and ground-truth images (second column) on datasets of Attic, Study room, Playroom and Reading corner.*

In this chapter we present a novel depth image based rendering (DIBR) approach to produce photo-realistic imagery of real scenes. There are several challenges for DIBR: (1) misalignment of object boundaries between color-and-depth image pairs often leads to ghost contours; (2) projection errors and missing depth information result in the visibility failure; and (3) useless and redundant input views often produce blurry images.

To address these issues, we propose a pixel-to-pixel multi-view depth refinement method to produce pixel-accurate alignment between color-and-depth image pairs, and an adaptive view selection approach to avoid choosing redundant or useless input views. Furthermore, we propose a layered 3D warping to avoid the loss of visible information. These components are designed to work together, reducing visual artifacts in synthesized images, while hardly sacrificing rendering speed. The evaluation results indicate that our method significantly improves the quality of synthesized images, achieves good performance on a wide variety of challenging scenes (see Figure 3.1) and performs best among popular DIBR algorithms.

3.1 INTRODUCTION

There is increasing demand on reproducing photo-realistic virtual versions of real scenes for a large number of vision applications, such as free-viewpoint television (FTV), physical training and virtual navigation. One promising approach that provides such realistic and interactive imagery is DIBR [Feh04]. DIBR allows users to interactively control the viewpoint to produce synthesized images from arbitrary positions. However, various visual artifacts like ghost contours, holes often appear in the synthesized image.

Ghost contours are mainly caused by the misalignment of object boundaries between a color image and its corresponding depth map. Object edges in the color image always have transitional pixels, while there are only sharp edges in its corresponding depth map. After projection, the edge-transitional regions are split and appear various visual artifacts. There are plenty of studies to correct misalignment by erasing edge-transitional regions [MFY$^+$09, CDSHD13] or smoothing depth edges [LZW$^+$17]. Nevertheless, these methods are more likely to introduce new visible artifacts, for they may remove useful information when erasing the misalignment. On the contrary, we introduce a pixel-to-pixel multi-view depth refinement algorithm to take advantage of useful information in transitional regions, and produce better alignment between color-and-depth image pairs. In addition, our refinement method is able to fill missing depth information with the consideration of photometric and geometric consistency among multiple images.

Redundant and useless input views often lead to blurring or incomplete synthesized images. Previous studies select input images by comparing angles or distances between the input and target views [DN17, LLF$^+$16], and the number of input images used for blending is often fixed. Therefore, they may fail to choose enough input views or choose incorrect and redundant views. In order to avoid such cases, we propose an adaptive view selection method with variable well-chosen input views to improve the quality of synthesized images.

In the blending process, visibility is often solved by the Z-buffer method that only recovers the front-most pixels [ZDdW10]. However, the Z-buffer approach fails to solve the visibility problem caused by the incorrect depth information or projection errors. As a result, when these errors exist, objects in the foreground may be occluded by objects in the background in the synthesized image. To address this issue, we divide the depth map into layers and apply 3D warping to synthesize images on each layer with a switching median filter to avoid the loss of visible information and over-smoothing. Since layered depths have the ability to represent occluded elements, our approach is better in dealing with the visibility problem. Our main contributions are summarized as follows:

- A novel depth refinement algorithm that respects photo-consistency and edge preservation to correct misalignment between color-and-depth image pairs and fill missing depth information.

- A novel adaptive view selection approach that effectively avoids selecting redundant and useless input views to improve the quality of synthesized images and the rendering speed.

- A novel rendering algorithm providing high-quality free-viewpoint synthesized images, which is based on layered 3D warping to deal with occluded elements and lower the rendering complexity.

We have applied our algorithm on a variety of complex indoor scenes, demonstrating that our method provides plausible novel views and significantly improves the peak signal to noise ratio (PSNR) compared to previous works.

3.2 RELATED WORK

Image-based rendering has been an active research area over a long period and a thorough review of it can be found in [SCK08]. Here we only discuss the most related approaches in DIBR.

The importance of maintaining the alignment of object boundaries between color-and-depth image pairs has been known for many years [ZK07, EdDM+08]. Zitnick et al. [ZK07] use color oversegment to detect object boundaries and then use neighboring Markov Random Filed to reduce artifacts at these areas. Similarly, Chaurasia et al. [CDSHD13] and Ortiz-Cayon et al. [OCDD15] divide the image into superpixels to preserve depth discontinuities and then project each superpixel to the virtual view by a local shape-preserving warping to improve the blending quality. Hedman et al. [HRDB16] combine two multi-view stereo methods to produce depth maps which respect occlusion edges. Alternative solutions including soft visibility [EdDM+08] and alpha matting [ZKU+04], correct misalignment by building a visibility map to distinguish occlusion boundaries. However, these approaches do not consider geometric consistency among input images and still suffer from silhouette flattening and inaccurate occlusion edges.

A different class of approaches is to directly remove edges with high discontinuities in the depth image [MFY+09, ZDdW10]. Mori et al. [MFY+09] expand the border of edge-transitional regions and use a boundary matting defined by a hard threshold to remove the mixture combing foreground and background information. Zinger et al.

[ZDdW10] detect pixels that have high discontinuities and then label them. After that, only unlabeled pixels are wrapped to the virtual view. However, the eliminating process also erases useful information and can introduce other visual artifacts. In contrast, our depth refinement method takes the advantage of these useful pixels and is able to achieve better alignment between color-and-depth image pairs.

There have been a lot of works [MFY+09, CDSHD13, LZW+17] that improve the quality of synthesized images by filling holes. Schmeing et al. [SJ15] use the inpainting approach [BSCB00] to fill holes. Solh et al. [SA12] use a hierarchical pyramid-like method to detect pixels of holes from lower resolution estimates of the synthesized image. They then fill holes use background information. Similarly, Dai et al. [DN17] also use the hierarchy idea to explore the depth distribution of neighboring pixels around each hole. Based on the distribution, they choose a number of pixels from the background and use them for hole filling. However, when there is no background information available near a hole, these approaches have poor performance. Li et al. [LZS18] use multiple reference views to fill holes, which has some similarity with ours. Nevertheless, the number of input views used in their method is fixed, which may lead to hole filling failure when the chosen view are useless or redundant. Instead, we use an adaptive number of input views to avoid such cases.

To solve visibility for synthesized images, many blending methods [CCL+05, MFY+09, SSS09, GAF+10] often use the Z-buffer algorithm. Hedman et al. [HRDB16] use a fuzzy depth test based on Z-buffer to blend multiple images. Dai et al. [DN17] defines a threshold to blend images with the similar idea like Z-buffer. However, these methods are generally unable to remove background information wrongly appearing in the foreground, which is caused by projection errors or incorrect depths. Unlike these approaches, we propose a layered 3D warping approach to resolve visibility, which can effectively reduce the loss of foreground information and remove background information that wrongly appears in the foreground.

More recently, deep learning-based approaches have been applied to synthesize virtual views [FNPS16, XZH+18, HPP+18, MSOC+19a, ZTF+18]. Srinivasan et al. [SWS+17] train deep learning pipelines to predict the local geometry for blending, and Hedman et al. [HPP+18] use a Convolutional Neural Network (CNN) based architecture to estimate pixel weights for rendering. Furthermore, Flynn et al. [FNPS16] and Wu et al. [WZW+17b] directly use deep learning methods for end-to-end view synthesis. However, in the current state, these methods still suffer from high computational costs and blurring when the virtual view is substantially different from input views. Besides, they are not suitable for small datasets collected from a large range of viewpoints. In contrast, our view synthesis approach does not require a large amount of data, and can provide consistent rendering with a sparse collection of input images.

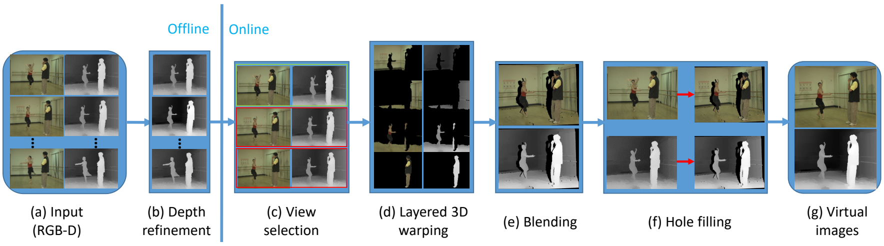


FIGURE 3.2: *Overview of our algorithm. (a) The input of our method are color-and-depth image pairs. (b) The initial depth maps are refined to achieve better alignment between object boundaries of color-and-depth image pairs. (c) Images with small view angles, short distances and large overlap are chosen as input images. (d) We divide the depth map into layers and perform 3D warping on each layer. (e) The synthesized images are blended together. (f) Holes in the synthesized image are filled with other input images, generating the final virtual images (g).*

3.3 OVERVIEW

Our goal is to achieve free-viewpoint rendering even in regions where a global 3D reconstruction of the scene has missing or inaccurate data for both weak and strong computing power devices.

High-quality DIBR depends on precise depth values and pixel-accurate alignment of object boundaries between color-and-depth image pairs. This is because inaccurate depth values and misalignment often lead to various visual artifacts, such as ghost contours. Unlike previous methods [CCL$^+$05, LLF$^+$16], which only aim to correct the misalignment of boundaries between color-and-depth image pairs, we aim to correct misalignment and fill missing depth information at the same time. Inspired by the idea of Patchmatch stereo [BRR11] that in natural stereo pairs relatively large regions of pixels can be modeled by approximately the same plane, we propose a pixel-to-pixel multi-view depth refinement method to refine depth maps. With the consideration of photo-consistency and edge preservation among multiple images, our approach is able to generate high-quality depth maps.

Even with high-quality depth maps, the synthesized image may still have holes caused by the lack of input images. However, increasing the number of input images is likely to introduce redundant or useless images. These additional images can sometimes be worse than a number of well-chosen images, as they may blur synthesized images. Besides, the more input images are chosen, the more computation time is required. To overcome these problems, we present an adaptive view selection algorithm which chooses input images based on angles, distances and overlaps between two views to avoid selecting useless and redundant images. In the rendering process we use a variable number of input images to synthesize the virtual image to lower the rendering complexity and improve the quality of synthesized images.

When blending input images, the Z-buffer method is often used to solve visibility. The intuition behind it is that closer objects occlude farther objects. However, it is not sufficient to achieve high quality for DIBR. To further improve the quality of synthesized images, we divide the depth map into layers, and then apply the 3D warping on each layer to produce the virtual image. Furthermore, we present a switching median filter to fill missing information in the layered synthesized image to avoid the loss of visible information and over-smoothing problem. After that, we blend these virtual images together to produce the final synthesized image.

Combining the novelties above, our pipeline works as follows: During offline processing, we employ a pixel-to-pixel multi-view depth refinement approach to improve the quality of initial depth maps by generating pixel-accurate alignment of object boundaries between color-and-depth image pairs and filling missing depth information (see Section 3.4.1). During online processing, to avoid blurring images, we select input images not only based on angles and distances but also the overlap between the input and virtual views in the query dataset (see Section 3.4.2). We then apply layered 3D warping that can better handle occluded elements to synthesize virtual images. Finally, our adaptive view selection approach is introduced to iteratively fill holes with the other input images (see Section 3.4.3). Figure 6.2 shows the pipeline of our work.

3.4 FREE-VIEWPOINT IMAGE BASED RENDERING

3.4.1 DEPTH REFINEMENT

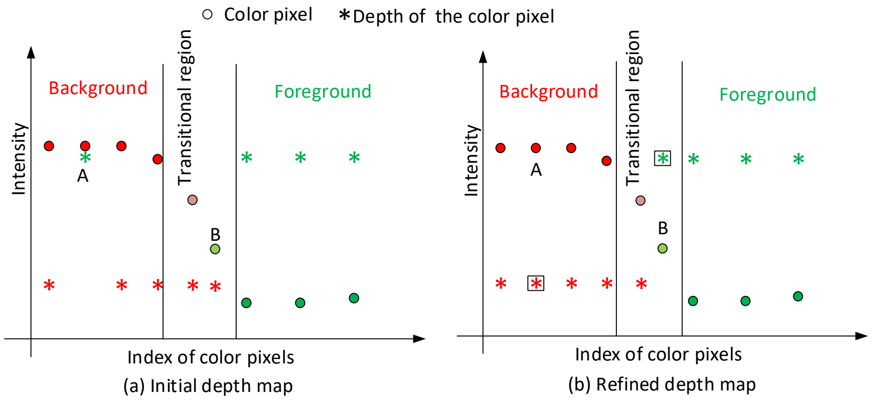
High-quality depth maps are necessary for consistent rendering. However, the depth map generated by 3D sensors often has inaccurate depth values and seldom aligns object boundaries with its corresponding color image, as illustrated in Figure 6.3(a). There, the background color pixel A is wrongly assigned with a foreground depth value, and in the transitional region the color pixel B which should be assigned a foreground depth value turns to have a background depth value. Figure 6.3(b) shows the refined depth map we aim to produce, where color pixel A and B are assigned with correct depth values. To achieve this goal, we propose a pixel-to-pixel multi-view depth refinement approach with the consideration of photometric and geometric consistency among pixels. Our matching cost function C is defined as,

$$C(i) = C_{pixel}(i) + C_{patch}(i), \tag{3.1}$$

where $C_{pixel}(i)$ and $C_{patch}(i)$ emphasize photo-consistency and edge preservation for the pixel i, respectively.

The photo-consistency $C_{pixel}(i)$ for the pixel i is measured by projecting it to other images, where we compare the color and gradient similarities.

$$C_{pixel}(i) = \lambda||x_i - x_r|| + (1-\lambda)||\bigtriangledown x_i - \bigtriangledown x_r||, \tag{3.2}$$



(a) Initial depth map (b) Refined depth map

FIGURE 3.3: *Problems in the depth map and refinement results. (a) The background color pixel A has incorrect depth information and the foreground color pixel B mismatches background information in the transitional region. (b) The incorrect depth value and misalignment are corrected after depth refinement.*

where x_i is the pixel we calculate cost for in the target image and x_r is the corresponding pixel of x_i in the reference image. Also, $||x_i - x_r||$ and $||\bigtriangledown x_i - \bigtriangledown x_r||$ indicate the color and gradient differences, respectively. λ is a measure parameter. We set $\lambda = 0.9$ in all the experiments. For a target image, we select ten reference color images based on distances and angles between the target and reference views. Next, we iteratively project pixels in the target image to the reference images and only save the cost value of the front-most pixel. In this way, we are able to avoid obtaining high cost values for correct depths.

The edge preserving term $C_{patch}(i)$ encourages the resulting depth map to have pixel-accurate alignment with its corresponding color image. It is defined from Patch-match stereo [BRR11].

$$C_{patch}(i) = (\sum_{q \in W_i} diff)/N, \tag{3.3}$$

where W_i denotes a patch centered on pixel i and q is the neighbor pixel of i. N is the size of the patch. The matching cost $diff$ consists of a weighted combination of $||x_i - x_q||$ and $||\bigtriangledown x_i - \bigtriangledown x_q||$:

$$diff = \lambda||x_i - x_q|| + (1 - \lambda)||\bigtriangledown x_i - \bigtriangledown x_q||. \tag{3.4}$$

In the depth refinement process, we first select pixels that need to be modified based on the cost value. If the cost value of pixel p is bigger than the average cost of a $(n \times n)$ patch centered on it, we search the patch to find the lowest cost (pixel q) in the patch.

This is because correct depth values have low matching costs that are computed with the consideration of photometric and geometric relationships among pixels. Next, we replace the depth and cost of p with q's, for spatial neighboring pixels are likely to have similar depth values. We run this process until all the pixels are compared. The comparison process is interleaved with the depth refinement. That is propagating good depth values to neighbors, if the costs are smaller than those of their neighbors. We set $n = 3$ in all the experiments. After propagation, we filter unusual depths with a weighted median filter [MHW+13] which is guided by the color image. The depth refinement algorithm is summarized in Algorithm 1.

Algorithm 1 Overview of the depth refinement procedure.

```
Input: Color images $I_1 ... I_N$, patch size $n \times n$ and depth maps $D_1 ... D_N$.
Output: Refined depth map $D_1$ for color image $I_1$.
Calculate photo-consistency cost $C_{pixel}(p)$ in $I_1$ and edge preserving cost $C_{patch}(p)$ in $I_1$.
Calculate matching cost $C(p) = C_{pixel}(p) + C_{patch}(p)$.
if (matching cost $C(p) >$ average cost of patch P ($n \times n$) centered on $p$ then
    for pixel $q_i \in$ patch $P$ do
        Find $q_i$ with the lowest matching cost $C(q_i)$ and replace the depth and matching cost of $p$ with $q_i$'s.
Run weighted median filter.
```

3.4.2 VIEW SELECTION

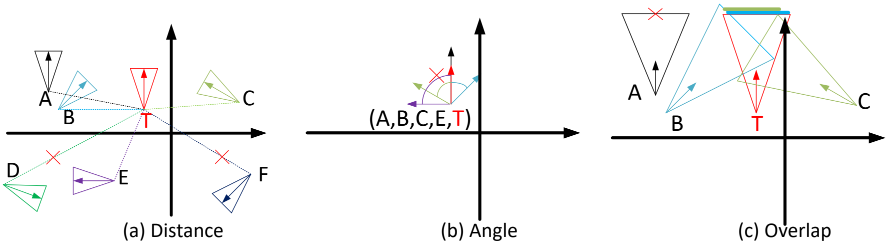
A large number of works (e.g., [MFY+09, CDSHD13, LZW+17]) improve the quality of synthesized images by correcting misalignment or filling holes. However, less attention has been paid to select input views, which is also important for improving the quality of synthesized images. Previous studies may choose incorrect or redundant views based on angles or distances between two views, which leads to blurring synthesized views. In order to avoid choosing such input views, we select views not only considering angles and distances but also overlaps between two views.

Figure 5.7 shows the selection process. Firstly, the distance between the input and the target views is calculated as shown in Figure 5.7(a), where A, B, C, D, E, F are input views and T is the target view. The distance between the input and target views is defined by:

$$distance(i) = \|O_t - O_i\|, \tag{3.5}$$

where O_t and O_i are the centers of target view t and input view i, respectively.

Then we rank the calculated distances and select the top ten images as a local group. From these local images, angles between the target and input views are calculated:



(a) Distance (b) Angle (c) Overlap

FIGURE 3.4: *View selection pipeline. (a) A, B, C, D, E, and F are input views and T is the target view. We first select a cluster of images which have short distances between the target and input views. (b) From the cluster of images, we select a subset of images with small angles they have with the target view. (c) Based on the overlap between the target and input views, we remove views having no overlaps with T.*

$$angle(i) = \arccos(\frac{\overrightarrow{n_t} \cdot \overrightarrow{n_i}}{\|\overrightarrow{n_t}\| \cdot \|\overrightarrow{n_i}\|}), \tag{3.6}$$

where $\overrightarrow{n_t}$ and $\overrightarrow{n_i}$ are the view directions of target view t and input view i, respectively.

If the angle is bigger than the field of view of the camera capturing input images, we get rid of it from the local input image group as shown in Figure 5.7(b).

Furthermore, in order to remove views, like A which has a small angle and distance, but no overlap with the target view, we calculate the overlaps between input and target views. If the overlap is zero, we remove it from the local image group (Figure 5.7(c)). To reduce the computation time of calculating overlaps, we downsample the input image with an equal sampling interval and only project sampled pixels into the target view. In this way, the computation time can be reduced depending on the sampling interval.

The target virtual image is synthesized by locally blending the input images. However, directly blending all images is time-consuming. So that, we use a variable number of input images to produce the virtual image. We first project an input image which is selected based on our view selection approach in the input image group to the virtual position, and then detect the holes in the virtual image. If the size of the largest hole is big, we then choose another image in the local input group to fill holes. We iteratively run this process until the largest hole has been sufficiently covered.

3.4.3 VIEW SYNTHESIS WITH LAYERED 3D WARPING

The whole pipeline of our layered 3D warping is described in Figure 3.5. The core part of DIBR methods is 3D warping. It projects pixels in the input image plane to world coordinates and then reprojects them to novel positions in another image plane using the corresponding depth information.

Figure 6.4 shows the projection process. Let P_1 be a pixel point in the image plane C_1. P_1 is projected into the world coordinate system at P. The relationship between

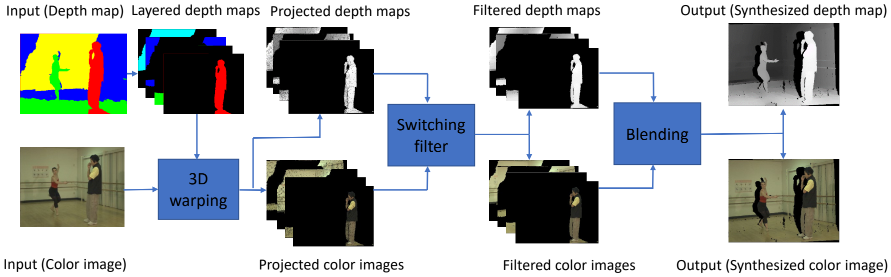


FIGURE 3.5: *Layered 3D warping. The input are color-and-depth images. Based on the maximum and minimum depth values, the depth image is divided into layers. On each layer, we apply 3D warping to synthesize the new image. A switching median filter is applied to fill missing information in these images. After that, all the filtered images are blended to produce the final color-and-depth image pairs.*

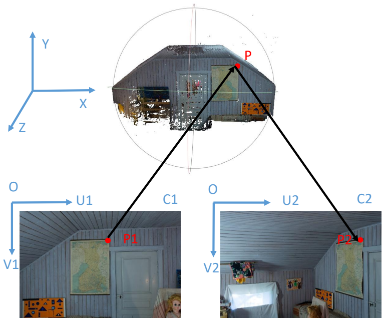


FIGURE 3.6: *3D warping. A point P_1 in the image plane C_1 is projected to a world point P and then P is projected to another image plane C_2 at position P_2.*

P and P_1 can be defined by left camera's intrinsic matrix K_1, rotation matrix R_1 and translation matrix T_1:

$$P = (K_1 * (R_1|T_1))^{-1} * z_1 * P_1, \tag{3.7}$$

where z_1 is the depth value of P_1. Furthermore, P is projected into the image plane C_2 at the pixel position P_2, which is calculated by

$$z_2 * P_2 = K_2 * (R_2|T_2) * P, \tag{3.8}$$

where z_2 represents the depth value of point P_2. K_2, R_2 and T_2 represent the intrinsic, rotation and translation matrices of the right camera.

However, the projection errors caused by 3D warping or incorrect depth information often lead to background information wrongly appear in the foreground, as shown in

Figure 6.5.

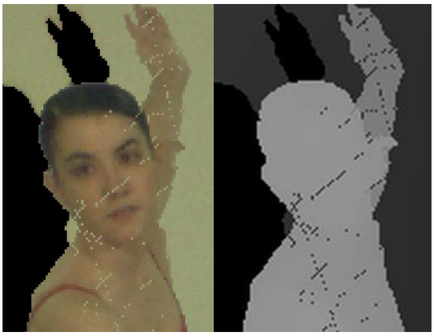


FIGURE 3.7: *Problems caused by 3D warping in the synthesized image.*

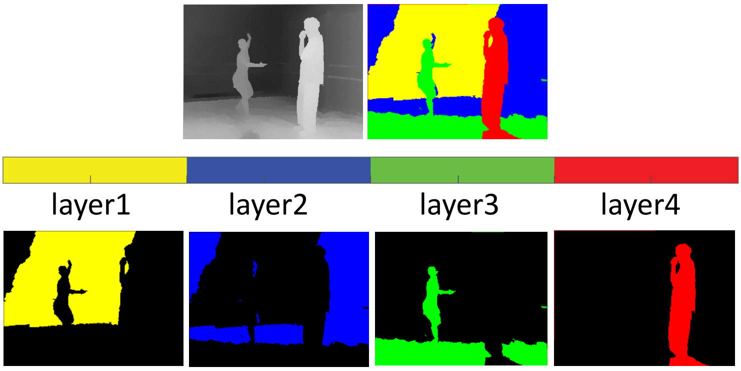


FIGURE 3.8: *The layered depth images. A depth image is evenly divided into layers based on the maximum and minimum depth values.*

To solve this problem, we evenly divide the depth image into layers based on the maximum and minimum depth values. Figure 6.6 shows an example of the layered depth maps. On each layer, we apply 3D warping with corresponding color-and-depth image pairs to produce new images and then introduce a switching median filter to remove unusual pixels in each new image.

A median filter [ACD$^+$09] is often used to filter unusual pixels in the projected image, for the distribution of these pixels has similar characteristics as salt-and-pepper noise. However, the traditional median filter is implemented uniformly across the whole image and tends to modify both noisy and good pixels at the same time. As a result, the filtered images are more likely to lose some details such as edges and small textures. Unlike the median filter, our switching median filter only refines pixels that have unusual information and can avoid smoothing over images. The switching median filter for pixel $P_{i,j}$ is defined as follows:

$$P_{i,j} = \begin{cases} median\left\{P'_{i+u,j+v} | (u,v) \in W\right\} & \text{if } P_{i,j} \in S \\ P_{i,j} & \text{otherwise} \end{cases}, \tag{3.9}$$

where $median$ is the traditional median filter and $P'_{i-u,j-v}$ is a pixel in the median kernel, $W = \{(u,v)| - (N+1)/2 \leq (u,v) \leq (N+1)/2\}$, N is the size of the median kernel and S is a cluster of chosen pixels. If $P_{i,j}$ is equal to zero and more than half of the pixels centered on $P_{i,j}$ are non-zeros, we consider $P_{i,j}$ belong to S.

After performing the switching median filter, we blend these new images together to produce the final synthesized images. We found four layers to be a good trade-off between quality and speed.

3.4.4 IMPERFECTIONS OF DIBR AND SOLUTIONS

In this section, we explain the imperfections of DIBR and summarize the solution for each of them. There are four basic problems in DIBR, that are ghost contours, cracks, occlusion, and holes.

Ghost contour. The ghost contour is mainly caused by the edge misalignment of object boundaries between a color image and its corresponding depth map. Object edges in the color image always contain transitional pixels, while edges in its related depth map do not have such transitional regions. After projection, the transitional areas of color images are split and appear various visual artifacts.

To overcome this problem, we refine the initial depth map by correcting the misalignment of boundaries between color-and-depth image pairs and filling missing depth information (see Section 3.4.1). In addition, when blending projected images, we detect big holes in the projected image and dilate these holes with several pixels, which is also useful to remove ghost contours.

Cracks. Due to the miss-focus and non-integer index problems, the input pixel is usually not projected to a point at an integer position. After resampling, there may be more than one value in a position, while there are no values in other positions, which results in crack artifacts in projected images.

The median filter is often used to remove cracks. However, traditional median filter often leads to the over-smoothing problem, which makes images lose small details. We introduce the switching median filter that only filters pixels among cracks to avoid above issues (see Section 3.4.3).

Occlusion. When objects in the background and foreground are projected to the same position, objects in the foreground may be occluded by objects in the background, which is caused by the incorrect depth information. Besides, objects that are supposed to show correctly can also be occluded due to projection errors. The Z-buffer approach is the most commonly used method to address these problems. However, it is not sufficient to generate high-quality synthesized images, for there are still many background pixels appearing in the foreground after applying this approach, especially for dynamic scenes.

To address this problem, we combine the layered 3D warping and switching median filter to synthesize new images (see Section 3.4.3). The layered depth map has the ability to represent geometry of occluded elements and the switching median filter can reduce the loss of visible information. The two components are designed to work together, giving high-quality performance.

Holes. Unobserved regions will result in holes in synthesized images. Moreover, the fixed number of input views used by traditional view synthesis methods is not guaranteed to cover the whole virtual view, which results in holes during rendering.

Unlike previous algorithms using a fixed number of input images, we use an adaptive number of images to fill holes in the synthesized image. Our adaptive view selection approach makes sure the given virtual view can be sufficiently covered, which avoids big holes in the synthesized image. At the same time, our approach (see Section 3.4.2) is able to limit the input views used for rendering. This helps to avoid blurry synthesized images and improve the rendering speed.

3.5 EXPERIMENTAL RESULTS

We test our approach on our own three datasets (Study room, Table1, Table2), four datasets from [HRDB16] (Attic, Dorm, Playroom, Reading corner), and two datasets from [ZKU$^+$04] (Ballet and Breakdancers). The breakdancers and ballet datasets are different from the other seven datasets, for they are dynamic scenes. Each of these two datasets contains a sequence of 100 color-and-depth image pairs, captured by eight static cameras that are positioned along an arc.

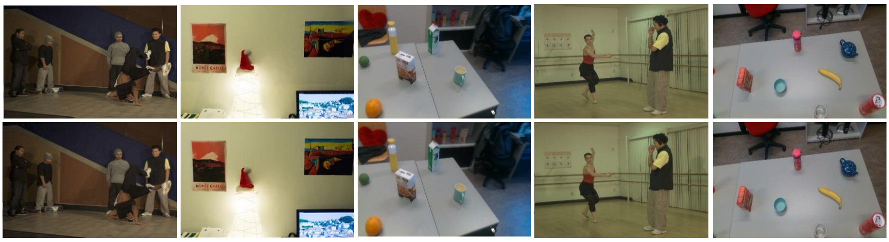


FIGURE 3.9: *Qualitative comparison between ground-truth images (second row) and simulated images (first row) on datasets of Breakdancers, Dorm, Table2, Ballet and Table1.*

Qualitative evaluation. We randomly choose an image from the dataset as our ground truth image and then use the other images in the dataset to synthesize the chosen image by our approach. The performance is shown in Figure 5.14, containing some synthesized image and their corresponding ground truth image. Figure 3.10 shows some additional synthesized images. From Figure 5.14 and Figure 3.10 we can see that our proposed method is able to provide high-quality synthesized images.

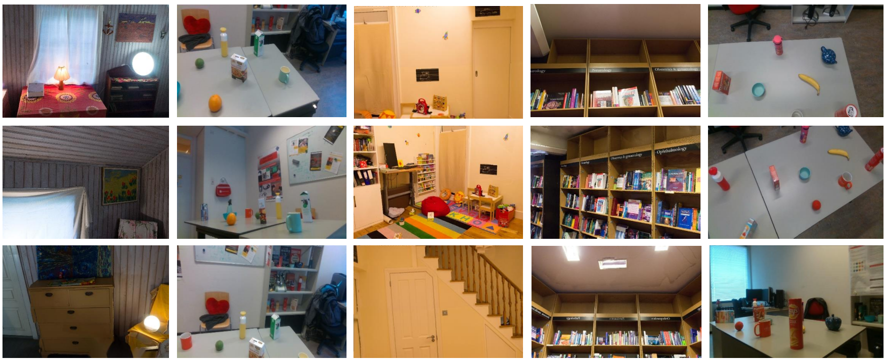


FIGURE 3.10: *Example results of synthesized images from different scenes (left to right): Attic, Table2, Playroom, Reading corner, Table1.*

Comparison with other methods. We compare our method with state-of-the-art learning-based algorithms designed for static scenes. The peak signal-to-noise ratio (PSNR) is used to evaluate image quality, where a higher PSNR value means a better image quality. Table 3.1 summarizes the quantitative evaluation results.

TABLE 3.1: *The PSNR comparison with different algorithms.*

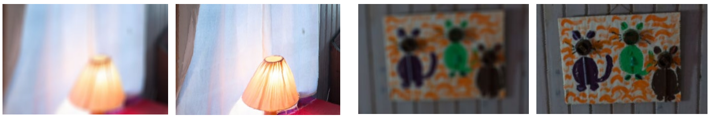
Methods	the average PSNR over 100 images (dB)	
	Table1	Study room
SM[ZTF+18]	22.15	10.53
LLFF[MSOC+19b]	24.17	13.22
NeRF[MST+20]	**37.97**	20.41
Ours	31.29	**26.60**

As we can see, even though the result of NeRF [MST+20] is better compared to other methods on Table1, our method achieves the best performance on Study room. This is because the Table1 is designed for pose estimation, where the camera poses are densely sampled, while Study room is a sparse image set. It indicates that our method is more robust to the dataset which is captured sparsely. Besides, our approach is free from training and provides plausible synthesized images.

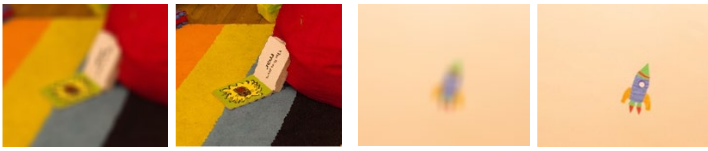
In Table 3.2, we also compare our method with state-of-the-art algorithms which are designed for dynamic datasets. We use two reference images to synthesize a new image on ballet and breakdancers datasets. We can see that our algorithm performs the best on both datasets.

TABLE 3.2: *The PSNR comparison with different algorithms.*

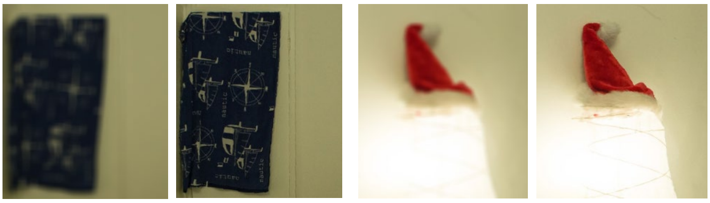
Methods	the average PSNR over 100 images (dB)	
	Ballet	Breakdancers
VSRS [Sof]	30.23	31.17
Liu [LLF+16]	32.52	33.33
Dai [DN17]	32.55	31.77
Loghman [LK15]	30.36	31.64
Ours	**33.40**	**33.59**



(a) *Attic*



(b) *Playroom*

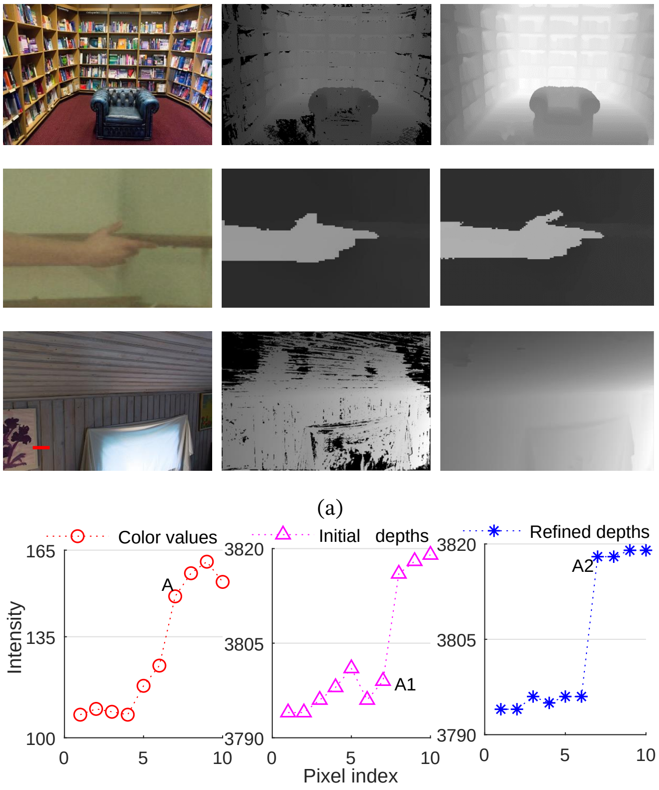


(c) *Dorm*

FIGURE 3.11: *Qualitative comparison between Local Light Field Fusion [MSOC+19b] (first and third column) and our approach (second and fourth column) on different datasets.*

In Figure 3.11 we compare our method to Local Light Field Fusion (LLFF) [MSOC+19b] which also uses layered depth maps to synthesize images. Since LLFF is designed for static datasets containing large overlaps, we only compare it on our static datasets with small changes. As we can see, LLFF suffers from the same limitation as other deep learning-based view synthesis methods, as images synthesized by LLFF are blurry. In contrast, our approach can provide sharp new images for various scenes.

Effect of depth refinement.

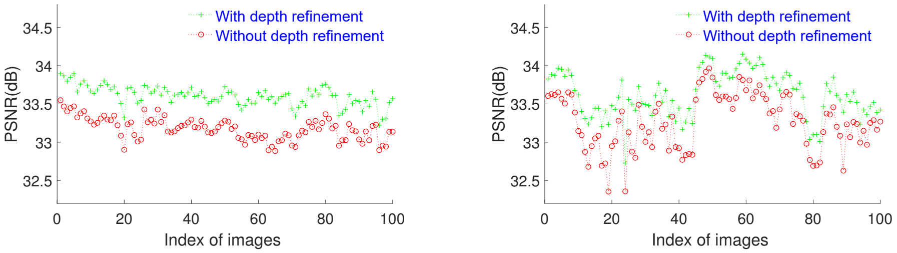


(a)

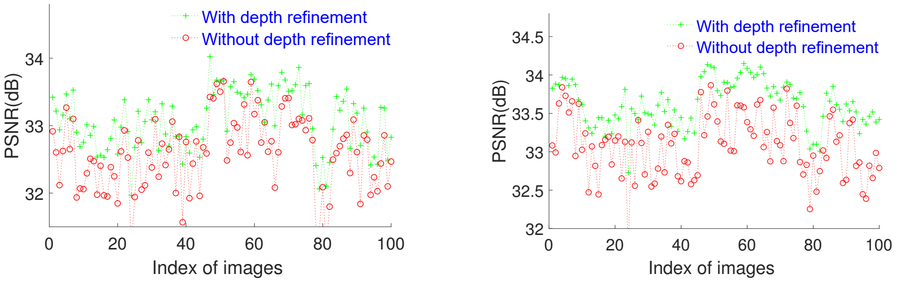
(b)

FIGURE 3.12: *Example results after depth refinement. (a): The visualization results of depth refinement on datasets (top to bottom): Reading corner, Ballet and Attic. (b): An example showing the misalignment between the foreground color A and background depth $A1$ is corrected by our depth refinement, where $A1$ is replaced with the correct foreground depth $A2$.*

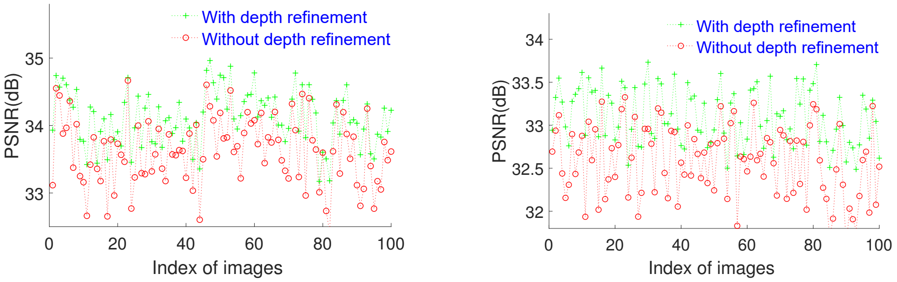
Figure 3.12(a) visualizes the depth refinement results on different datasets. We can see that our pixel-to-pixel multi-view depth refinement method is able to improve the quality of the depth map by filling missing depth information or refining incorrect depth values. Figure 3.12(b) shows the alignment process where a foreground color pixel A in the object boundary is wrongly assigned a background depth value $A1$, and after the depth refinement, this value is replaced with the correct foreground depth value $A2$ in the refined depth map. The color and depth intensities are obtained along the horizontal red line in the Attic dataset in (a).



(a) *Ballet* (b) *Breakdancers*



(c) *Attic* (d) *Dorm*



(e) *Table1* (f) *Playroom*

FIGURE 3.13: *PSNR comparison with and without depth refinement on each frame.*

Figure 3.13 shows the quantitative evaluation results with and without our depth refinement method on different datasets. As we can see, our proposed approach consistently improves the PSNR through all the testing frames. With the refined depth map, the growth of PSNR in the Ballet dataset is the largest. This is because the number of misalignment of boundaries between the color image and the depth map, and imprecise depth values in the Ballet dataset is more than those in the other datasets. After the depth refinement, these issues are solved, resulting in the increased PSNR.

Furthermore, we compare our depth refinement algorithm with the guided filter [HST12], a popular edge-preserving smoothing filter [HST12] and traditional median filter [ACD$^+$09]. We use the color image as the guided image and compare the average PSNR over 100 frames, as shown in Table 5.1. It can be seen that all the approaches are able to improve the quality of synthesized images, but our approach achieves better performance in various scenes.

TABLE 3.3: *The PSNR comparison among the guided filter, median filter and our depth refinement approach.*

	the average PSNR over 100 images (dB)		
	guided filter [HST12]	median filter [ACD$^+$09]	ours
Attic	29.39	17.30	33.28
Dorm	29.82	17.41	33.71
Ballet	28.85	16.23	33.43
Table1	30.37	17.98	34.31
Table2	30.61	17.99	33.21
Playroom	29.44	17.11	33.53
Study room	27.87	16.21	31.75
Breakdancers	30.29	17.72	33.89
Reading corner	28.53	16.54	31.78

Effect of view selection. Previous studies choosing input views by angles or distances are likely to select incorrect or redundant views, which results in blurring novel views with the low PSNR. In contrast, our selection algorithm tends to avoid choosing such views and is more likely to produce sharp synthesized images with the high PSNR. In Table 3.4, we compare the average PSNR over 100 images on a variety of datasets. It can be seen that our method selecting input views with the consideration of angles, distances and overlaps between the input and target views significantly improves the PSNR by a large margin, especially for the datasets of Dorm and Study room.

TABLE 3.4: *The PSNR comparison of synthesized images with input views selected with different strategies.*

	the average PSNR over 100 images (dB)		
	distances	angles	ours
Attic	25.51	24.12	33.28
Dorm	17.84	16.37	33.71
Ballet	26.34	25.89	33.49
Table1	24.90	23.34	34.31
Table2	20.87	18.94	33.21
Playroom	22.53	20.01	33.53
Study room	19.72	17.87	31.75
Breakdancers	28.25	27.19	34.67
Reading corner	20.11	17.86	31.78

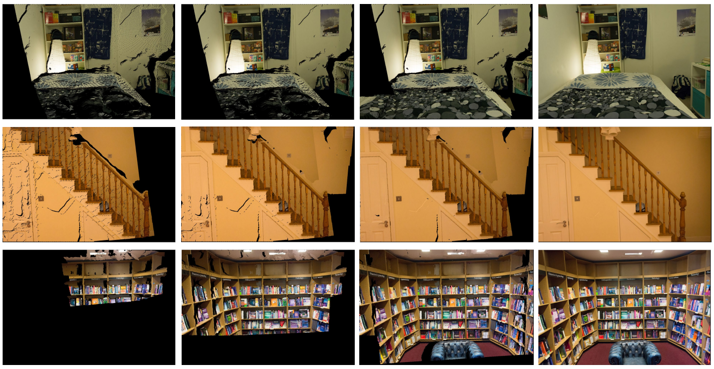


FIGURE 3.14: *Synthesized images by one view (first column), two views (second column), three views (third column) and adaptive views (fourth column) on different datasets.*

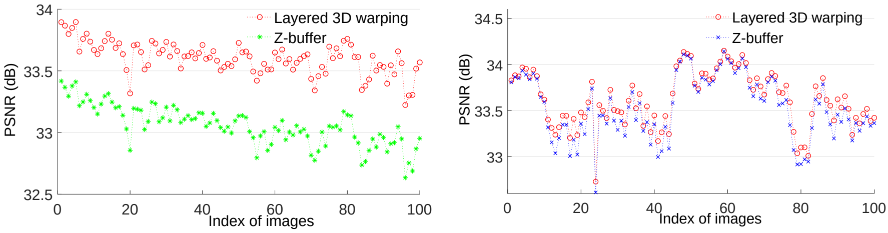
The quality of synthesized images is influenced by the quantity of well-chosen input images. We compare the hole sizes of synthesized image using different input views in Table 3.5 and Figure 3.14. For traditional methods, the number of input views is fixed, such as two or three, which does not guarantee to cover the whole virtual view. As a result, big holes often appear in the synthesized image. In contrast, our method with a variable number of input images is able to reduce the hole size significantly. For example, for the Study room dataset, the hole size is reduced by 20.13%, and 7.80% compared to methods with two and three input views, respectively. This is because the captured images in Study room dataset are very sparse. If the virtual view is substantially different from the input images, the synthesized image produced by the fixed number of

TABLE 3.5: *Hole size comparison of synthesized images using different input views. The hole size is defined by the percentage of missing pixels in the whole image.*

	Hole size(%)			
	1 view	2 views	3 views	ours (adaptive views)
Attic	50.07	10.31	2.26	0.01
Dorm	60.19	18.35	5.85	0.03
Ballet	50.16	3.15	1.93	0.02
Table1	31.13	2.51	1.33	0.01
Table2	35.19	3.24	2.67	0.02
Playroom	45.96	10.51	6.21	0.03
Study room	60.19	20.15	7.82	0.02
Breakdancers	47.89	2.15	1.21	0.01
Reading corner	23.12	10.49	1.18	0.02

input images may still have large holes.

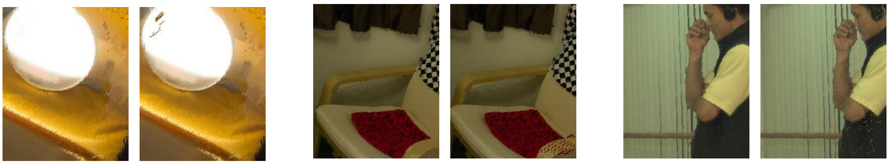
Effect of layered 3D warping. Figure 3.15 compares the PSNR with layered 3D warping and Z-buffer on two dynamic datasets which are more challenging than static ones. We can see that our layered 3D warping consistently improves the PSNR through all the testing frames. The effectiveness is also demonstrated by Figure 3.16, which shows some snapshots of synthesized images with our layered 3D warping. As can be seen, the background pixels are correctly removed and replaced by correct foreground pixels after layered 3D warping.



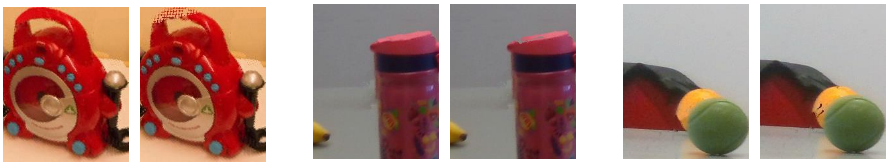
(a) *Ballet* (b) *Breakdancers*

FIGURE 3.15: *PSNR comparison with layered 3D warping and Z-buffer on each frame.*

We introduce the switching median filter to fill missing information in the synthesized image. Compared with the traditional median filter, the main advantage of our switching median filter is to avoid over-smoothing. To verify its effectiveness, we compare the average PSNR over 100 frames with the median filter, as shown in Table 3.6. It can be seen that our approach achieves better performance in different scenes.



(a) *Attic* (b) *Dorm* (c) *Ballet*



(d) *Playroom* (e) *Table1* (f) *Table2*

FIGURE 3.16: *Synthesized images with (first) and without (second) layered 3D warping on different datasets.*

TABLE 3.6: *Quantitative evaluation of the PSNR on different scenes.*

	the average PSNR over 100 images (dB)	
	median filter [ACD$^+$09]	ours (switching median filter)
Attic	31.57	33.28
Dorm	30.12	33.71
Ballet	31.15	33.43
Table1	30.63	34.31
Table2	31.11	33.21
Playroom	30.64	33.53
Study room	29.57	31.75
Breakdancers	31.91	33.89
Reading corner	29.67	31.78

Quality and time efficiency. The quality comparison of synthesized images with different depth layers is shown in Figure 3.17. We can see that more depth layers will improve the PSNR of synthesized images. However, more layers will also increase the computation time (see Figure 3.18) . We found four layers to be a good trade-off between quality and speed.

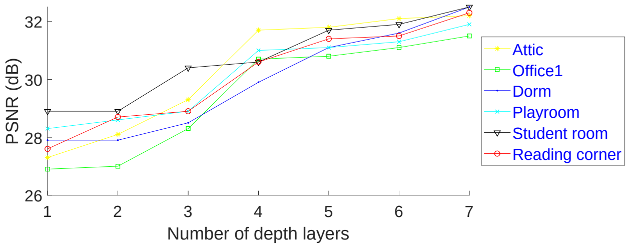


FIGURE 3.17: *Performance with different layered depth images.*

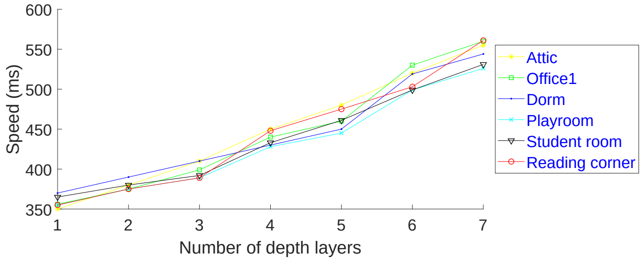


FIGURE 3.18: *The rendering speed of our method with different layered depth images.*

Effect of multi-layered depth maps. To verify the necessity of depth processing in the view synthesis framework, we calculate the average PSNR over 100 images on different datasets, as shown in Table 3.7. The traditional depth image based rendering, the method combing depth image based rendering and depth refinement, and the approach combining DIBR and layered 3D warping are referenced as DIBR, DIBR_DR and LDIBR respectively. The LDIBR_DR is the algorithm combing depth refinement and layered 3D warping. From Table 3.7, we can see that using DIBR_DR or LDIBR alone improves the performance as they are able to better process depth information and the combined method LDIBR_DR performs best.

TABLE 3.7: *Quantitative evaluation of the synthesized image in terms of PSNR with different approaches.*

	the average PSNR over 100 images (dB)			
	DIBR	DIBR_DR	LDIBR	LDIBR_DR
Attic	27.99	32.50	31.45	33.28
Dorm	26.89	33.12	31.62	33.71
Ballet	25.43	33.17	33.05	33.49
Table1	28.13	33.21	31.33	34.31
Table2	29.19	30.24	30.21	33.21
Playroom	27.96	29.52	30.21	33.53
Study room	28.12	30.22	29.45	31.75
Breakdancers	26.52	33.26	33.53	33.67
Reading corner	23.12	25.16	27.84	31.78

3.6 CONCLUSION AND FUTURE WORK

In this chapter we have proposed a novel view synthesis framework that first refines depth maps by correcting misalignment of object boundaries between color-and-depth image pairs and filling missing depth information. We then divide the depth map into layers and introduce a fast rendering algorithm combining an adaptive view selection approach and layered 3D warping to synthesize high-quality free-viewpoint images. The experimental results demonstrate that the quality of synthesized images is improved significantly with refined and layered depth maps. Since the rendering time of our proposed algorithm only depends on the display resolution of synthesized images, it can be used in low computational power devices such as mobile phones and virtual reality head-mounted displays as well as other systems requiring rendering. However, some limitations are worth noting. When the depth map produced by the depth camera has too much missing information, the synthesized image generated by our method shows various artifacts. Therefore, new methods are required to generate high-quality depth images for scenes with texture-less or reflective objects.

4

PRESIM: A 3D PHOTO-REALISTIC ENVIRONMENT SIMULATOR

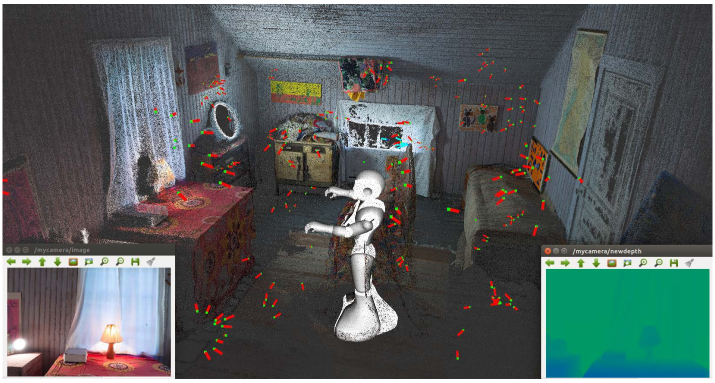


FIGURE 4.1: *Snapshots from PreSim showing a robot moving in an indoor environment: the left subwindow is the synthesized color image and the right subwindow is the depth image.*

In the previous chapter, we proposed a novel view synthesis algorithm to provide photo-realistic imagery of real scenes. In this chapter we use this approach to design a 3D environment simulator for synthesizing free-viewpoint RGB-D views, as shown in Figure 4.1. Recent years have witnessed great advancement in visual artificial intelligence (AI) research based on deep learning. To take advantage of deep learning, we need to collect a large amount of data in various environments and conditions. However, collecting such data is a time-consuming and labor-intensive task. Apart from that, developing and testing visual AI algorithms for multisensory models (e.g., mobile robots) are expensive and in some cases dangerous processes in the real world. We present PreSim, a 3D environment simulator which provides photo-realistic images using a view synthesis module and supports flexible configuration of multimodal sensors to address both of these issues. We demonstrate that PreSim has several advantages: (i) it provides a photo-realistic 3D environment which allows seamlessly integrating multisensory models in the virtual world and enables them to perceive and navigate scenes, (ii) it has an internal view synthesis module which allows transforming algorithms developed and tested in simulation to physical platforms without domain adaption, (iii) it can generate an infinite amount of data for vision-based applications, such as depth estimation, object recognition and object pose estimation.

4.1 INTRODUCTION

Recent years have witnessed great success of data-driven methods that use deep networks for computer vision tasks, such as depth estimation [LRB+16] and 6D object pose estimation [WXZ+19]. These data-driven methods need a large amount of data to train and test their models. However, collecting and labeling data are time-consuming and tedious. Gradually, the simulated environment is becoming an effective way to solve these problems, for it is able to provide an infinite amount of annotated data for various AI tasks. A major current focus of environment simulators is to reproduce high-quality free-viewpoint rendering of real senses. There are a number of open source simulators [UM20, YMB+18] to achieve this goal by parameter settings of scene details, including geometry, texture, lighting and 3D modeling of static objects. However, parameter setting is a time-consuming and labor-intensive process. Even with precise modeling and suitable parameter settings, the simulated world still lacks richness and diversity of the real world. This disadvantage may result in the failure of transferring algorithms that are developed and tested in simulation to physical platforms for many vision-based tasks, such as object recognition, obstacle avoidance, and visual navigation. This problem is known as the reality gap: the discrepancy between synthetic and real data.

To address this issue, game engines such as Unreal Engine which allow photo-realistic rendering are introduced to build virtual environments. However, the simulated environment heavily depends on the game engine's detailed datasets, which makes it impossible for users to build their own environments with their own datasets. On the other hand, game engines often use 3D graphics pipelines to provide real-time rendering. Thus, the rendering time increases linearly with the number of polygons to be rendered (scene complexity). To achieve real-time performance, it requires dedicated hardware and architecture design for 3D graphics. On the contrary, image based rendering which can provide real-time realistic imagery does not have these limitations. It only requires a sparse collection of captured images and allows a 3D scene to be visualized realistically without full 3D reconstruction. This approach has shown high-quality results in outdoor [CDSHD13] and indoor environments [HRDB16]. In addition, the run time of image based rendering mainly depends on the display resolution of the output image rather than scene complexity. Therefore, it can be used for both strong and weak processing power devices.

Taking advantage of image based rendering approaches, we introduce PreSim which is a 3D photo-realistic environment simulator for training and testing vision-based algorithms. We aim to narrow the reality gap between simulation and reality by providing huge amounts of photo-realistic virtual RGB-D views from arbitrary locations for vision-based applications. The main contributions of our simulator are:

- A photo-realistic 3D virtual environment that provides users with ground truth poses of the multisensory model and free-viewpoint color-and-depth image pairs, even in regions where a global 3D reconstruction of the scene has inaccurate or missing data.

- A global visualizer providing real-time positions and whole trajectories of the moving robot, and a global 3D map.

- A sequence controller and recorder components to control the movement of sensors and store all the required information for developing AI algorithms.

- A novel view synthesis module built on image based rendering that combines depth refinement, adaptive view selection and layered 3D warping to lower the rendering complexity and improve the quality of synthesized images.

Since PreSim is designed in a modular fashion, it allows scene augmentation and easy expansion to meet the user's requirements. We hope our simulator will further enrich and boost the research in robotic vision applications.

4.2 RELATED WORK

Here we discuss several notable works in environment simulators that are closely related to our work. For view synthesis approaches, a thorough review of them can be found in Chapter 3.

There are many environment simulators, such as Gazebo [KH04], CHALET [YMB⁺18], RotorS [FBAS16], and Atari [BNVB13], to model and visualize physical environments. Gazebo [KH04] is a well-known simulator that uses high-performance physics engines for rendering of indoor and outdoor environments. While Gazebo has rich features, it has difficulty to create visually rich environment of large scale and offer the realistic imagery. CHALET [YMB⁺18] is a 3D house simulator implemented by a professional game engine called Unity3D. It allows creating new virtual indoor environments and supports a range of common household activities. ViZDoom [KWR⁺16] is a semi-realistic 3D

TABLE 4.1: *A comparison of PreSim to other environment simulators. 3D: 3D nature of the rendered scene, Photo-realistic: photo-realistic rendering, Customizable: flexibility to be customized to other applications and Extendable datasets: permission for comstomer datasets.*

Simulator	3D	Photo-realistic	Customizable	Extendable datasets
Gazebo [KH04]	√		√	√
Atari [BNVB13]				
Malmo [JHHB16]	√		√	
VRKitchen [GGS+19]	√	√	√	
AI2-THOR [KMH+17]	√	√	√	
MINOS [SCD+17]	√		√	
House3D [WWGT18]	√		√	
Gibson Env [XZH+18]	√	√		
Habitat [SKM+19]	√	√	√	
PreSim (ours)	√	√	√	√

world simulator. It is based on the first-person shooter video game, Doom and allows developing bots that play Doom using the screen buffer. HoME [BPA+17] provides 45,000 diverse 3D house layouts. It uses Panda3D [GM04], an open-source 3D game engine to render indoor scenes based on object textures (wooden, transparent, metal, etc.), light and shadows. However, the drawback of the above simulators is the same as Gazebo. They are not capable of photo-realistic rendering.

A different class of approaches based on photo-realistic engines allows rendering of realistic camera streams [MB19, SDLK18, VS18, QZZ+17, DHH+20]. Among these, both AirSim [SDLK18] and CARLA [DRC+17] are autonomous vehicle simulators built on Unreal Engine 4 (UE4) and are able to provide physically and visually realistic simulations. VRKitchen [SDLK18] is an interactive 3D Virtual environment also built on UE4. It provides users with a variety of virtual kitchen environments. Habitat [SKM+19] uses Magnum engine to build photo-realistic virtual environments and provides a modular library for developing AI tasks (e.g., visual navigation) in it. However, these simulators are limited by richness of simulated environments due to their high dependency on the engines. In contrast, our environment simulator enables users to build their own environments with their datasets.

More recently, public datasets such as SUNCG [SYZ+17] and Matterport3D [CDF+17] have been used to create virtual environments. MINOS [SCD+17] is proposed to set up a

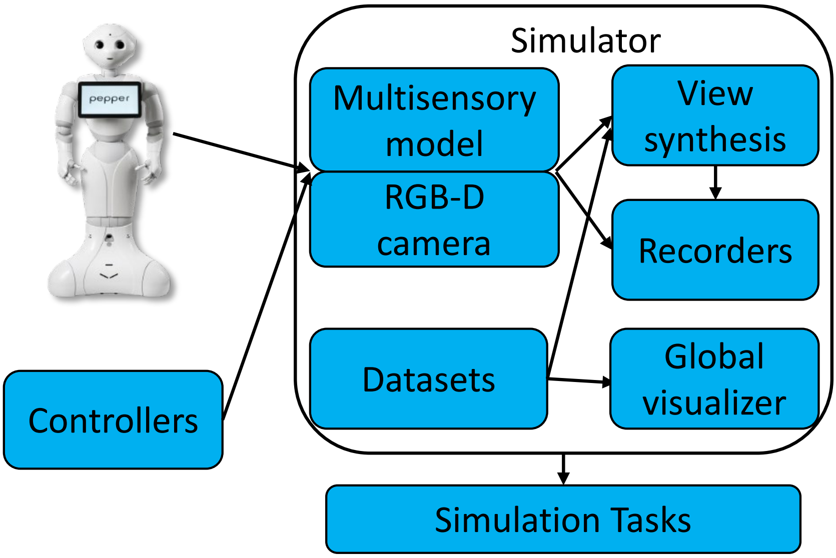


FIGURE 4.2: *The architecture of our simulator. It shows the main components of our simulator including a robot model, sensors, controllers, datasets, a view synthesis module and a global visualizer.*

benchmark for indoor navigation algorithms. It provides training, validation and testing datasets generated from both SUNCG and Matterport3D [CDF+17] datasets. Since these datasets consist of real-world scenes, MINOS allows realistic rendering. It also provides a flexible user API which allows removing and adding objects to configure the indoor environment. Another simulator that is based on SUNCG is House3D [WWGT18]. It contains a large number of house layouts with various objects and allows freely exploring the space. Gibson Env [XZH+18] is also based on real captured scenes. It has an internal view synthesis module allowing deploying the trained models in the real world without domain adaptation. While the goal of Gibson Env and our work is similar, Gibson Env requires a large amount of data to train a view synthesis network to avoid visual artifacts in synthesized images. A detailed comparison between our system and other environment simulators is summarized in Table 4.1.

4.3 PHOTO-REALISTIC VIRTUAL ENVIRONMENT

4.3.1 SYSTEM OVERVIEW

The architecture of our simulator is shown in Figure 4.2. It is composed of a robot model, sensors, controllers, scene datasets, a view synthesis module and a global visualizer. Our

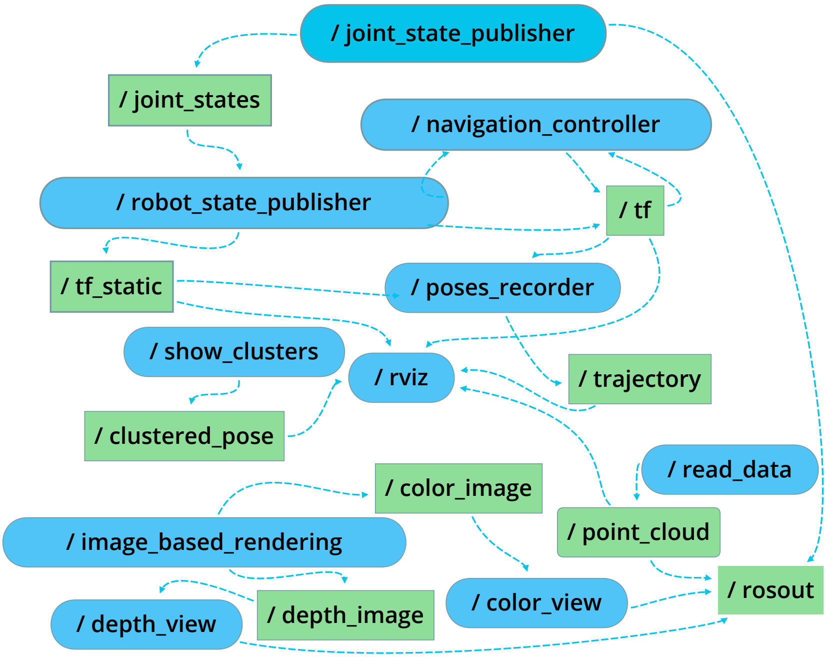


FIGURE 4.3: *The ROS node graph of our simulator.*

simulator is based on Robot Operating System (ROS) which has a modular design and can be customized, upgraded and reused. Figure 4.3 shows its ROS graph structure. In the virtual environment, we first import the point cloud of the real scene, which is generated from 3D reconstruction into the ROS and show it together with camera poses of input images in Rviz, a 3D visualizer for the ROS framework. Then we control the virtual camera's movement throughout the virtual world and estimate its 6D pose, including rotation and translation matrices. The estimated pose is then taken as a reference to select the most similar color-and depth image pairs in a query input dataset. Next, we use the selected color-and-depth image pairs to synthesize the virtual view based on our view synthesis module. At the same time, the whole trajectory of the moving camera, synthesized color-and-depth image pairs are logged. In the following, we provide more details on the individual components of our simulator.

4.3.2 VIEW SYNTHESIS

Our goal is to build a free-viewpoint photo-realistic environment for vision-based tasks. Unlike previous methods that build the whole virtual environment on the perfect reconstructed 3D geometry, our view synthesis module takes a sparse set of RGB-D images as the input and produces new color-and-depth image pairs from arbitrary viewpoints. An overview of our view synthesis pipeline is shown in Figure 4.4.

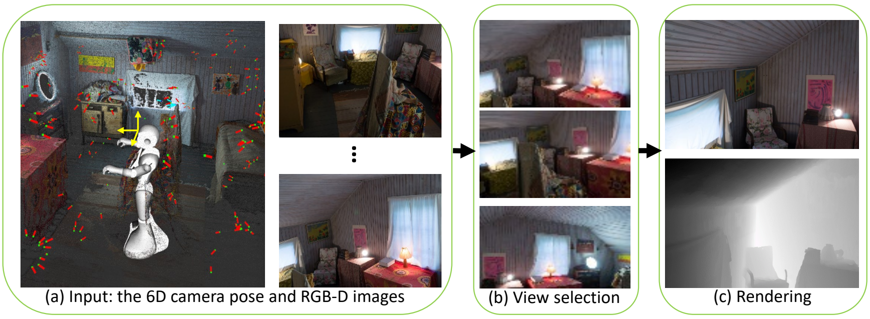


FIGURE 4.4: *The view synthesis pipeline including input, view selection and rendering. The input are RGB-D images and the 6D pose (red line shows positions of all the input image and yellow lines shows the 6D pose of the camera).*

It consists of view selection step followed by a fast rendering process. The input are RGB-D images and the 6D camera pose obtained from tf package. The view selection step is based on the 6D pose to find the most similar input views with the virtual view. The selection step is able to avoid selecting redundant or useless input views. The fast rendering algorithm provides high-quality free-viewpoint synthesized images by layered 3D warping. The layered 3D warping synthesizes images in different depth layers to lower the rendering complexity and improve the quality of synthesized image. A more detailed description about the view synthesis method has been presented in Chapter 3.

4.3.3 SCENE DATASETS

PreSim provides seven datasets that cover different practical scenarios. The Study room, Table1 and Table2 datasets are collected by us. The color-and-depth image pairs in these three datasets have a resolution of 1280×720 and the depth images are stored in millimeters as 16-bit PNGs. The Attic, Playroom, Dorm and Reading corner datasets are from [HRDB16]. The color-and-depth image pairs in these four dataset have a resolution of 1024×768 and the depth images are also stored in millimeters as 16-bit PNGs.

Each of these datasets has a sparse set of RGB-D images with corresponding camera poses and a 3D point cloud. The camera poses and 3D point clouds are produced by COLMAP, a 3D reconstruction software [SF16b, SZFP16]. Apart from that, users are also able to integrate their own datasets with PreSim. In the following, we describe the characteristics of these datasets.

Table1. Table1 dataset contains 233 color-and depth image pairs. There are many daily used objects such as reflective bottles and boxes in this dataset as shown in Figure 4.5. This dataset can be used for robotic grasping and object recognition tasks. This is

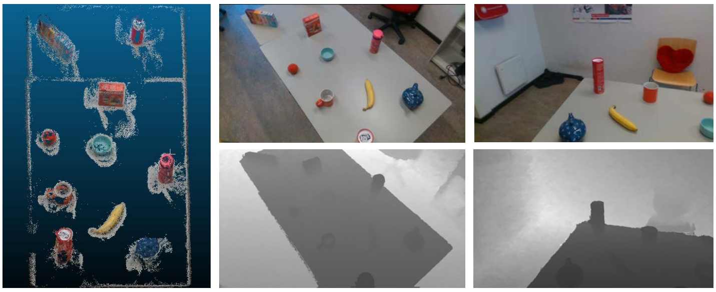


FIGURE 4.5: *The point cloud of the table1 dataset and examples of color-and-depth image pairs.*

because the key challenge of robotic grasping is how to estimate accurate 6D poses for objects with different sizes, shapes and textures, particularly for reflective objects.

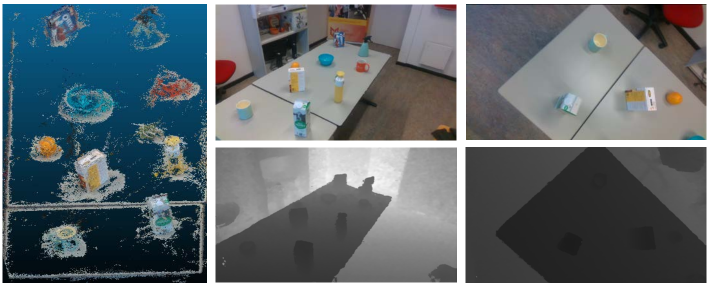


FIGURE 4.6: *The point cloud of the table2 dataset and examples of color-and-depth image pairs.*

Table2. Figure4.6 shows the Table2 dataset which contains 201 color-and depth image pairs. This dataset is also designed for robotic grasping and object recognition tasks. It is a complementary dataset to Table1. In this dataset, apart from daily used objects, it has texture-less objects such as the milk box and cups which are also challenging for 6D object pose estimation.

Study room. This dataset has 225 color-and depth image pairs. There are some black and texture-less objects (e.g., the white walls and writing board) which are challenges to find features and estimate depth information, as shown in Figure 4.7. This dataset is suitable for many vision-based tasks such as depth estimation and visual navigation.

Attic. The attic dataset contains 224 RGB-D images. This dataset can be used for robotic vision tasks such as object recognition and 6D object pose estimation. The conspicuous object in this dataset is a doll sitting on a chair in the middle of the room, which shows clear occlusion (see Figure 4.8). Apart from this doll, there are mirrors and lights

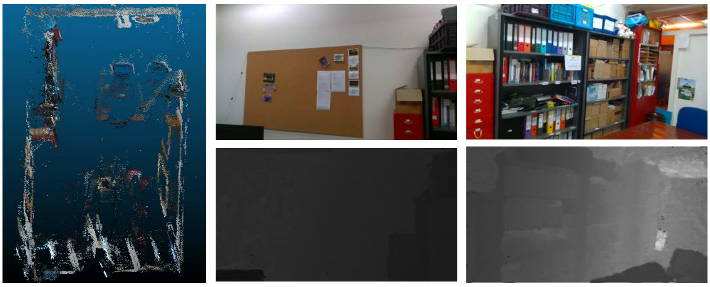


FIGURE 4.7: *The point cloud of the study room dataset and examples of color-and-depth image pairs.*

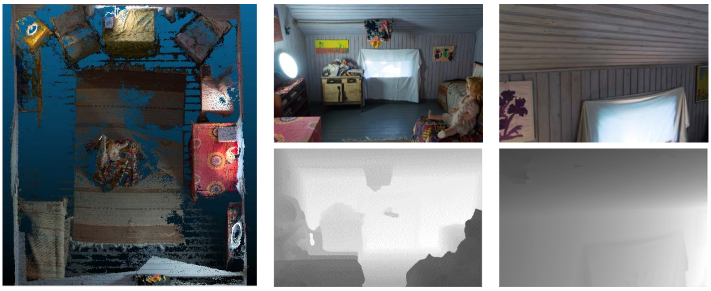


FIGURE 4.8: *The point cloud of the attic dataset and examples of color-and-depth image pairs.*

in this dataset, which are also challenges for object recognition and 6D object pose estimation.

Playroom. Figure 4.9 shows the Playroom dataset. It includes 219 color-and depth image pairs. There are some toys, chairs and desks in this room. These things contain many small geometric details that are difficult to estimate the depth information. Thus, this dataset can be used for depth estimation and object recognition. The size of this room is medium-sized. Thus, it is also suitable for visual navigation.

Dorm. There are 202 RGB-D images in the Dorm dataset. As shown in Figure 4.10, there are some texture-less objects, such as walls, lights and computer monitors in the room, which are challenges for depth cameras. This dataset is able to be used for depth estimation, and visual navigation.

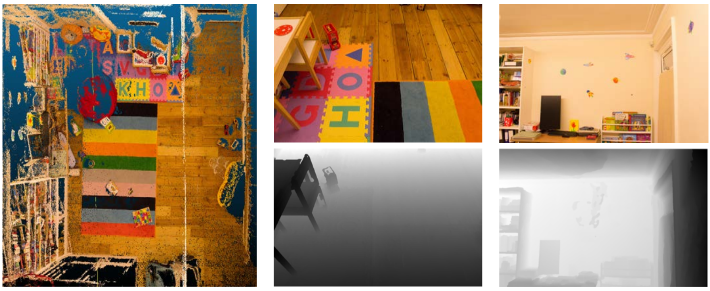


FIGURE 4.9: *The point cloud of the playroom dataset and examples of color-and-depth image pairs.*

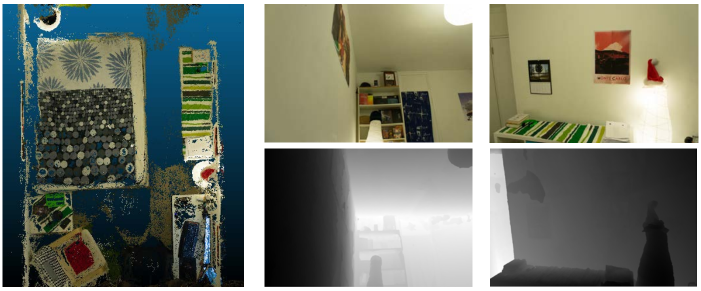


FIGURE 4.10: *The point cloud of the dorm dataset and examples of color-and-depth image pairs.*

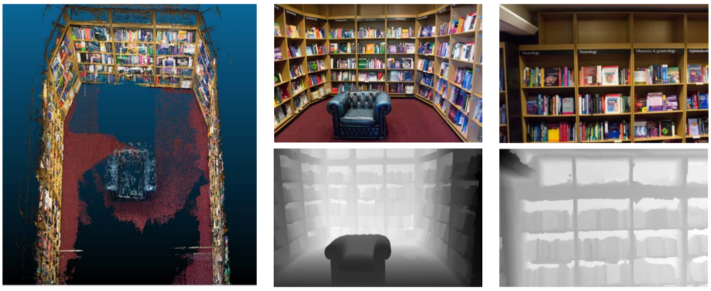


FIGURE 4.11: *The point cloud of the reading corner dataset and examples of color-and-depth image pairs.*

Reading corner. This dataset contains 167 color-and depth image pairs. It can be used for vision-based tasks such as object recognition and visual navigation. There is a large leather chair in the corner with strong view-dependent effects as shown in Figure

4.11. The scene also contains difficult occlusion characteristics including books and bookshelves.

Figure 4.12 summarizes the vision-based tasks which these datasets can be used for.

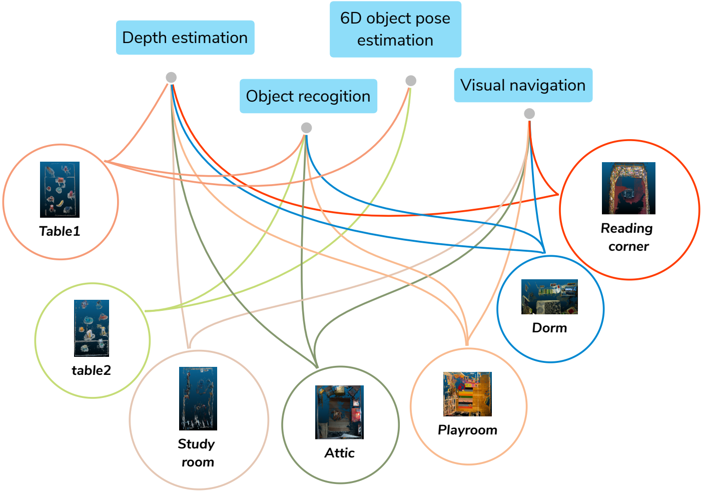
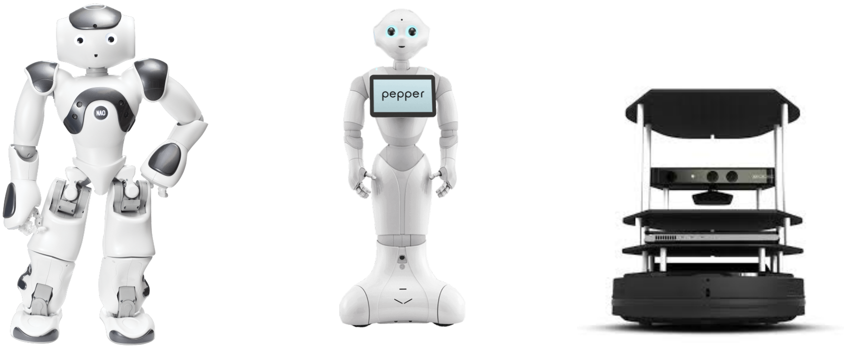


FIGURE 4.12: *The overview of eight datasets including Table1, Table2, Study room, Attic, Playroom, Dorm and Reading corner.*

4.3.4 ROBOTS AND CONTROLLERS



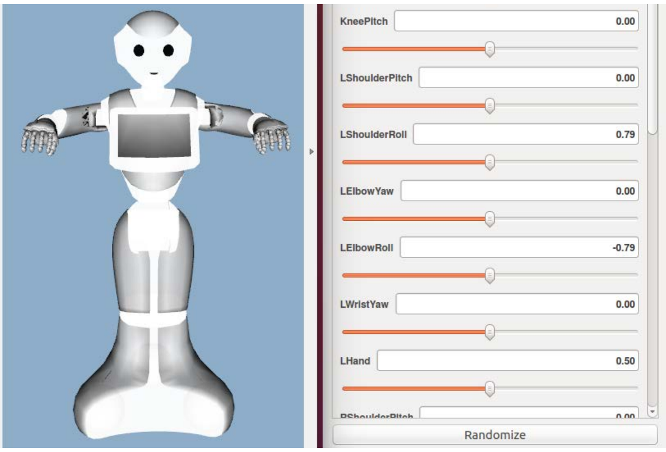
(a) *Nao* (b) *Pepper* (c) *Turtlebot*

FIGURE 4.13: *Examples of robot models in PreSim.*

PreSim is designed to study the problem of domain transfer from simulation to the real world. Therefore, it is important for the robot to be constantly subject to constraints of space and physics such as collision and gravity, throughout learning.

Robot models. Our simulator is designed for arbitrary robot (e.g., humanoid robots) using Universal Robotic Description Formats (URDFs). Thus, the robot model and its properties can be configured, such as the type of sensors and the frequency of data transmission. Figure 4.13 shows a series of available robot models. As a demonstrator, we use the Pepper robot, which is a social humanoid robot from SoftBank. The URDF including all elements (sensors, joints, links) and meshes are used to describe Pepper.

Integrated controllers. In our virtual world, we provide a set of practical controllers to reduce the controlling complexity for the robot's dynamic motions. The joint state controller (see Figure 4.14) is used to control the behaviors of joints of the robot, including changing the pitch, roll and yaw angles and positions. This control process is achieved by publishing and subscribing ROS messages under ROS.



KneePitch 0.00
LShoulderPitch 0.00
LShoulderRoll 0.79
LElbowYaw 0.00
LElbowRoll -0.79
LWristYaw 0.00
LHand 0.50

FIGURE 4.14: *The controller interface shows various parameters used to configure the movement of joints of the Pepper robot.*

The low-level navigation controller allows controlling the navigation of the robot by directly sending movement commands to the base of the robot. Besides, the tf package is used to get the real-time position of the robot, which is also used in rendering process. Figure 4.15 shows the ROS graph of the low-level navigation. We use the following commands to achieve controlling tasks:

(1) start roscore,

(2) bring up the simulated environment,

(3) bring up the robot,

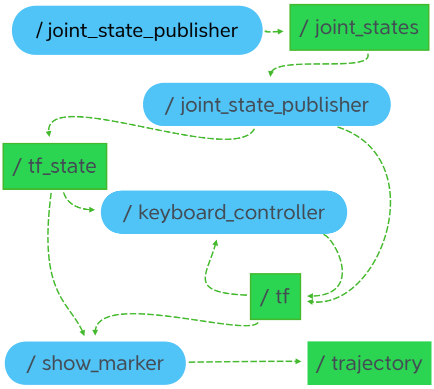


FIGURE 4.15: *The low-level navigation graph of the mobile robot.*

(4) rosrun joint state controller and,

(5) rosrun navigation controller.

Furthermore, we also provide data recorders for users to save all the required information including the robot's trajectory, camera's poses and synthesized color-and-depth image pairs. Figure 4.16 shows an example of a robot model and its trajectory.

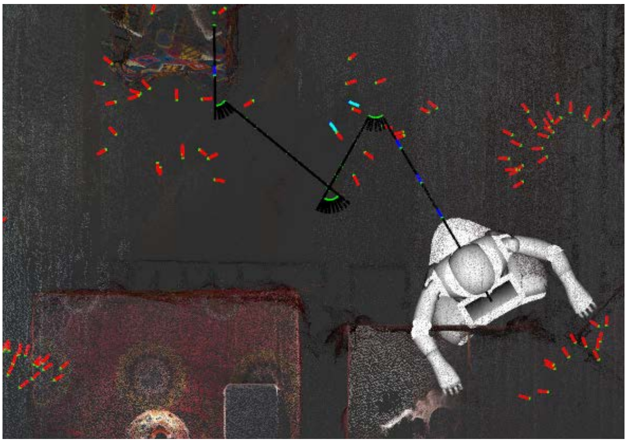


FIGURE 4.16: *The demonstrator model and its trajectory. Green points and red lines are positions and view directions of input views, light blue points and small blue lines are real-time positions and view directions of the virtual camera and the long line is the whole trajectory of the virtual camera.*

4.4 TASKS

Our virtual environment simulator provides various scenarios which can be used for different vision-based tasks. Apart from that, our simulator can be quickly set up to generate datasets with a large amount of photo-realistic color-and-depth image pairs with ground truth 6D poses (see Figure 4.17).

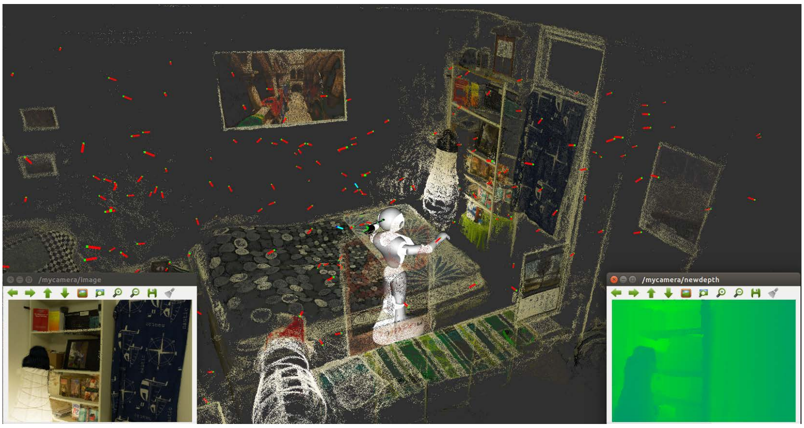
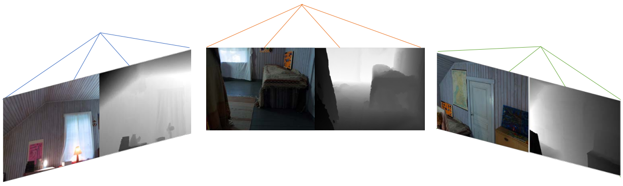


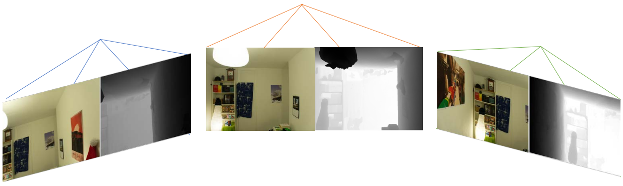
FIGURE 4.17: *Our environment simulator showing a robot moving in an indoor environment to synthesize RGB-D views: the left subwindow is the synthesized color image and the right subwindow is the depth image.*

Depth estimation. Recently, deep learning methods have been used to predict depth images for their corresponding color images. However, current datasets based on 3D sensors have key limitations, including indoor-only images (NYU) [SHKF12], small numbers of training examples (Make3D) [SSN08], and sparse sampling (KITTI) [GLSU13]. Besides, the transparent, irregular and reflective objects are hard captured by 3D sensors. Compared with datasets produced by collecting a large number of images, synthesizing such a dataset requires less hardware, time and human labor while resulting in better quality. Taking advantage of PreSim, we use it to generate depth datasets which can be used as training data for learning-based depth estimation algorithms. Examples of the generated images used for depth estimation is shown in Figure 4.18(a) and (b).

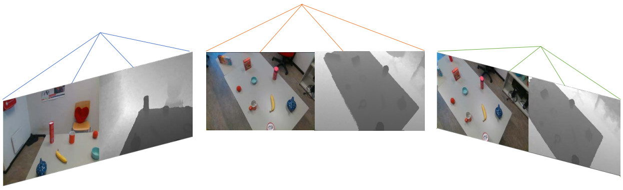
Object recognition. Finding objects in the scene is important for many robotic vision tasks. Methods for object recognition are divided into feature-based approaches [DT05, WOC$^+$07] and deep learning-based approaches [Gir15, LAE$^+$16, RDGF16]. For feature-based methods, scale-invariant feature transform (SIFT) and histogram of oriented gradients (HOG) features are the commonly used features. After obtaining these features, algorithms such as support vector machine (SVM) are used to do the classification. Unlike feature-based approaches depending on particularly defined features, deep



(a) *Attic*



(b) *Dorm*



(c) *Table1*

FIGURE 4.18: *Examples of generated datasets for robotic tasks.*

learning-based methods are able to directly achieve object recognition. They are often based on CNN which requires a large amount of data to train their models. Objects in the dataset used for object recognition should have different sizes, occlusion and scales, and be captured by a variety of viewpoints. Our simulator is able to generate plenty of data satisfying the above requirements. Therefore, the dataset generated by our simulator can be used to train object recognition networks. Some example of generated images used for object recognition is shown in Figure 4.18(c).

6D object pose estimation. For robotic grasping and manipulation tasks, estimating its 6D pose is a key factor. To estimate the 6D pose, traditional methods first extract and match local features. Then based on the matched features, the pose is estimated by solving a Perspective-n-Point(PnP) problem [LMNF09, RRKB11, MAMT15]. Nonetheless, these methods have difficulty to estimate pose with texture-less objects. Recently data-driven methods that use deep networks for pose estimation from RGB-D images have been proposed [TSF18, XSNF17, WXZ$^+$19]. Our simulator not only produces color-and-depth image pairs, but also generates their corresponding 6D poses. Thus, the data generated by our simulator can be used to train deep learning approaches for the 6D object pose estimation task.

Visual navigation. Training robots by trial-and-error in the real world is an expensive, tedious and in some cases dangerous process. One promising approach that addresses this issue is the utility of virtual world [XZH$^+$18, AWT$^+$18]. Our virtual environment provides various types of information such as photo-realistic color-and-depth image pairs, sensor poses and 6D pose for all the joints of the robot. Such rich data makes it possible to design reward functions for robotic vision tasks, and our simulator allows researchers to integrate reinforce learning approaches to our virtual environment. Thus, our virtual environment provides the possibility for robots to learn to recognize objects, localize and navigate themselves to a target position automatically. Besides, our view synthesis module narrows the reality gap by providing photo-realistic rendering, which makes it easier to transfer trained models from virtual to reality.

4.5 EXPERIMENTAL RESULTS

We evaluate PreSim on our three own datasets (Study room, Table1 and Table2), four datasets (Attic, Dorm, Playroom, Reading corner) from [HRDB16], and two datasets (Ballet and Breakdancers) from [ZKU$^+$04]. The breakdancers and ballet datasets are different from the above seven datasets, for they are dynamic scenes. Each of them contains a sequence of 100 color-and-depth image pairs, captured by static eight cameras which are positioned along an arc.

4.5.1 EVALUATION OF VIEW SYNTHESIS

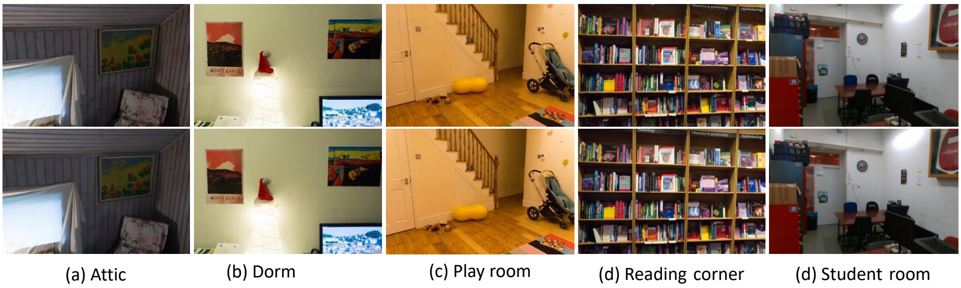

(a) Attic (b) Dorm (c) Play room (d) Reading corner (d) Student room

FIGURE 4.19: *Qualitative comparison of ground truth images (second row) with synthesized images (first row).*

Overall performance. We randomly choose an image from the dataset as our ground truth image and then use our view synthesis method to synthesize the chosen image. Figure 5.7 shows some examples of synthesized images and their corresponding ground truth images. Figure 6.4 shows some additional synthesized images. From Figure 5.7

and Figure 6.4 we can see that our proposed method is able to provide high-quality synthesized images.

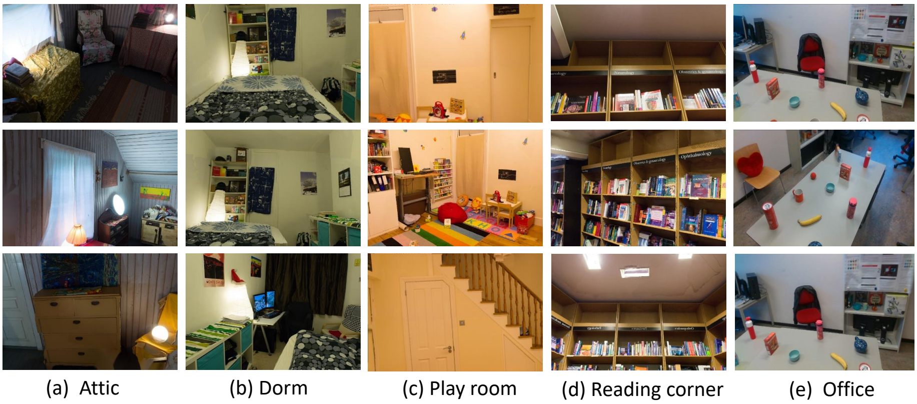


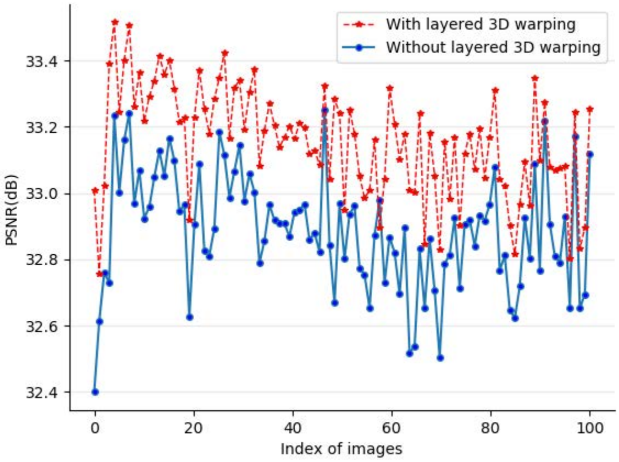
FIGURE 4.20: *Example results of synthesized images from different datasets.*

Effectiveness of adaptive view selection. The quality of the synthesized image is influenced by the quantity of the correct images used. We use the peak signal-to-noise ratio (PSNR) to evaluate the image quality. A higher PSNR value means a higher image quality. Table 4.2 shows the average PSNR over 100 images generated in positions with and without enough well-chosen input images. It can be seen that our view selection method significantly improves the PSNR of synthesized images. It indicates that our view selection approach is able to avoid selecting incorrect or redundant views, resulting in sharp synthesized images with the high PSNR.

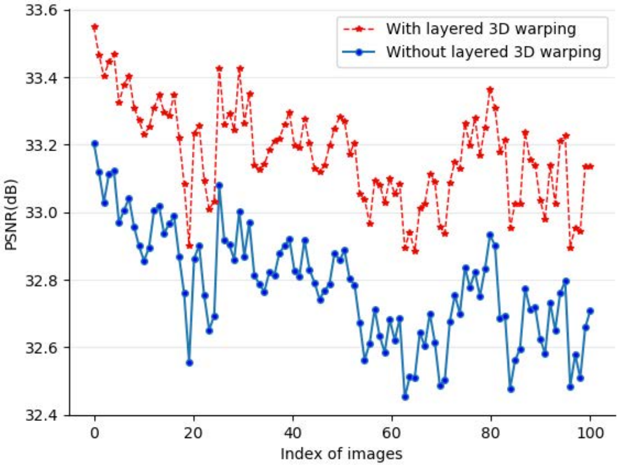
TABLE 4.2: *The PSNR comparison with and without variable input images.*

	the average PSNR over 100 images (dB)		
	variable input images	two input images	Three input images
Attic	33.28	25.51	30.41
Dorm	33.71	17.84	28.75
Ballet	33.49	26.34	29.13
Table1	34.31	24.90	27.91
Table2	33.21	20.87	25.34
Playroom	33.53	22.53	29.03
Study room	31.75	19.72	27.98
Breakdancers	34.67	28.25	31.24
Reading corner	31.78	20.11	25.67

Effectiveness of layered 3D warping. Figure 4.21 compares the PSNR with and without layered 3D warping on a variety of datasets. We can see that our layered 3D warping consistently improve the PSNR through all the testing frames. This is because the layered 3D warping has the ability to better handle the occluded elements with layered depth maps.



(a) *Study room*

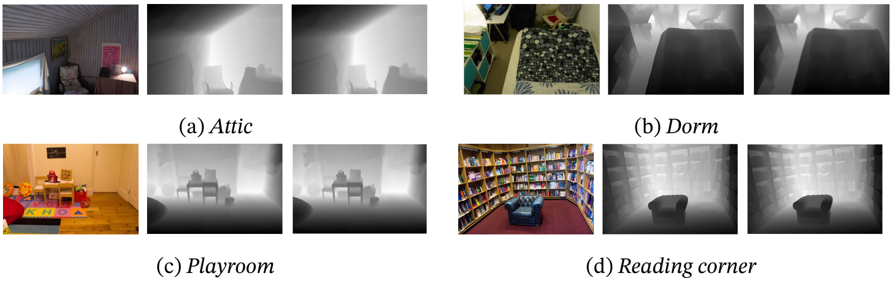


(b) *Reading corner*

FIGURE 4.21: *PSNR comparison with and without layered 3D warping on each frame.*

4.5.2 VALIDATION TASKS LEARNED IN PRESIM

In our experiments, we use a number of vision-based tasks to validate PreSim.



(a) *Attic*

(b) *Dorm*

(c) *Playroom*

(d) *Reading corner*

FIGURE 4.22: *Examples of depth prediction. The first column is the color image, the second column is the ground truth depth map and the third column is the predicted depth map.*

Depth estimation. The dataset generated by our simulator can be used as training data for single-view depth estimation with deep learning algorithms. We train Densedepth [AW18], a popular network architecture for depth prediction on our dataset. Then we test the trained model with color images that are never seen during training. Figure 6.7 visualizes depth predictions from a random number of testing images. To prove the effectiveness of our simulator, in Figure 4.23 we also test the trained model on a popular depth dataset, NYUDv2 [SHKF12].

We can see that knowledge learned from our simulated data can be seamlessly transferred to real-world data in terms of accuracy, which indicates that our datasets can bridge the gap between simulation and reality. It is also verified by Table 4.3, where we evaluate the performance of depth prediction quantitatively based the root mean squared error (RMS) (lower is better).

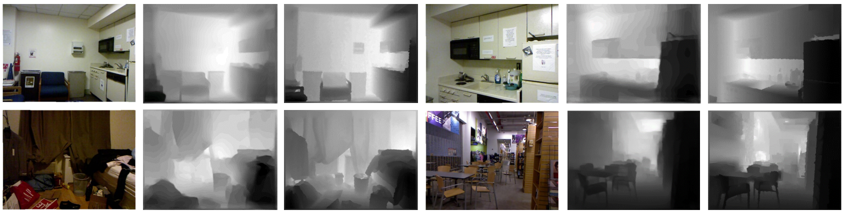
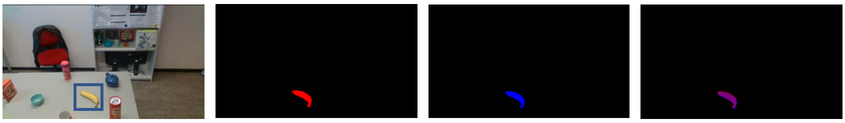


FIGURE 4.23: *Examples of depth prediction on NYUDv2 dataset. The first and fourth columns are color images, the second and fifth columns are predicted depth maps and the third and sixth columns are ground truth depth maps.*

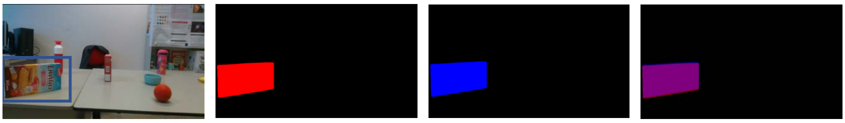
TABLE 4.3: *Quantitative evaluation of the depth prediction in terms of RMS.*

Dataset	Attic	Dorm	Playroom	Reading corner	NYUDv2	Table1	Table2
RMS	0.411	0.435	0.427	0.449	0.791	0.481	0.495

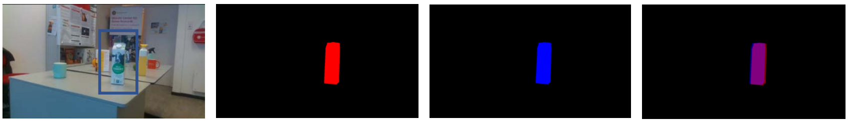
Object recognition. In order to recognize objects in images, we resort to semantic segmentation networks which classify each pixel in the image into an object class. We use our dataset to train the segmentation network introduced by [XSNF17]. Then, the trained model is tested with color images that are never seen during training. This network is an encoder-decoder architecture which takes an image as input, predicts an object label for each pixel in the input image, and generates a semantic segmentation image as output. Figure 4.24 shows the segmentation results for four objects.



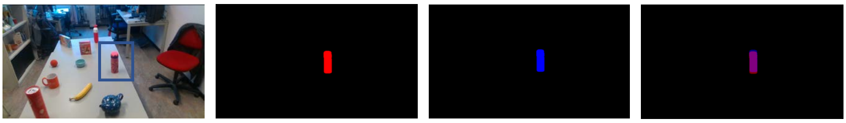
(a) *Banana*



(b) *Biscuit_box*



(c) *Milk_box*



(d) *Vacuum_cup*

FIGURE 4.24: *Results of semantic segmentation. The first column is the color image containing the target object , the second column is the ground truth mask for the object, the third row is the predicted mask and the last column is the comparative map between the predicted and the ground truth masks.*

We use the pixel Intersection over Union (IoU) as the evaluation metric to evaluate the performance of the trained model. The pixel IoU quantifies the percent overlap between the ground truth mask and the predicted output. Specifically, it first computes the number of pixels in the overlap between ground truth and predicted masks, and then divides the number of pixels by the total number of pixels in area of union of the two masks. Table 4.4 shows the performance of the trained model. As shown in this evaluation, it comes to the same conclusion as the depth estimation experiments: knowledge learned from our synthesized dataset can be successfully transferred to real-world data.

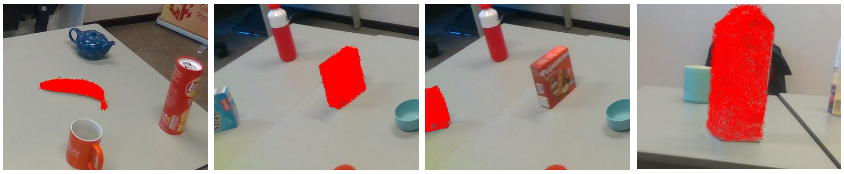
TABLE 4.4: *Quantitative evaluation of the semantic segmentation in terms of IoU.*

Object	banana	biscuit_box	chips_can	cookie_box	gingerbread_box	milk_box	pasta_box	vacuum_cup
IoU	91.2	89.1	89.5	80.3	90.4	81.1	88.9	82.5

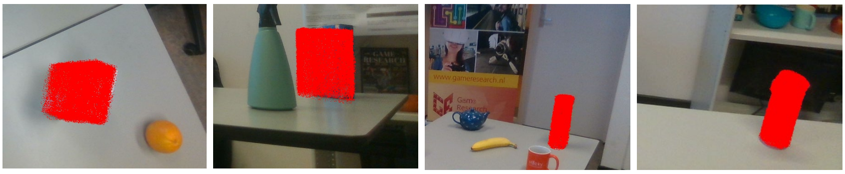
6D object pose estimation. We generate a benchmark dataset using our simulator for 6D object pose estimation. Based on the generated dataset, we organized the Shape Retrieval Challenge benchmark on 6D object pose estimation (`https://yhldrf.github.io/Datasets.github.io/`). The goal of this benchmark is to investigate how different state-of-the-art pose estimation approaches perform in terms of various object properties, including shapes, sizes, textures, changing light conditions, and occlusion. This benchmark gives us insight into the current state of the field of pose estimation. Besides, we learn important lessons from the current pose estimation algorithms, including the advantages and disadvantages of state-of-the-art approaches and understand where researchers' attention should be paid to make progress, see [YVA⁺20] for more details.

Several research groups participated in our pose estimation challenge. They trained their networks on our synthesized training dataset and tested their trained models on a testing dataset including synthesized and real captured images. The DenseFusion network [WXZ⁺19] is trained with our data, and then tested with synthesized and real captured images in order to see if it can transfer the knowledge to the real world data. We visualize the estimation results as shown in Figure 4.25. It can be seen that the pose estimation approach can provide accurate 6D poses for eight objects, which indicates that knowledge learned from our synthesized dataset can be successfully transferred to the real world data without domain constraints.

We present ablation studies to investigate the differences of using: (1) captured images and (2) photo-realistic images as our training images. We use DenseFusion [WXZ⁺19] and GraphFusion which achieves the best performance on our benchmark dataset for the performance evaluation. Both models are trained on 1000 captured and synthesized images respectively, and tested on 100 real captured images that are never seen during training. Comparison results are shown in Table 4.5.



(a) *Qualitative evaluation of 6D pose estimation on the synthesized dataset*



(b) *Qualitative evaluation of 6D pose estimation on the real captured dataset*

FIGURE 4.25: *Example of 6D pose estimation results for eight objects (from top to bottom: banana, gingerbread_box, biscuit_box, milk_box, pasta_box, cookie_box,chips_can, and vacuum_cup) with DenseFusion [WXZ+19].*

TABLE 4.5: *The 6D pose estimation accuracy in terms of ADD using different input images.*

	Real		Synthetic	
	DenseFusin	GraphFusion	DenseFusin	GraphFusion
banana	0.79	0.75	0.82	0.82
biscuit_box	0.80	0.83	0.89	0.91
chips_can	0.21	0.33	0.51	0.66
cookie_box	0.40	0.51	0.56	0.61
gingerbread_box	0.66	0.72	0.83	0.88
milk_box	0.22	0.41	0.50	0.66
pasta_box	0.69	0.68	0.74	0.81
vacuum_cup	0.47	0.47	0.61	0.60
MEAN	0.53	0.59	0.68	0.74

It is noteworthy that photo-realistic images are able to improve the performance of 6D object pose estimation approaches by a large margin. For example, DenseFusion and GraphFusion models trained on synthetic data outperform the models trained on real captured images by 14% and 13% in terms of the vacuum cup, respectively. The main reason is that real images captured by humans have limited viewpoints and lack the richness of scenes, while synthesized photo-realistic images add extra richness and diversity to the dataset. This is a useful finding, for synthesizing photo-realistic images needs less hardware and does not require any human effort to capture and annotate training images.

4.6 CONCLUSION AND FUTURE WORK

We propose PreSim, a 3D photo-realistic virtual environment simulator to develop vision-based algorithms for AI research. By leveraging a variety of indoor environment datasets and augmenting the data through image based rendering, we provide a large amount of photo-realistic color-and-depth image pairs with ground truth 6D poses. The generated data can be used for training and testing data-driven approaches for various AI applications such as depth estimation, object recognition and 6D object pose estimation. Experiments demonstrate our simulator narrows the reality gap between the virtual environment and the real scene, so that computer vision-based algorithms developed in the simulator can be transferred to real vision applications without domain adaption.

Limitation and future work. Even though our virtual environment provides photo-realistic color-and-depth image pairs, the current rendering method is limited by quality of the initial capture, and we also suffer from the same limitation as all 3D reconstruction methods and especially for texture-less and transparent objects. As a result, new methods are required to generate precise 3D models for these objects. Another limitation is that our virtual environment does not include dynamic contents, such as moving people and objects. Our future work is to combine high-quality view synthesis approaches and virtual object datasets to provide realistic dynamic virtual environments.

5

SIM-TO-REAL 6D OBJECT POSE ESTIMATION DATASET CONSTRUCTION

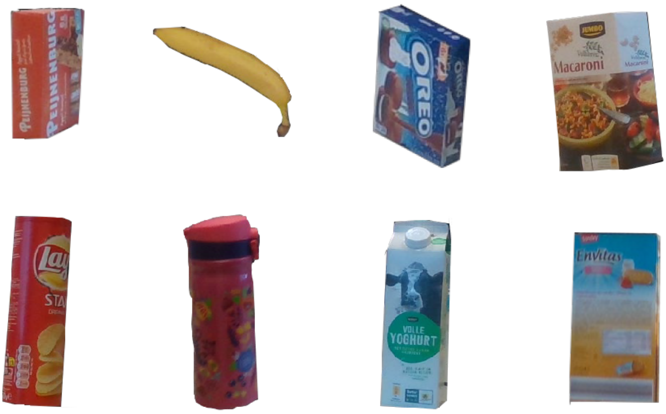


FIGURE 5.1: *Overview of our dataset*

This chapter presents RobotP dataset designed for benchmarking in 6D object pose estimation which plays an important role in robotic grasping and manipulation research. The RobotP dataset consists of photo-realistic indoor scenes, and the objects in it cover a variety of shapes, rigidity, sizes, weight and textures, as shown in Figure 5.1. It is freely distributed to research groups worldwide by the Shape Retrieval Challenge benchmark on 6D pose estimation (`https://yhldrf.github.io/Datasets.github.io/`). For our dataset provides a large amount of high-resolution color-and-depth image pairs with ground truth 6D poses, deep learning-based techniques designed for 6D object pose estimation can be trained on it.

In this chapter we first present an extensive literature review on existing benchmarks and proposed datasets for robotic vision tasks. Then we analyze their scopes, advantages and limitations. Based on the literature survey, this chapter addresses two key challenges of 3D datasets used for 6D object pose estimation: (1) how to produce representative datasets containing high-quality color-and-depth image pairs with a variety of viewpoints, accurate ground truth 6D poses, 3D models, object masks and 2D bounding boxes, and (2) how to take advantage of simulated environments to generate infinitely a large amount of data.

5.1 INTRODUCTION

Benchmarks play an important role in the development of robot research fields. They allow comparing different algorithms to offer insight into the effectiveness of each approach. Recent years have witnessed a large number of data-driven approaches that are successfully applied to address robotic vision problems, such as robotic manipulation and grasping. A key requirement in benchmarks to evaluate these approaches is a good dataset. The dataset should be representative enough for the problem at hand and also contain a considerable amount of variability. However, there are few datasets satisfying the above requirements, especially for 6D object pose estimation, due to the inherent difficulty of data collection.

Generating 6D object pose estimation dataset presents specific challenges. The first challenge is selecting and modeling objects which are suitable for benchmarking 6D object pose estimation performance. Most of the research groups select objects based on the aims they plan to achieve [DUB+17]. As a result, the selected objects maybe do not cover various pose estimation challenges, and not be available to other researchers (e.g., they are only available in certain regions). To address these problems, when selecting objects, we take several practical issues into consideration, such as the size, cost and characteristic of the object. To generate high-quality 3D models, we first use a well-chosen 3D camera to collect RGB-D images and then propose to generate the 3D model for each object by an image-based 3D reconstruction approach.

The second challenge is to provide high-quality RGB-D images, ground truth poses, segmentation masks and 2D bounding boxes for each object. 3D cameras allow easy 3D acquisition of objects, but have several limitations. For example, depth images have missing data and do not align well with their corresponding color images. Even though depth recovery algorithms [YYL+14] provide aligned depth images, they fail in occlusion regions when the camera is near the object. In contrast, multi-view stereo (MVS) can achieve better results for these regions. Taking advantage of these two kinds of depth images, we propose a novel depth generation approach to create high-quality depth images by aligning and fusing them.

The Structure from Motion (SfM) [SF16a] which is based on feature matching to estimate 6D poses is often used to generate ground truth poses. A fundamental limitation of SfM is that it is unable to provide accurate pose for texture-less objects. To address this problem, a pose refinement approach combining local and global pose optimization is introduced. Object mask and 2D bounding box annotation is a time-consuming and expensive process, as the annotation is often generated by humans [RDS+15]. Instead of relying on humans, we propose a novel method to generate accurate segmentation masks and 2D bounding boxes automatically and cost-effectively.

The third challenge is generating large numbers of scene images captured in a variety of viewpoints. Collecting real-world data is a tedious and labor intensive process. Compared with datasets produced by real-world data, synthesizing such a dataset requires less hardware, time and human labor while it is more likely to result in better quality. Taking advantage of image based rendering which can provide the free-viewpoint and realistic imagery of real scenes, we generate a large number of reasonable and photo-realistic images with ground truth 6D poses. Even though we use synthesized images, they are still useful, as the synthesized images are photo-realistic, which are able to bridge the reality gap that allows models trained with synthetic data to the real world without domain adaption.

Even though more and more algorithms, aiming to estimate the 6D object pose have been published, it is unclear how well scenarios and methods perform. New approaches are usually compared with only a few competitors on a particular dataset. To address these issues, BOP benchmark [HMB+18] is proposed, which combines eight datasets in a unified format. However, the datasets used by BOP benchmark have high cost (time and money) associated with ground truth annotation. Besides, these datasets have low resolution and limited viewpoints. Unlike these datasets, our dataset has higher resolution and the distance and view angle between the object and the camera are more various. Our main contributions are summarized as follows:

- A representative dataset providing high-quality RGB-D images, ground truth poses, object segmentation masks, 2D bounding boxes and accurate 3D models for daily used objects with different sizes, shapes, and textures, which covers a wide range of pose estimation challenges.

- A novel pose refinement approach that uses a local-to-global optimization strategy to achieve the improved accuracy of each pose and global pose alignment.

- A novel depth generation algorithm producing high-quality depth images, which is able to accurately align the depth image to its corresponding color image and fill missing depth information.

• Careful merging of multi-modal sensor data for object reconstruction, followed by an algorithm to produce the segmentation mask and 2D bounding box for each object automatically.

• A training dataset generated by a free-viewpoint image based rendering approach in a simulated environment. It provides a large amount of high-resolution and photo-realistic color-and-depth image pairs which have plausible physical locations, lighting conditions, and scales.

• The Shape Retrieval Challenge benchmark on 6D object pose estimation. The benchmark allows evaluating and comparing pose estimation algorithms under the same standard. Evaluation results indicate that there is considerable room for improvement in 6D object pose estimation, particularly for objects which have dark colors or reflective characteristics, and photo-realistic images are helpful to increase the accuracy of pose estimation algorithms.

5.2 RELATED WORK

In this section we discuss the most related works in datasets and benchmarks for robotic vision tasks.

Datasets. Prior works collect various datasets for vision-based applications such as object detection and image classification [EVGW+10], as well as for benchmarking in 3D shape retrieval [shr]. However, few datasets are available for 6D object pose estimation which plays an importance role in robotic grasping and manipulation. LineMOD [HHC+11] and YCB-Video [XSNF17] dataset are the two mostly used 3D object datasets for 6D object pose estimation. However, they have several limitations: the objects are often located in the center of the image plane; images are captured in similar distances; captured images have low resolution; and data annotation for these datasets is tedious and labor-intensive.

The KIT Object Models Database [KXD12] contains 2D images and 3D triangulated mesh models of over 100 objects which are obtained semi-automatically. Even though the number of objects in this dataset is large, the viewpoints are limited and the objects are not easily accessible to other researchers. With the development of data-driven methods for robotics applications [MML+18], the importance of synthetic data has been highlighted. Recent works [DFI+15, WSH+19] combine real and synthesized data to generate 3D object datasets, which render 3D object models on real backgrounds to produce synthesized images. While the backgrounds are realistic, the synthesized images are not photo-realistic. This is because the rendered objects are often flying in midair or out of context [DFI+15]. Unlike these methods, we are able to mimic the physical behavior of the camera and provide reasonable and photo-realistic images.

Benchmarks. Benchmarks is an important part in robotic vision research and its necessity has been recognized by many researchers [dPMM06, MLKN09]. The BOP benchmark [HMB$^+$18] combines eight different datasets containing daily used indoor objects for 6D object pose estimation. However, the captured images in these datasets have limited view angles and positions. The KITTI benchmark [GLU12] provides real-world outdoor data to evaluate approaches designed for autonomous driving, but it needs to record a large amount of new data using special sensors. To compare and evaluate algorithms for robotic grasping, the OpenGRASP benchmarking suite [UKA$^+$11] provides the simulation environment containing test cases, robot models and scenarios to test methods and rank them. However, the simulation environment is not photo-realistic, resulting in the reality gap. The probabilistic object detection challenge based on a collection of simulated indoor images [HDS$^+$20] aims to standardize the evaluation of object detection for robotics applications. While it provides a large number of synthetic images, the viewpoints are also limited.

5.3 THE ROBOTP DATASET

Our goal is to build a benchmarking dataset that allows evaluating and comparing the performance of different 6D pose object pose estimation methods under the same standard. We aim to cover as many pose estimation challenges as possible, including occlusion, poor lighting conditions, and varying viewpoints, shapes and textures, with a special focus on the effect of training images.

The dataset generation works as follows: We first select eight representative and daily used objects based on many practical issues (Section 5.3.1). Then we collect real-world data for these objects by a well-chosen 3D camera under different scenarios (Section 5.3.2). Next, from the collected data, we estimate ground truth 6D poses for these objects (Section 5.3.3) and generate high-quality depth images (Section 5.3.4). After that, we reconstruct textured 3D models, and based on the 3D models, we generate object masks and 2D bounding boxes automatically (Section 5.3.5). Furthermore, to augmenting the collected data, we produce a large number of photo-realistic images with ground truth 6D poses (Section 5.3.6).

5.3.1 OBJECT SELECTION

The first step of generating the RobotP dataset is to choose representative objects that are frequently used in daily life. When selecting objects, several issues have been considered:

(1) In order to cover as many aspects of pose estimation challenges as possible, the selected objects have a variety of sizes, shapes, textures, and reflective properties. For example, objects with few textures are added to the dataset, as it is a challenge for pose estimation approaches to estimate 6D poses for texture-less objects.

(2) We aim to provide a 3D dataset allowing easily carrying, shipping and storing,

which is helpful to carry out robotic manipulation experiments in the real world. Thus, the portability of the object is taken into consideration.

(3) To make the dataset easily reproducible, we choose the popular consumer products which are low price and easy to buy as our target objects.

With consideration of these practical issues, we select eight representative objects to create our dataset, as shown in Figure 5.1.

5.3.2 COLLECTING SCENE DATA

The second step is to collect a set of color-and-depth image pairs which are able to represent the selected objects. We use these images to generate 3D models for the selected objects and synthesize new images. When collecting the scene data, there are two main research questions that should be solved: (1) which types of 3D camera we should choose to obtain high-quality color and depth images, and (2) how can we collect the data that allows us to produce better 3D models. To investigate the first question, three main issues should be considered:

•The resolution of the captured image should be as high as possible. This is because with high resolution images, we are able to get richer information about the captured object. Even though we can upsample the depth image to get a higher resolution image, the details of objects are still lost.

•The range of the 3D camera should be long enough, so that we can get images with a variety of view positions. Besides, the camera should be portable and low-cost, allowing large groups of inexperienced users to collect data.

•The 3D camera should have a high frame rate. The higher frame rate promises better tracking of capture processes, improving the quality of generated 3D model.

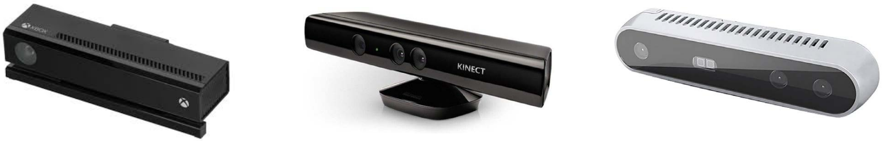


FIGURE 5.2: *Different 3D cameras. Left: time-of-flight camera. Middle: structured-light camera. Right: depth-from-stereo camera.*

After analyzing these practical issues, we search for 3D cameras that satisfy the above requirements. There are three main types of 3D cameras: time-of-flight, structured-light and depth-from-stereo 3D cameras, as shown in Figure 6.2. A detailed summary of the available consumer 3D cameras can be found in [GVS18]. The most popular 3D camera is Kinect V2, which is based on TOF to estimate depths and has been used by many researchers. However, it has several disadvantages: (1) it is not able to provide high-resolution images; (2) the camera calibration is not precise enough; (3) Microsoft has announced that it is no longer manufacturing Kinect. Thus, we do not choose Kinect V2 as our 3D camera. Another popular type of 3D cameras is based on structured light, such as Kinect V1. These cameras rely on recognizing a specific pattern generated by a laser projector in a single image to estimate depths. Therefore, they can be used at night and the power consumption is low. However, they are easily influenced by sunlight and other structured-light devices. Besides, they require a dedicated power source, for the laser projector has to illuminate the entire scene, and the power consumption grows with range.

Depth from stereo is another alternative approach to structured light. Intel RealSense D400 serial cameras (e.g., depth camera D415) that use depth from stereo to estimate depth attract more attentions. The Intel RealSense D415 provides more accurate depth perception and longer range, which is helpful to capture more details on small objects. Besides, D415 uses the infrared projector to improve the depth perception ability for texture-less scenes. This is a big benefit for improving the quality of the 3D model generated by the collected data. Because of these reasons, the D415 is selected as our target 3D camera to collect data. Table 5.1 describes the basic features of the Intel RealSense D415.

TABLE 5.1: *The basic parameters of the Intel RealSense D415 camera.*

Camera	Baseline	Depth FOV HD (degree)	IR Projector FOV	Color Camera FOV	Z-accuracy (or absolute error)	Module Dimensions (mm)
D415	55mm	H: 65±2/ V:40 ±1/ D:72±2	H: 67/ V:41/ D:75	H:69 ±1/ V:42±1/ D:77±1	<2%	X=83.7/ Y=10/ Z=4.7

To solve the second problem, we also need to take some practical issues into consideration:

(1) Visual overlap. To make sure that the details of objects are able to be reconstructed completely, the adjacent images should have enough visual overlaps between each other. Thus, when capturing the object, we need to move the camera slowly to reduce the motion blur and increase the overlap. Our goal is to make sure the overlap between two adjacent images is more than 50 percent.

(2) Textures. Features also play an important role in 3D modeling. The captured images should have rich texture which allows us to extract enough features. Therefore, we should avoid capturing completely texture-less regions, such as white walls. Since our dataset contains texture-less objects, we also need to place additional background objects (e.g., posters) to make sure we can get enough features.

(3) Viewpoints. In order to get a wide variety of images, we need to capture images from different viewpoints and avoid repeatedly capturing images at the same position. At

the same time, we should make sure we can get enough images from a relatively similar viewpoint. This is because we need enough overlaps to reconstruct the 3D model.

(4) Illumination conditions. To ensure the 3D camera have the best performance, we should capture images with similar illumination conditions. For high dynamic range scenes containing very bright, direct sunlight to extreme shade, there is no differentiation in bright areas as everything appears just pure white, and there is no differentiation in darker areas as everything appears pure black, which makes it impossible to get features.

With the consideration of the above practical issues, we acquire RGB-D videos by the Intel RealSense D415 camera connected to a laptop. Depth and color frames are captured with resolution of 1280×720. For camera calibration, we use its default parameters, as Intel has its own calibration system which has advances over the free calibration software. It has the ability to calibrate both extrinsic and intrinsic parameters, and calibrate multiple cameras simultaneously. Apart from calibration, we use the Intel "High Density" setting for depth calculation. It provides us with better quality depth images containing few holes and allows us to capture as much data as possible in all the depth ranges. We store scans as compressed RGB-D data on the connected laptop that allows recording RGB-D videos for several hours.

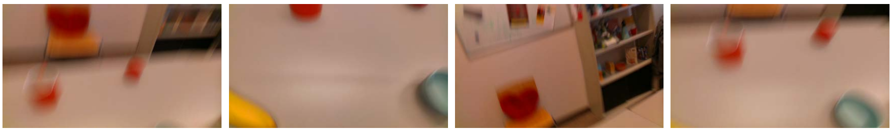


FIGURE 5.3: *Examples of captured blurring images.*

After obtaining RGB-D videos, we process the data offline to remove motion blur which is caused by the hand-held camera, as shown in Figure 6.4. To get clear color-and-depth images, we extract all the frames from the RGB-D video. Then based on the motion blur metric proposed by [CDLN07], we detect blurring images and then delete them. The reason to remove blurring images is that we are likely to get wrong features in the blurring image, which results in the failure of feature matching process. Since our pose estimation is based on the matched features, without good matching, the pose estimation process may fail. After that, we remove images having too similar viewpoints. This is because more images do not necessarily improve the 3D reconstruction results while may lead to a slow reconstruction process, or even result in a worse reconstruction result. The whole selection process is done by downsampling images with an equal sampling interval and at the same time we check the motion blur to make sure the selected image is sharp. Finally, we get a subset of color images as our input data to estimate camera poses.

5.3.3 GROUND TRUTH POSE ESTIMATION

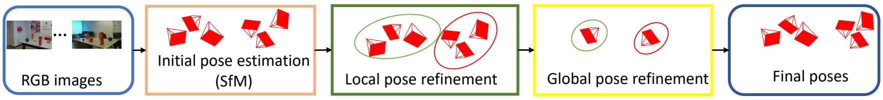


FIGURE 5.4: *The pipeline of the pose estimation process. The input are RGB images and the initial poses of these images are estimated by SfM. After that, the initial poses are refined locally and globally.*

We use COLMAP [SF16a], a state-of-the-art Structure from Motion system to calibrate color images and estimate initial camera poses. However, due to the inherent limitation of 3D reconstruction methods, when the change between two input images becomes larger, the estimated camera pose is more likely to have errors. Besides, if the image does not have enough features, the estimated pose has errors, too. In order to improve the accuracy of estimated poses, we perform pose refinement combing local and global pose refinement steps. Instead of refining all camera poses simultaneously, we first divide poses into groups and then refine poses in each group. After that, we choose a key pose from each group and then refine these key poses globally. Our aim is to respect the local details and also be compatible with global consistency. The pipeline of pose estimation is described in Figure 5.4.

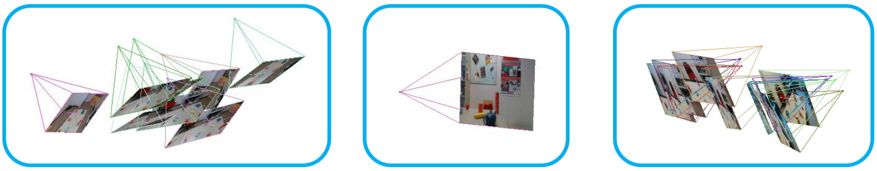


FIGURE 5.5: *The local pose groups. They are clustered based on angle and distance similarities.*

In the pose refinement process, poses are first grouped based on their similarities among each other. The similarity is measured by comparing the angle and distance between two poses. We randomly define one pose as our key pose and then calculate the angle and distance between the key and other poses. We first rank the poses based on the distances they have with the key pose. Next, we check if their corresponding angles are bigger than the field of view of the camera. If so, we then delete the pose from the rank. After that, we choose up to ten top poses as a local group. Then the other groups are obtained by the same pipeline from the remaining poses. Figure 5.5 shows the clustered groups used for pose refinement.

We refine the camera pose in the local group under the consideration of its neighbors with bundle adjustment. Bundle adjustment method is often used in 3D reconstruction as the joint non-linear refinement of camera parameters and feature points. We choose the key pose's corresponding image as the key image, and then we detect feature points between the key image and other images. For each image, SIFT features are detected and matched. The reason to use SIFT feature is that it is robust for the major variation, such as image translation, scaling, and rotation. Next, we project these feature points into the 3D world space using the camera pose of the key image. Lastly, we apply local bundle adjustment to refine camera poses with these 3D points. To account for potential outliers, the Huber function is used as the robust loss function in local bundle adjustment, and we use Ceres Solver library to solve the optimization function. The cost function for grouped images is defined as:

$$\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n}||e_{i,j}||^2 = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{n}||f(P_j, x_i) - X_j||^2, \tag{5.1}$$

where $e_{i,j}$ is the reprojection error and P_j is the projection matrix. Assume that n 3D points are seen in m views, function f projects point x_i in the image plane to 3D world space and X_j is the reference point in the world space.

After the local pose refinement, we build a feature group by computing features from key images for global pose refinement. However, the feature group may contain multiple instances of the same real-world point which are found in separate pairwise image matches. To address this issue, we only add features which have been used in local pose optimization process to the feature group. We then compute the 3D positions of these features based on the optimized poses. Once the feature points and their corresponding 3D points are obtained, we use the same loss function as local pose optimization to refine these global poses.

5.3.4 DEPTH GENERATION

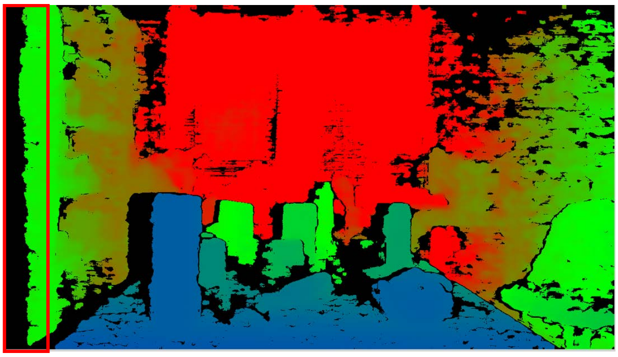


FIGURE 5.6: *The captured depth image. The red rectangle shows left invalid depth band.*

Color images have been used successfully by deep learning for many robotic vision tasks, such as object recognition and scene understanding. However, grasping objects with the exact physical dimensions is a very hard problem which requires not only RGB data but also extra information. Depth images can provide such a brand-new channel of information and are essential elements in datasets designed for robotic grasping tasks. Apart from that, depth images play an important role for improving the performance of view synthesis approaches. However, captured depth images often suffer from missing information and misalignment between color-and-depth image pairs due to the inherent limitation of depth cameras. The Intel RealSense D415 camera also has the same limitation. Even though the alignment and hole filling methods from Intel are applied, the quality of the captured depth image is still low, especially when the camera is near the object (see Figure 5.6). Therefore, new algorithms are required to refine these captured depth images to improve their quality.

5.3.4.1 DEPTH AND COLOR IMAGE ALIGNMENT

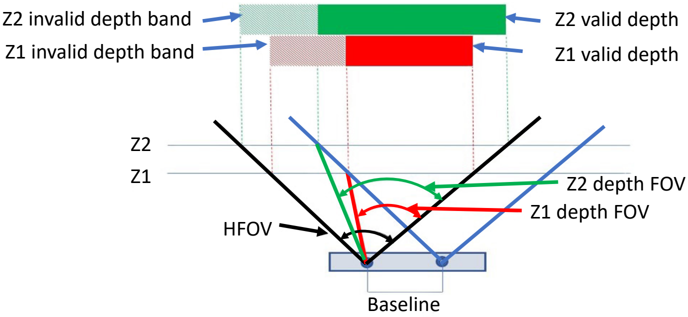


FIGURE 5.7: *Description of depth field of view to depth image.*

Figure 5.7 describes the process of depth field of view to depth image. For the Intel RealSense D415 camera is based on depth from stereo to calculate depth values, it uses the left sensor as the reference for stereo matching. This leads to a non-overlap region in the field of view of left and right sensors where no depth information is available at the left edge of the image (see Figure 5.6).

Based on the stereo vision, the depth field of view (DFOV) at any distance (Z) can be defined by [Int]:

$$DFOV = \frac{HFOV}{2} + tan^{-1}(tan\frac{HFOV}{2} - \frac{B}{Z}), \tag{5.2}$$

where HFOV is the horizontal field of view of left sensor on depth module and B is the baseline between the left and right sensor.

We can see that when the distance between the scene and the depth sensor decreases, the invalid depth band increases, which results in the increase of the invalid depth in the overall depth image. This is because the overall depth image is the combination of invalid depth band and valid depth image. Furthermore, if the distance between the object and the depth sensor decreases, the misalignment between the color and depth image also increases, as shown in Figure 6.5. This is because the alignment utility performs per-pixel geometric transformation estimated by the provided depth data . If there is too much invalid depth data in the depth image, the estimated transformation matrix is likely to be inaccurate, resulting in the failed alignment.

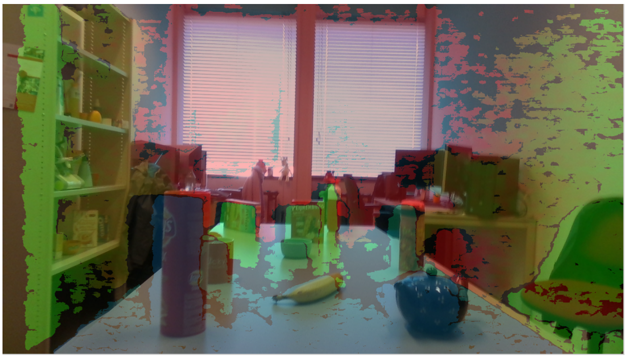
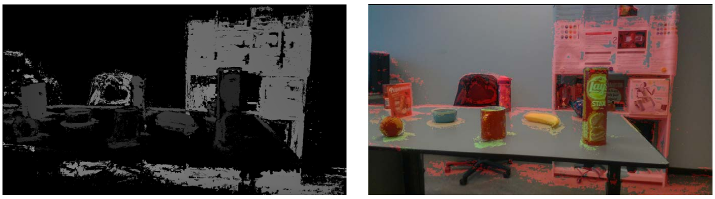


FIGURE 5.8: *Misalignment of color-and-depth image pairs. The images are generated when the distance between the object and camera is near, showing large misalignment.*

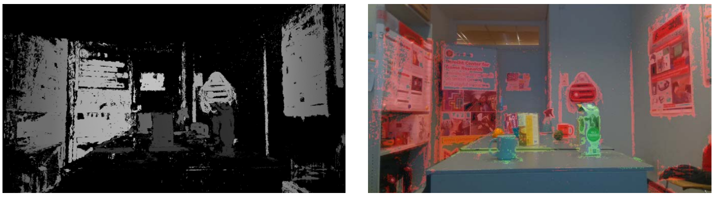
In previous works, in order to align the depth image to its corresponding color image, a new depth image is created, which has the same size as the color image but the content being depth data calculated in the color sensor coordinate system. In other word, to create such a depth image, the projected depth data is determined by transforming the original depth data to the color sensor coordinate system based on the transformation matrix between the color and depth sensors. However, it is difficult to get the correct transformation matrix, as the depth and color images are defined in different spaces and have different characteristics.

To solve this problem, we first create an estimated depth image for each color image by MVS from COLMAP. The estimated depth image has better alignment with the color image (see Figure 5.9), as it is estimated with the consideration of photometric priors and global geometric consistency. Then we align captured depth images to estimated depth images to achieve better alignment between color-and-depth image pairs. Since the captured and estimated depth images have the same characteristics, it is easier for us to align the captured depth image to the estimated depth image.

For two color image patches, the similarity is measured by comparing color differences. Similarly, in order to find correspondences in depth images, we compare depth values and normals between them. To compare depth values, we should make sure the



(a) *Table1*



(b) *Table2*

FIGURE 5.9: *The depth images are estimated by COLMAP on different datasets showing better alignment.*

estimated and captured depth images have the same scene scale. However, a fundamental limitation of the estimated depth image is that we do not know the scale of the scene. We use linear regression in a RANSAC loop to find the metric scaling factor. After obtaining the scaling factor, we use it to scale the estimated depth image to the captured depth image.

To estimate normals, we convert the depth image to a point cloud by camera intrinsic matrix. Then we compute the surface normal at each point in the point cloud. Determining the normal to a point on the surface can be considered as estimating the normal of a plane tangent to the surface. Thus, this problem becomes a least-square plane fitting estimation problem [Rus10]. Let x be a point, and $\vec{n}$ be a normal vector. The plane is represented as $\pi(x, \vec{n})$. The distance from a point q_i in a point set Q to the plane π is defined by $d_i = (q_i - x) \cdot \vec{n}$. For the values of x and $\vec{n}$ fit the least-square sense, $d_i = 0$. Then we define x as the centroid of Q:

$$x = \frac{1}{k} \sum_{i=1}^{k} q_i, \tag{5.3}$$

where k is the number of points in Q. The solution for estimating the normal $\vec{n}$ is therefore reduced to analyze the eigenvectors and eigenvalues of a covariance matrix C created from Q. More specifically, the covariance matrix C is expressed as :

$$C = \frac{1}{k} \sum_{i=1}^{k} \alpha_i (q_i - x)(q_i - x)^T,\ C \cdot \vec{v}_j = \beta_j \cdot \vec{v}_j, j \in \{0, 1, 2\}, \tag{5.4}$$

where α_i is a possible weight for q_i, $\vec{v_j}$ is the j-th eigenvector of the covariance matrix and β_j is the j-th eigenvalue. Based on 5.4, the normal $\vec{n}$ can be computed.

For the generated dataset is used for object pose estimation, our aim is to produce better aligned color-and-depth pairs for objects not the overall scene. We first extract a patch P_c containing a target object in the captured depth image D_c. Then we define an offset map whose size is the same as P_c but the content being index differences between P_c and its corresponding patch in the estimated depth image D_e. In ideal conditions, the values in the offset map should be zeros.

The matching process which is based on PatchMatch [BSFG09] is implemented by first initializing the offset map with predefined values by prior information. Then extract a patch Q_e based on the offset map as the corresponding patch for P_c. The pixel $p(x, y)$ in P_c is transformed to pixel $q(x', y')$ in Q_e by:

$$q(x', y') = p(x + x_{off}, y + y_{off}), \tag{5.5}$$

where (x_{off}, y_{off}) is the index offsets for each pixel in P_c.

After that, we perform an iterative process which allows good index offsets propagating to its neighbors to update the offset map. The iteration starts with the top left pixel and then an odd iteration starting with the opposite direction. We firstly calculate the depth differences dd_i between pixel $p_i \in P_c$ and pixel $q_i \in Q_e$, and the angles between normals $\vec{n_{p_i}}$ and $\vec{n_{q_i}}$. If the angle is smaller than a predefined threshold, dd_i is saved. Then, if dd_i is smaller than its neighbors, we replace the offsets of p_i's neighbors with p_i's offset. After every iteration, we calculate the sum c_i of dd_i. We stop propagation when the change of c_i is negligible. Finally, based on the offset map, we map the captured depth image to the estimated depth image. The depth alignment process is summarized in Algorithm 2.

5.3.4.2 DEPTH FUSION

Even though the captured depth image is aligned to its corresponding color image, the invalid depth band still exists. Apart from that, it has missing information and noise, especially when reflective or transparent objects are captured. Therefore, the quality of the captured depth image is not good enough for synthesizing high-quality images. On the other hand, the estimated depth image generated by MVS not only has better alignment with its corresponding color image, but also provides useful depth information in regions where the depth camera has poor performance. However, the estimated depth image is not able to provide reliable depth information for texture-less or occluded objects, due to the inherent limitations of MVS, as shown in Figure 5.10. Thus, we are not able to use the estimated depth image to synthesize new images, either.

For the characteristics of the captured and estimated depth images are complementary, we fuse the captured and estimated depth images together to create a fused depth image. The fused depth image takes advantage of both real captured and estimated depth

Algorithm 2 Overview of depth alignment procedure

```
Input: Captured depth image $D_c$, estimated depth image $D_e$;
Output: aligned depth map $D_{c1}$ for $D_c$;
Run RANSAC to find the metric scaling factor.
Extract patch $P_c$ in $D_c$.
Calculate scaled depth values and normals of $P_c$.
Initialize offset map $O$.
Find a patch $Q_e$ in $D_e$ based on $O$.
for $p_i \in P_c$ and $q_i \in Q_e$ do
    Calculate depth difference $dd_i$ and normal angle $ang_i$ between $p_i$ and $q_i$.
    Run PatchMatch propagation to update offset map $O$.
Mapping $D_c$ to $D_{c1}$ based on $O$.
```

images, resulting in the improved quality. We generate the fused depth image D_f by maximum likelihood estimation [THo19]:

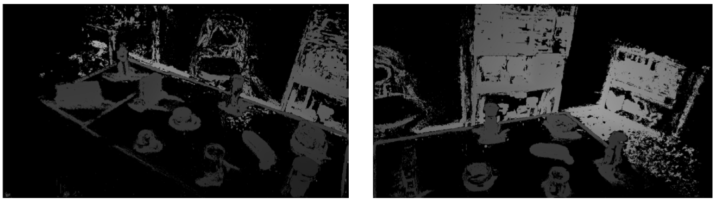
$$D_f = \underset{d}{argmax}((R_e P_e)(R_c P_c)), \tag{5.6}$$

where d is the depth value, R_e is a reliability map and P_e is a probability map produced from the estimated depth image, and R_c is a reliability map and P_c is a probability map produced from the captured depth image.

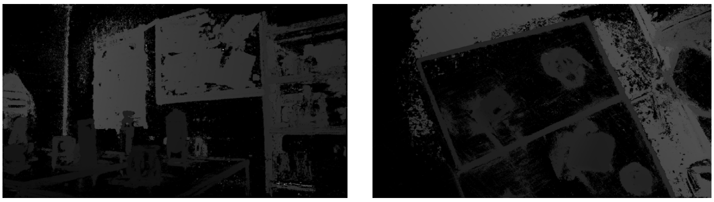
The reliability map R_c for the captured depth image is computed according to the variation between the depth value and camera's range. The reliability r_c of each depth value d is calculated by

$$r_c = \begin{cases} \dfrac{MaxD^2 - d^2}{MaxD^2 - MinD^2}, & MinD < d < MaxD \\ 0, & otherwise \end{cases}, \tag{5.7}$$

where $MaxD$ and $MinD$ are the minimum and maximum distances which the depth camera is able to measure. From 5.7, we can see that when the distance between the



(a) *Table1*



(b) *Table2*

FIGURE 5.10: *The estimated depth images by multi-view stereo on different datasets.*

camera and the scene increases, the precision decreases. After calculating the reliability for each pixel, we get the reliability map R_c.

The estimated depth image is generated based on COLMAP. It runs in two stages: photometric and geometric. The photometric stage only optimizes photometric consistency during depth estimation. In the geometric stage, a joint optimization including geometric and photometric consistency is performed, which can make sure the estimated depth maps agree with each other in space. We generate the reliability map R_e for the estimated depth image by comparing depth values d_p and d_g computed from photometric and geometric stages, respectively:

$$r_e = \begin{cases} \dfrac{\delta - |d_g - d_p|}{d_g}, & d_g < |d_g - d_p| \\ 0, & otherwise \end{cases}, \tag{5.8}$$

where r_e is the reliability of each depth value and δ is the maximum accepted depth difference which is set to be 50 in our experiments. When the depth value calculated based on geometric consistency has a large difference compared with the depth value calculated based on photometric consistency, we consider this value is unreliable.

One of the main limitations of the reliability map is that it does not take the idea that spatial neighboring pixels are able to be modeled by similar planes into account. In order to solve this problem, we introduce the probability map. To calculate the probability of a depth value in the captured or estimated depth image, we define a (5×5) support region S centered at the pixel i whose depth value is d_i. For each pixel $j \in S$, if j is far from i, it is reasonable to associate a low contribution to j when calculating the probability for

d_i. Following this intuition, the probability p_{d_i} is estimated by

$$p_{d_i} = \sum_{j \in S} e^{-\frac{\Delta_{i,j}}{\gamma_1}} \cdot e^{-\frac{\Delta^{\pi}_{i,j}}{\gamma_2}}, \tag{5.9}$$

where $\Delta_{i,j}$ is the euclidean distance between i and j, $\Delta^{\pi}_{i,j}$ accounts for the distance from j to the plane π calculated by i and S, and γ_1 and γ_2 control the behavior of the distribution (see [THo19] for a detailed description). After calculating the probability for every depth value in the estimated and captured depth images, we produce the probability maps P_e and P_c, respectively.

From these maps, we generate high-quality fused depth images based on 5.6. However, there are still many outliers and misalignment between color-and-depth pixel pairs in the fused image. Therefore, we use the pixel-to-pixel multi-view depth refinement approach which is proposed in Chapter 3 to refine the fused depth image.

5.3.5 3D MODELING

We use the 3D point cloud produced by COLMAP as our initial model. However, COLMAP only uses color images to generate the 3D point cloud. It fails on some objects with fewer features, such as transparent or texture-less objects. To obtain more accurate 3D point clouds, we use depth images generated in Section 5.3.4 to refine the initial model, for they provide reliable depth information in regions with missing features.

To refine the initial point cloud, we project a pixel in a color image to its neighboring images to check if it is visible in them. These neighboring images are selected by view selection approach proposed in Chapter 3. If it is visible in more than five images, we project this pixel to the world coordinate system to get a 3D point which is added to the initial 3D point cloud. Then we check if there are many similar 3D points around it. If so, we will not add this point into the 3D point cloud, for we only save key points in our 3D point cloud. After all the pixels in the image are projected, we remove outliers which are often caused by measurement errors, boundaries of occlusion or surface reflectance by StatisticalOutlierRemoval filter from PCL [RC11]. This filter performs a statistical analysis on the neighboring points of each point. Supposing that the filtered point cloud is Gaussian distribution, all points whose mean distances are outside an interval defined by the global distance mean and standard deviation can be considered as outliers and trimmed from the point cloud. We repeat the previous steps until all the images are projected. The whole process is summarized in Algorithm 3.

Mask and bounding box generation. Apart from 3D models, we also provide masks and corresponding 2D bounding boxes for the objects in our dataset. Our goal is to generate accurate segmentation masks and 2D bounding boxes automatically and cost-effectively. To achieve our goal, we take three practical issues into consideration:

- Quality. Each mask and its corresponding bounding box need to be tight. For example, the bounding box should be the minimum bounding box that fully encloses all

Algorithm 3 Overview of 3D point cloud generation procedure.

```
Input: Input color images $I_1...I_N$ and depth images $D_1...D_N$ which contain the object
    $O$, the initial 3D point cloud $PC'$ and the visible view threshold $\varepsilon$;
Output: The 3D point cloud $PC$ for the object $O$;
for image $i \in I_1...I_N$ do
    Find $i$'s neighboring images $Q$ and calculate the visible time $m_j$ of $i$'s each pixel in Q.
    if the visible time $m_j > \varepsilon$ then
        Project $p_j$ into $PC'$ and check if it is redundant.
Filtering outliers.
Update $PC'$ to $PC$.
```

visible parts of the object. In order to estimate the 6D object pose, the first step is to detect the target object. The quality of masks and bounding boxes influences the performance of object detection algorithms which affects the accuracy of the estimated pose.

•Coverage. Each object instance needs to have a segmentation mask and a 2D bounding box which should only contain the object other than the background. For learning-based object detection and recognition approaches, they need to know exactly which part is the target object in a whole image.

•Cost. The designed algorithm should not only provide high-quality masks and 2D bounding boxes, but also have the minimum cost, as data annotation is a labor intensive and time-consuming process.

To address these issues, we take advantage of the generated 3D point cloud and 6D poses. To produce the segmentation mask, we project the point cloud into each image plane using the estimated camera pose and camera intrinsic matrix. In this way, our goal can be achieved easily. However, this mask may contain missing information caused by the projection errors or inaccurate points in the 3D point cloud. We fill missing information in the mask by the stitching median filter proposed in Chapter 3. After that, we obtain the 2D bounding box by detecting the minimum area of the mask.

5.3.6 PHOTO-REALISTIC RENDERING

In this section, we provide a detailed description of how we generate photo-realistic color-and-depth image pairs based on extremely realistic movements of a robot. Our aim is to overcome the limitations of captured datasets [HHC+11, XSNF17] and build a 3D dataset containing high-quality images generated from rich viewpoints and scales. Inspired by the low cost of producing large-scale synthetic datasets with accurate ground truth information, as well as the recent success of synthetic data used for training 6D pose estimation approaches, we use our physically accurate environment simulator proposed in Chapter 4 to synthesize data for our dataset.

The trajectory generation process should be automated and controllable to avoid unreasonable images and human labor, for we generate data with various angles and distances. Previous works often generate the trajectory of camera poses by the SLAM system which is operated by a person to collect hand-held motions. After that, these poses are inserted into the scenes to synthesize new images. However, this approach depending on humans to collect trajectories limits the potential scale of the dataset. Other methods synthesize images by just randomly projecting 3D objects into an arbitrary scene, for it is difficult to generate new images from the same image distribution. However, the images generated by random poses are unrealistic compared to real-world scenes. For example, the projected objects are often flying in midair or out of context.

To address this issue, we import the robot model which is equipped with cameras into our environment simulator. Then we move the robot to positions where its camera can capture the target object. At the same time, we record these sparse positions as our initial poses, which can be obtained by the tf package from ROS. For each pose, we randomly rotate or move the camera along its axes to obtain new poses. To make the target object visible, we set the maximum rotation angle to be less than 30 degree and the movement distance to be less than $0.2m$. In this way, we are able to generate infinitely many reliable camera poses.

Our poses have three main advantages. Firstly, our poses are random, but always tracking the target object, rather than moving along a wall. Secondly, they contain a variety of movements like that of a person collecting data. Lastly, they also have limited rotational freedom that emphasizes yaw and pitch rather than roll which is less important in 6D object pose estimation.

Based on the generated camera poses, the view synthesis module of our simulator is used to synthesize new color-and-depth image pairs. Even though we set rotation and movement threshold to avoid synthesizing images without the target object, there are still some such images. During offline processing, we project the 3D model of the target object to synthesized images to check if these images contain the target object. If not, we delete the images. In this way, we are able to make sure all the synthesized images satisfy our requirements.

5.3.7 CONTENT

Based on the pipeline described above, we generate the RobotP dataset containing 8 rich-texture, low-texture and reflective objects recorded on two table layouts. The RobotP dataset consists of two subsets. One is a training dataset containing synthesized data and the other is a testing dataset combining of synthesized and captured data. More specifically, we provide the following data:

- 6D poses for each object.
- Color images with the resolution of 1280×720 in PNG.

- Depth images with the resolution of 1280×720 in PNG.
- Segmentation binary masks for each image.
- 2D bounding boxes for each object.
- 3D point clouds with RGB color and normals for each object.
- Calibration information for each image.

This initial release of the dataset contains a training set of 400 synthesized color-and-depth image pairs and a testing set of 100 captured and synthesized color-and-depth image pairs. The dataset is available at `https://yhldrf.github.io/Datasets.github.io/`.

5.4 EXPERIMENTAL RESULTS

We test our approaches on two scenarios (Table1, Table2) used for dataset generation, which contain a variety of objects with different sizes, shapes, textures and occlusion.

Data collection. Figure 5.11 shows the point clouds generated by RGB images captured in different conditions. These conditions include visual overlaps, textures, viewpoints and illumination. The point cloud shown in Figure 5.12 is produced by RGB images which are captured with the consideration of all the conditions mentioned above. As we can see, only when taking all the factors into consideration, we are able to get the best 3D reconstruction results.

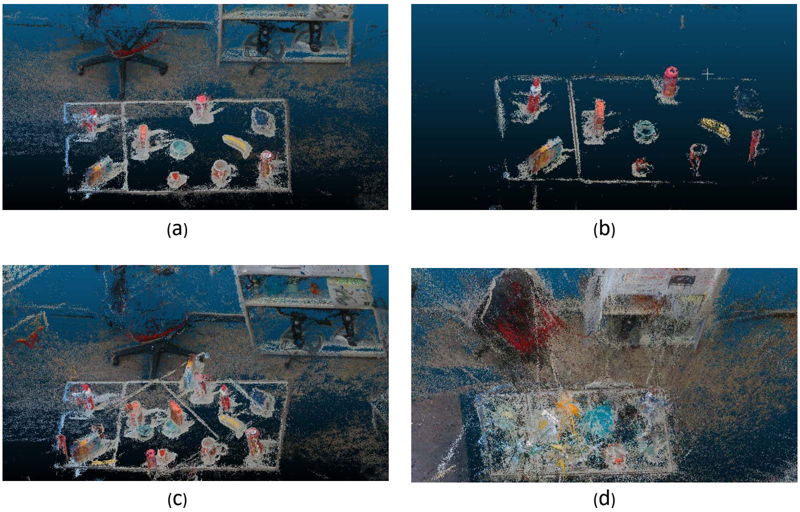


FIGURE 5.11: *The point clouds are generated by RGB images captured under four conditions separately: (a) visual overlaps, (b) textures, (c) viewpoints and (d) illumination.*

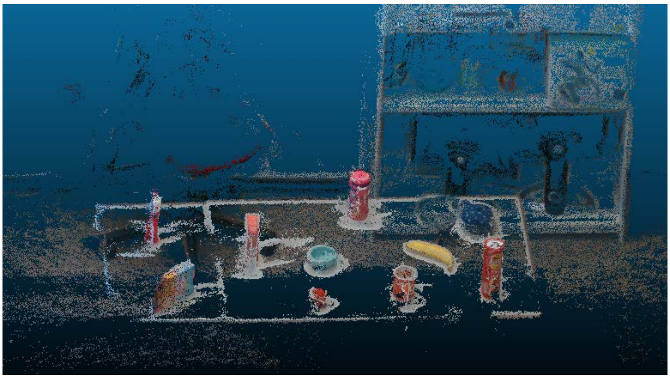
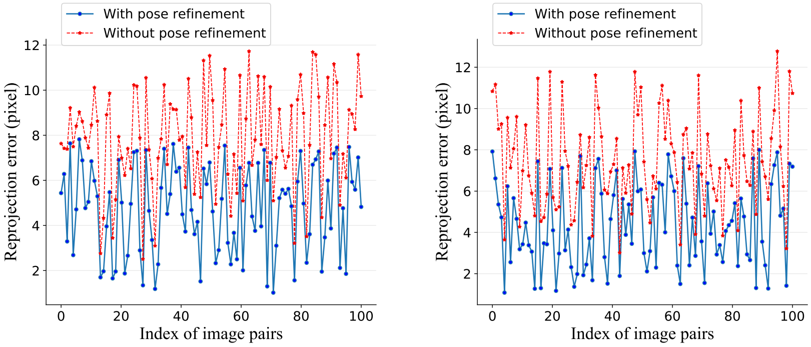


FIGURE 5.12: *The point cloud is generated by RGB images captured with the consideration of visual overlaps, textures, viewpoints and illumination.*

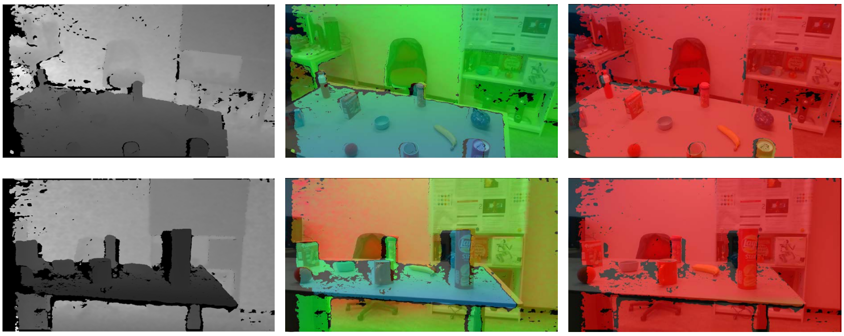
Effect of pose refinement. We use the reprojection error to measure the accuracy of the estimated 6D poses. The reprojection error is calculated by first projecting 2D correspondences in an image to its matching image plane and then computing the pairwise distances in the image space. Figure 5.13 shows the reprojection errors that are calculated over 100 image pairs. From Figure 5.13 we can see that the refined poses have lower reprojection errors throughout all the image pairs, which verifies the effectiveness of the pose refinement step.



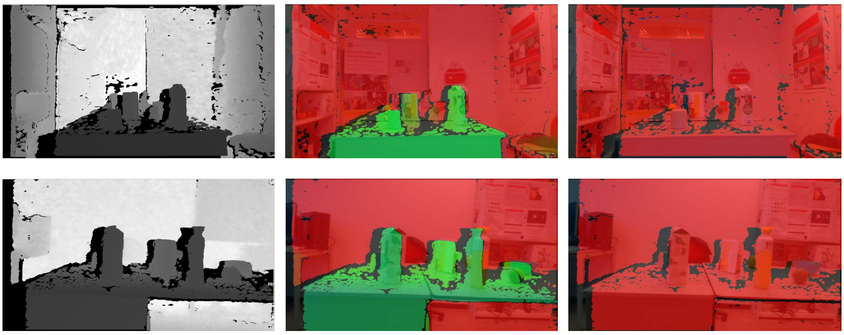
(a) *Table1* (b) *Table2*

FIGURE 5.13: *Reprojection error comparison with and without pose refinement on each image pair.*

Effect of depth alignment and fusion approaches. Figure 5.14 shows depth alignment results on different datasets. It can be seen our alignment approach effectively aligns the depth image to its corresponding color image.



(a) *Table1*

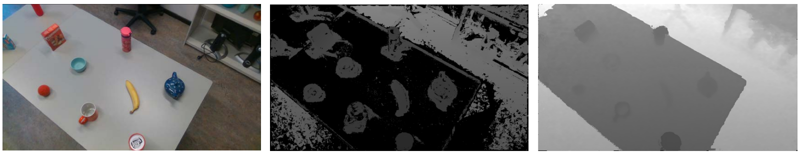


(b) *Table2*

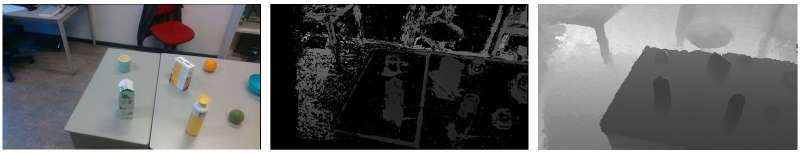
FIGURE 5.14: *Examples of depth alignment results on table1 and table2 datasets. The first column is the aligned depth image, the second column is the matching between captured depth and color images, and the third column is the matching between aligned depth and color images.*

The main advantage of our depth fusion algorithm is its robustness towards texture-less regions. As shown in Figure 6.4.3, the performance of COLMAP degrades significantly when there is texture-less region in the image. In contrast, our method has better performance in such images.

Point clouds of objects. We use point clouds to represent the objects in our dataset. The modeling results are shown in Figure 5.16. Compared with COLMAP, our approach achieves better performance, especially for texture-less objects.

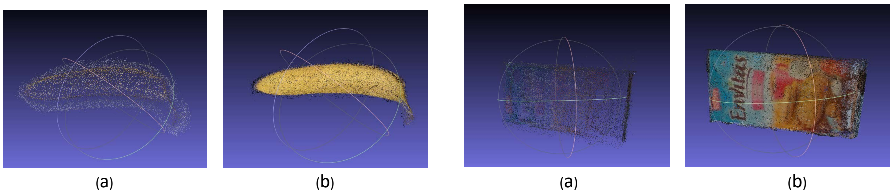


(a) *Table1*

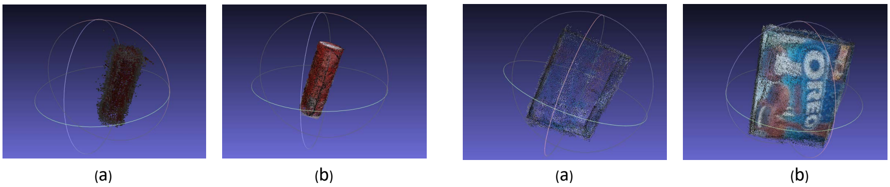


(b) *Table2*

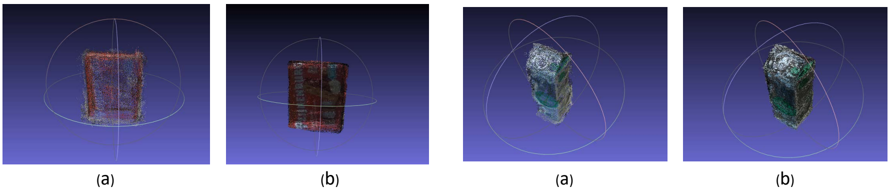
FIGURE 5.15: *The depth fusion results on table1 and table2 datasets. The first column are color images, and the second column are the estimated depth images by COLMAP and the third column are the depth images generated by our approach.*



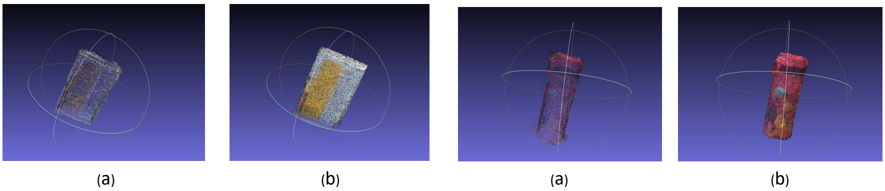
(a) *Banana* (b) *Biscuit_box*



(c) *Chips_can* (d) *Cookie_box*



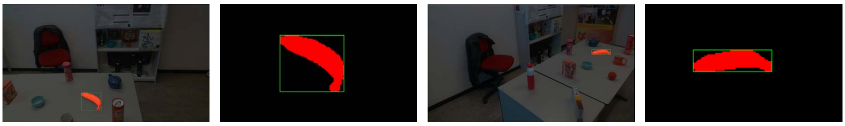
(e) *Gingerbread_box* (f) *Milk_box*



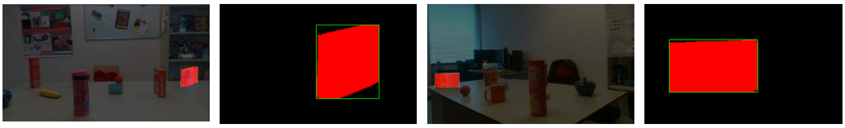
(g) *Pasta_box* (h) *Vacuum_cup*

FIGURE 5.16: *The 3D point clouds of eight objects. The point clouds shown in figures (a) and (b) are generated by COLMAP and our approach, respectively.*

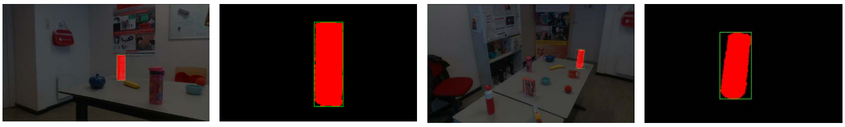
Based on the 3D point clouds, we generate the object mask and 2D bounding box automatically. Figure 5.17 shows example results of object masks and their corresponding bounding boxes. It can be seen that the generated bounding boxes are tight and stably concentrate on the objects, which demonstrates that our method produces highly accurate masks and bounding boxes.



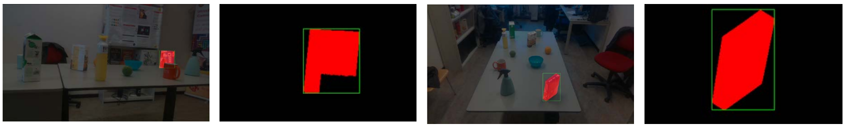
(a) *Banana*



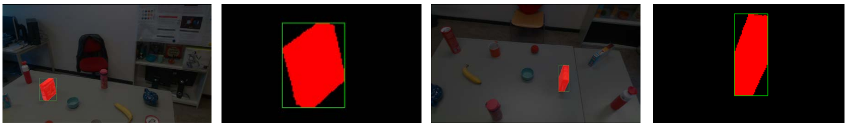
(b) *Biscuit_box*



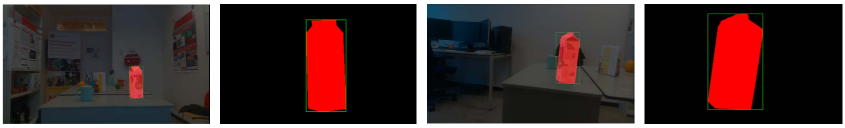
(c) *Chips_can*



(d) *Cookie_box*



(e) *Gingerbread_box*



(f) *Milk_box*

FIGURE 5.17: *Examples of segmentation masks and bounding boxes for different objects.*

Qualitative evaluation of synthesized images. Figure 5.18 visualizes some examples of synthesized images in our dataset. As we can see that our dataset is able to provide photo-realistic color-and-depth image pairs.

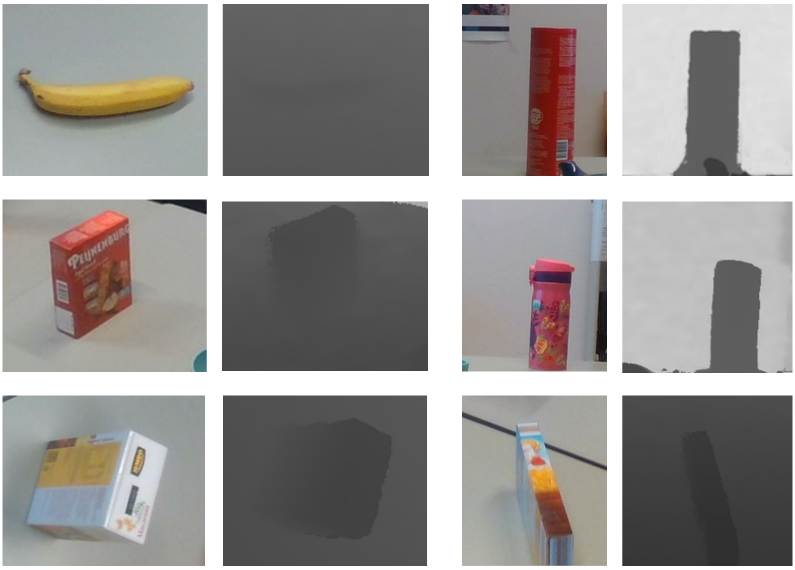


FIGURE 5.18: *Example results of synthesized color-and-depth image pairs for six objects.*

5.5 6D OBJECT POSE ESTIMATION CHALLENGE

6D pose estimation is crucial for many vision-based applications, such as augmented reality, robotic grasping and visual navigation. However, estimating object poses is challenging, for the objects in the real world have various shapes, sizes and textures. Apart from that, the captured images from them are affected by sensor noise, changing lighting conditions and occlusion. On the other hand, different pose estimation methods have different strengths and weaknesses, and it is unclear how well they perform, for the lack of benchmarks with high quality datasets.

To address these issues, we organize the Shape Retrieval Challenge (SHREC) benchmark on 6D object pose estimation. It consists of two components: (1) an openly accessible dataset, and (2) an annual competition and corresponding workshop. We use different evaluation metrics to compare the proposed methods based on our RobotP dataset.

The competition and workshop provide a way to measure and track the progress of pose estimation and discuss the lessons learned from different research groups. Besides, the benchmark and workshop have the potential to further enrich and boost the research of 6D object pose estimation and its applications. A more detailed description of 6D object pose estimation algorithms will be presented in the next chapter.

5.6 CONCLUSION AND FUTURE WORK

In this chapter, we have presented the RobotP dataset, an extremely realistic dataset containing high-resolution color and depth images, ground truth 6D poses, segmentation masks, 2D bounding boxes and 3D models. To build the dataset, we choose eight representative objects with the consideration of many practical issues including cost, sizes, shapes, textures and portability. Then we use a well-chosen 3D camera to collect data for these objects. To estimate accurate 6D poses for the collected data, a pose refinement approach combining local and global optimization is introduced. To further improve the quality of captured depth images, we generate new depth images by aligning and fusing estimated depth images generated by MVS and captured depth images. Based on the fused depth images, we produce accurate 3D models, and then we use these models to generate segmentation masks and 2D bounding boxes automatically. Besides, taking advantage of our simulator described in Chapter 4, we synthesize a large number of photo-realistic color-and-depth image pairs with ground truth 6D poses.

Our dataset is designed to serve as a widely used benchmark dataset for robotic grasping and manipulation tasks. Apart from that, it can be used for other robot vision tasks, such as object detection, semantic segmentation and depth estimation. This dataset is freely distributed to research groups through the 6D object pose estimate challenge organized by us. We hope our dataset will satisfy the growing need of deep learning approaches and benefit the 6D object pose estimation research.

We also note some limitations of our dataset, which we hope to improve in the future.

- The synthetic dataset needs to be expanded by adding more challenging objects such as reflective and texture-less objects, and challenging conditions such as heavy occlusion and varying lighting conditions.

- We plan to make object models easily integrated into a variety of robot simulation packages. When these modes are imported into a simulation environment, a variety of motion planners and optimizers can use these models either as collision or manipulation objects.

- We also plan to make the dataset generation process more user-friendly, so that users can generate their own dataset for their purpose.

- More details about the objects will be proposed, including the mass, size and inertia of each object.

6

6D OBJECT POSE ESTIMATION

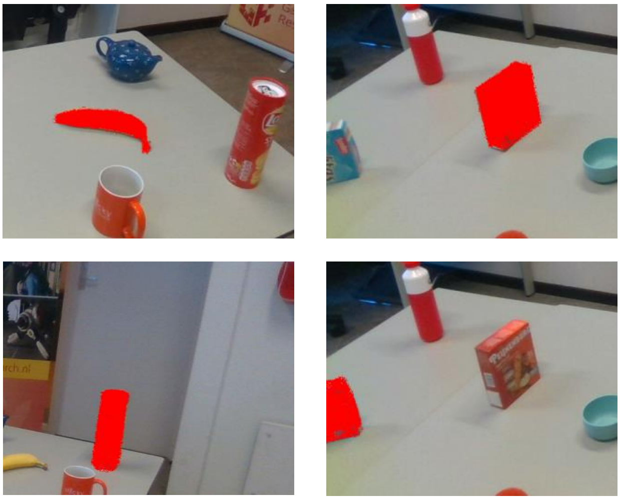


FIGURE 6.1: *Visualization of estimated poses by our method. Each 3D model is projected to the image plane with the estimated 6D pose.*

The ability to estimate 6D object pose including its orientation and location is essential for many vision-based applications, such as visual navigation, robotic grasping and virtual reality. The awareness of the 3D rotation and 3D translation matrices of objects in a scene is referred to as 6D, where the D stands for the degree of freedom of the pose. However, heavy occlusion, changing light conditions and cluttered scenes make pose estimation challenging. To address these issues, we created a new dataset (RobotP) and organized the Shape Retrieval Challenge (SHREC) benchmark on 6D object pose estimation, introduced earlier in Chapter 5.

In this chapter we conduct a variety of experiments based on our benchmark to investigate two main issues: (1) how different state-of-the-art pose estimation approaches perform in terms of various object properties, including shapes, sizes, textures and occlusion; (2) what lessons we can learn from the current pose estimation algorithms and where the attention should be paid to make progress.

Apart from that, we propose a novel approach to estimate the 6D pose of a given object. Our method effectively extracts color and geometric features by learning-based networks. Then, we properly fuse them by a graph attention network, which makes our approach robust to handle heavy occlusion, low texture and sensor noise for 6D object pose estimation. The evaluation results indicate that our method significantly improves the accuracy of the estimated 6D pose.

6.1 INTRODUCTION

6D pose estimation is crucial for augmented reality, virtual reality, robotic grasping and autonomous navigation [WXZ+19]. However, the problem is challenging due to the variety of objects in the real world. They have varying 3D shapes and the quality of captured images from them is affected by sensor noise, changing lighting conditions and occlusion. With the emergence of cheap RGB-D sensors, the accuracy of estimated object poses is improved for both rich and low texture objects [TSF18]. Nevertheless, existing methods still have difficulty to meet the requirement of accurate 6D pose estimation for objects with texture-less and reflective property, and heavy occlusion.

Previous methods using RGB images as input work by extracting and matching handcrafted features and then 6D pose is estimated by solving a Perspective-n-Point (PnP) problem. Such methods are often fast and robust to occlusion. However, they heavily rely on rich features and are unable to handle texture-less objects. Instead of relying on improving handcrafted features, we learn more robust features and semantic cues by applying deep learning models.

Taking advantage of depth sensors, RGB-D based methods [BKM+14, KMT+17] predict more accurate 6D pose of low texture objects even in poor lighting conditions than RGB-only methods. However, these algorithms often require a time-consuming pose refinement step (e.g., iterative closest point (ICP) algorithm) to improve pose accuracy.

Recent approaches [WXZ+19, XCJ19] introduce end-to-end deep learning networks to improve the performance of 6D object pose estimation with the fused per-pixel color and geometric feature extracted from RGB-D images. In order to extract geometric information from the depth image, they first transform the depth image to a point cloud and then process each point independently. However, these methods do not consider relationships between point pairs, resulting in the loss of local features and the decreased accuracy of the estimated 6D pose. To address this issue, we apply edge convolution which considers both local and global point structures to compute geometric features.

Apart from discriminative geometric features, fusing color and geometric features is also important for improving the accuracy of the estimated 6D pose. Due to these two types of features are defined in different spaces, fusing them is a key challenge. Existing approaches [WXZ$^+$19, XCJ19] just concatenate these two kinds of features, which fail to fully exploit the correlation between them. Unlike previous approaches, we introduce a graph attention based framework to effectively compute the hidden representations between visual and geometric features and then fuse them properly.

Even though more and more algorithms, aiming to estimate the 6D object pose have been published, it is unclear how well scenarios and methods perform. This is because a new approach is often compared with few methods in a special dataset. Based on our benchmark described in Chapter 5, we conduct a variety of experiments to investigate how different pose estimation approaches perform in terms of various object properties, such as shapes, sizes, textures and occlusion. It gives us insight into the current state of the field of pose estimation. Besides, we learn important lessons from the current pose estimation algorithms, including the advantages and disadvantages of state-of-the-art approaches. In summary, we present three main contributions:

- A comprehensive evaluation of 6D object pose estimation approaches. We use different evaluation metrics to compare the proposed methods on our benchmark. Evaluation results indicate that approaches that fully exploit the color and geometric features are more robust for 6D pose estimation of reflective objects and occlusion. Besides, methods that estimate the 6D pose in a single and consecutive network are more robust to texture-less objects and run faster.

- An efficient feature extraction method extracting representative local and global geometric features from point clouds, which makes it robust to handle heavy occlusion, low texture and sensor noise for 6D object pose estimation.

- A novel multi-feature fusion network that improves 6D pose prediction performance by applying a graph attention network to fully exploit the relationship between visual and geometric features and compute hidden feature representations between these features.

We show performance results on a variety of objects (see Figure 6.1), demonstrating that our proposed method provides high accurate 6D object pose. Our approach achieves state-of-the-art performance on commonly used two datasets, including LineMOD [XSNF17] and YCB-Video [HHC$^+$11] datasets and our new dataset.

6.2 RELATED WORK

6D object pose estimation has been an active research area for a long time. Here we only discuss the most related previous work in 6D pose estimation. A thorough review of 6D pose estimation approaches can be found in [SGHSK20].

Pose estimation based on RGB images. To estimate the 6D object pose, traditional RGB based methods first establish 2D-3D correspondences between 2D key points and 3D models either by extracting and matching local features or predicting 2D projections of predefined 3D key points. Based on these correspondences, 6D poses are estimated by solving PnP problems [KLS14, WRM+08]. Hand-crafted features, such as SIFT [NH03] and ORB [MAMT15], are often used by these methods, for they are robust to scales, rotation, illumination and view angles. However, the heavy dependence on hand-crafted features and fixed matching process have limited empirical performances of these methods to predict 6D poses for texture-less objects in poor lighting conditions or clustered scenes.

Other methods [XSNF17, KGC15] use deep learning-based approaches to directly estimate 6D object poses from color images. For example, PoseNet [KGC15] and PoseCNN [XSNF17] directly regress to a 6D pose by a CNN-based architecture from a single RGB image. However, their predictions are sensitive to small errors due to the large search space. Besides, these approaches require careful tuning hyper-parameters for their associated loss functions.

Pose estimation based on RGB-D images. A different class of approaches takes advantage of depth sensors that provide rich information to estimate poses for texture-less objects. These methods [JMP+18, KMT+16] extract 3D features from color-and-depth image pairs and then perform correspondence matching to predict 6D poses. Ipose [JMP+18] uses an encoder-decoder architecture to extract features from color image and then computes the 2D-3D correspondences between the color image and the 3D model. Instead of predicting pose directly, the 6D pose is estimated by solving the PnP problem with the obtained correspondences and depth information.

On the other hand, recent methods [MKB+17, WXZ+19] use the fused RGB-D data to estimate the 6D pose directly. Michel et al. [MKB+17] fuse the RGB-D information in the early stage, where the depth information is treated as a fourth channel and concatenated with RGB channels. Alternative solutions including Densefusion [WXZ+19] fuse the color and depth information in the later stage, which generate dense pixel-wise features to estimate poses. However, all the methods fail to effectively exploit the fuse strategy between color and geometric information. Inspired by graph attention networks, we propose a graph attention based architecture to fully exploit the relationships among RGB-D data for 6D pose estimation.

6.3 ANALYSIS OF BENCHMARKING APPROACHES FOR 6D OBJECT POSE ESTIMATION

We conduct a series of experiments to make deeper understanding of 6D object pose estimation, and propose insights for designing the next generation of general 6D object pose estimation algorithms.

6.3.1 EVALUATION METRICS

In our benchmark, we require participants to submit the estimated 6D object poses of the testing set. The performance of 6D object pose estimation is evaluated by ADD(-S) which are the average distance metric (ADD) [HLI+12] and the average closest point distance (ADD-S) [XSNF17].

Given the ground truth rotation matrix R and translation matrix T and its corresponding estimated rotation matrix $\hat{R}$ and translation matirx $\hat{T}$, the ADD computes mean distances between all 3D model points x transformed by $[\hat{R}|\hat{T}]$ and $[R|T]$:

$$ADD = \frac{1}{N} \sum_{x \in S} ||(Rx + T) - (\hat{R}x + \hat{T})||, \tag{6.1}$$

where S is a set of 3D model points and N is the number of points.

The ADD-S is an ambiguity-invariant pose error metric, which takes care of both symmetric and non-symmetric objects into an overall evaluation.

$$ADD\text{-}S = \frac{1}{N} \sum_{x_1 \in S} \min_{x_2 \in S} ||(Rx_1 + T) - (\hat{R}x_2 + \hat{T})||. \tag{6.2}$$

The area under the accuracy-threshold curve (AUC) which is calculated from ADD(-S) is another evaluation metric. Specifically, if the ADD(-S) is smaller than a threshold defined from the diameter of the 3D object model, we consider the estimated pose is correct. Based on that, we define a variable range of thresholds from 0% to 100% of the 3D object diameter and then compute the ADD(-S) for each threshold. With the two sets of values, we can get the AUC, and then the area under the AUC is calculated.

We also use the reprojection error, which is often used to evaluate the performance of 6D object pose estimation approaches based on feature matching, as our fourth performance metric. Rather than computing distances between two 3D point pairs, the reprojection error is calculated by first projecting 3D points into an image plane and then computing the pairwise distances in the image space.

6.3.2 APPROACHES

All the proposed methods are described in the following sections. We choose DenseFusion [WXZ+19] as our baseline approach for the 6D object pose estimation.

6.3.3 DENSEFUSION: 6D OBJECT POSE ESTIMATION BY ITERATIVE DENSE FUSION

DenseFusion [WXZ+19] is a heterogeneous neural network architecture with RGB-D images as input. It processes color and depth information separately and uses a dense fusion network to extract pixel-wise dense features, from which the 6D object pose is estimated. Furthermore, an end-to-end iterative pose refinement network is proposed to further improve the accuracy of the predicted pose while achieving real-time speed.

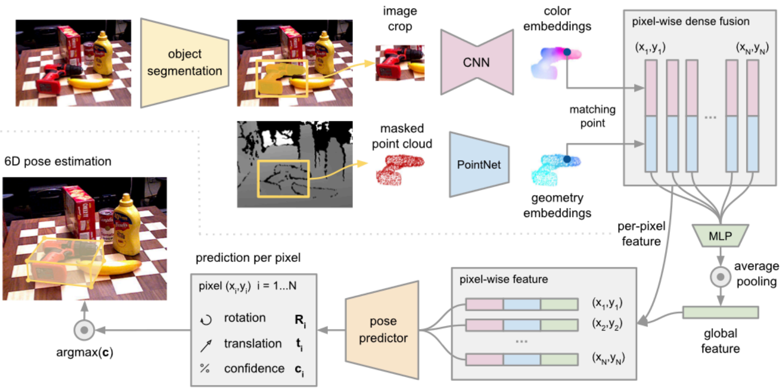


FIGURE 6.2: *Pipeline of the DenseFusion networks. The network first generates object segmentation masks and 2D bounding boxes from color images. The color-and-depth image pairs are cropped using the bounding boxes and fed into learning-based networks to learn features. The learned features are fused at each corresponding pixel. The pose predictor estimates a 6D pose for each fused feature and the predictions are voted to obtain the final 6D object pose.*

Figure 6.2 shows the overall architecture of DenseFusion. The architecture consists of two stages. In the first stage, the target object is detected in the input color image using semantic segmentation from [XSNF17]. Then, the color and depth images are cropped based on the segmentation, and the cropped depth image is transformed to a point cloud using the intrinsic camera matrix. After that, the cropped color image and the point cloud converted by the cropped depth map are fed to the second stage.

In the second stage, the cropped color image is fed to a CNN-based network Resnet-18 [HZRS16] encoder followed by 4 up-sampling layers as decoder to extract color features. The point cloud is fed into a PointNet-based network [QSMG17] by applying a multi-layer perceptron (MLP) to produce geometric features. After that, the color and depth features are fused to estimate the 6D object pose based on an unsupervised confidence score. Lastly, the predicted pose is refined by the iterative pose refinement network.

We implement the DenseFusion network within the PyTorch framework and the model is trained using Adam optimizer with an initial learning rate at 0.0001. The iterative pose refinement module contains a 4 fully connected layers and 2 refinement iterations is used for the experiments.

6.3.4 ASS3D: ADAPTIVE SINGLE-SHOT 3D OBJECT POSE ESTIMATION

Multimodal inputs can improve the performance of various computer vision tasks, but it is usually at the cost of efficiency and increased complexity. ASS3D focuses on RGB-D 6D object pose estimation and exploits multimodal inputs using a lightweight fusion scheme which is complemented by multimodal supervision through rendering. In this way, ASS3D overcomes the complexity of multimodal inputs by transferring it to the model training phase instead of the inference phase. Given the distinct domains that color and depth information resides in, a disentangled architecture is employed, as depicted in Figure 6.3, to process them separately. After that, a learnable fusion scheme is introduced to fuse features.

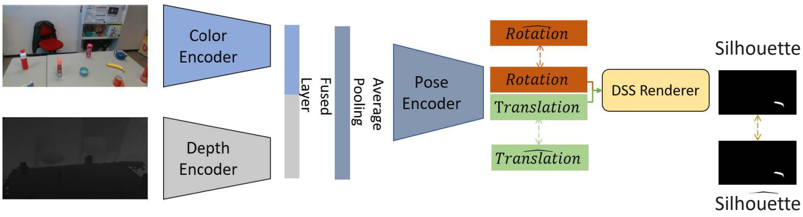


FIGURE 6.3: *Overall network architecture. The color and depth images are processed separately and the extracted features are fused in a later stage. An average-pooling function is applied as the symmetric reduction function. The features are then fed into a pose encoder which directly regresses a rotation and translation. The predicted pose is subsequently used for rendering the object and deriving its projected silhouette. This allows utilizing an additional supervision signal during training, which increases the overall performance of the model.*

More specifically, two ResNet-34 models are used as the backbone encoders for extracting features, which are later fused and flattened by an average-pooling function. This approach allows associating the geometric feature of each point to its corresponding image feature based on a projection onto the image plane using the known camera intrinsic parameters. The fused features are then fed into a pose encoder consisting of three fully connected layers that eventually disentangled to 3D rotation and 3D translation heads. Following the definition of the model's architecture, ASS3D supervises it using a direct pose regression objective as the weighted sum of two different losses. Particularly, a $L2$ loss $\varepsilon_t = ||t - \tilde{t}||$ is used for the translation and a geodesic distance for the rotation $\varepsilon_r = \arccos \frac{trace(R\tilde{R}^T)-1}{2}$, similar to [GZD$^+$20]. The loss for the predicted pose is then:

$$\varepsilon_{pose} = \lambda_{six_d}\varepsilon_t + (1 - \lambda_{six_d})\varepsilon_r, \tag{6.3}$$

where the weight λ_{six_d} acts as a regularization term. This is complemented by a silhouette loss which is enabled by a point splatting differentiable renderer [YSW+19]. The 3D vertices $\nu \in \mathbb{R}^3$ of each object's point cloud is transformed using the predicted pose. The differentiable point cloud renderer then renders the transformed model's silhouette, which is used along with the ground truth annotated silhouettes as an additional supervision signal. Instead of using a traditional intersection over union (IoU) loss, a Gaussian smooth silhouette loss as defined in [GZD+20] is applied for the silhouette loss:

$$\varepsilon_{silhouette} = \frac{1}{N}\sum_{\in\Omega} S\ \mathcal{S}(\tilde{S}) + \tilde{S}\ \mathcal{S}(S), \tag{6.4}$$

where S is a Gaussian smoothing function. The silhouette loss is a smoother objective function compared to the common IoU loss, while it takes the ground truth silhouette into consideration simultaneously, offering that way a fully symmetric objective. However, the most appropriate Gaussian filter to be used is dependent on each object shape, and can also vary during training, offering higher precision as the model converges. Towards that end, a new adaptive filter is used by learning the standard deviation of the Gaussian during training. The final learning objective is a weighted sum of the aforementioned losses:

$$\varepsilon_{total} = \lambda_{pose}\varepsilon_{pose} + \lambda_{silhouette}\varepsilon_{silhouette} \tag{6.5}$$

It is apparent that the introduction of the weights λ_{pose} and $\lambda_{silhouette}$ will introduce similar challenges as aforementioned (e.g., finding the best combination for each object will be challenging and time-consuming). Motivated by that, those two weights are treated as learnable parameters and adding them to the learning objective. Thus, the weights are able to adapt to the various objects and, additionally, to better regularize the two losses during training.

Finally, the model is trained for 100 epochs on a GeForce RTX 2080 TI 11 GB. All the color and depth images are resized to 320×180 resolution, and the batch size is set to 16. For optimizing the model's parameters the Adam optimizer with a learning rate of 1×10^{-4} is used. Additionally, learnable Gaussian standard deviation and the weights of 6.5 are optimized with a SGD optimizer with a learning rate of 1×10^{-5}.

6.3.5 GRAPHFUSION: 6D OBJECT POSE ESTIMATION WITH GRAPH BASED MULTI-FEATURE FUSION

GraphFusion proposes a graph based multi-feature fusion network to improve 6D pose prediction performance, which combines effective feature extraction networks and a graph attention network (GAT) [VCC+17] to fully exploit the relationship between visual and geometric features.

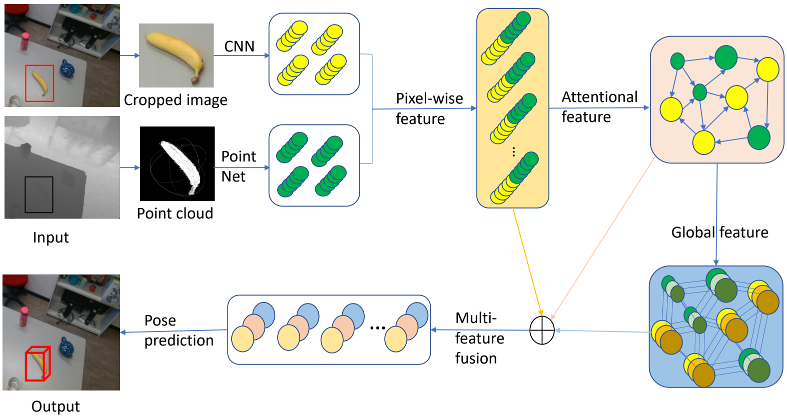


FIGURE 6.4: *Overview of the graph based pose estimation architecture. The input of the networks are captured color-and-depth images pairs. These images are cropped with the semantic segmentation architecture. After that, the visual and geometric features are extracted and fused by a graph attention network which is introduced to exploit the fusion strategy between color and geometric features. The 6D object pose and its corresponding confidence score are predicted by the fused features and the final pose is chosen based on the confidences.*

The aim of this approach is to achieve the real-time 6D pose estimation, using RGB-D images as input, as shown in Figure6.4. A CNN-based encoder-decoder architecture is used to learn visual features from color images. To extract geometric features from the depth map, the depth map is converted to the point cloud using the camera intrinsic matrix. There are two ways to process the point cloud. Classic approaches often convert point cloud data into regular grids by projecting 3D data into 2D images or splitting raw data into 3D voxel grids. Then the transformed data is processed using approaches based on regular data. Other approaches are to directly process each point in the point cloud. PointNet [QSMG17] is the first one to apply this idea, which achieves permutation invariance by use of a symmetric function. Instead of transforming to regular data, a PointNet-based network is used to extract geometric features from the point cloud.

Even with learned features that contains the visual appearance and geometry structure information, accurate 6D object pose also depends on the fused features. To effectively fuse features, a graph attention based framework is introduced to exploit relationship between visual and geometric features, as opposed to prior works which just concatenates these features. Combining the insights above, the approach works as follows:

The input are captured color-and-depth image pairs and a semantic segmentation architecture from [XSNF17] is used to segment the target object and crop the color and depth images. Next, the visual features are extracted by a CNN-based network and geometric representations are computed from the point cloud using a PointNet-based network . The point cloud is generated by converting its corresponding depth map. With these features, a graph attention network is introduced to perform the fusion between color and geometric features. After that, the 6D object pose and its corresponding confidence score are predicted by the fused features, one pose per fused feature. Then, the pose with the highest confidence is chosen as the estimated pose. Lastly, the 6D pose is further improved by iterative pose refinement.

6.3.6 OVERALL PERFORMANCE

The overall performance of DenseFusion, ASS3D, GraphFusion without refinement (GraphFusion_wo) and GraphFusion is shown in Table 6.1 and Table 6.2. DenseFusion and ASS3D are proposed from two different research groups, and GraphFusion without refinement (GraphFusion_wo) and GraphFusion is proposed from one research group. We use ADD, ADD-S and the area under ADD curve (AUC) to measure the prediction.

We can see that GraphFusion achieves the best performance. GraphFusion outperforms ASS3D and DenseFusion 11% and 5% in terms of ADD, respectively.

TABLE 6.1: *Quantitative evaluation of the 6D pose in terms of ADD and ADD-S.*

	DenseFusion		ASS3D		GraphFusion_wo		GraphFusion	
	ADD	ADD-S	ADD	ADD-S	ADD	ADD-S	ADD	ADD-S
banana	**0.86**	0.86	0.70	0.75	0.76	0.80	0.83	**0.87**
biscuit_box	0.91	0.95	0.78	0.88	0.80	0.84	**0.93**	**0.96**
chips_can	0.56	0.94	**0.75**	0.85	0.53	0.57	0.69	**0.97**
cookie_box	**0.62**	0.74	0.49	0.66	0.51	0.56	0.61	**0.75**
gingerbread_box	0.87	0.94	0.63	0.86	0.79	0.83	**0.90**	**0.95**
milk_box	0.50	**0.81**	0.58	0.62	0.52	0.53	**0.66**	0.77
pasta_box	0.77	0.91	0.63	0.72	0.76	0.78	**0.84**	**0.96**
vacuum_cup	0.61	0.90	**0.65**	0.75	0.51	0.53	0.63	**0.97**
MEAN	0.71	0.88	0.65	0.76	0.65	0.68	**0.76**	**0.90**

TABLE 6.2: *The 6D pose estimation accuracy in terms of the area under AUC.*

	DenseFusin	ASS3D	GraphFusion_wo	GraphFusion
banana	**0.77**	0.66	0.66	0.75
biscuit_box	0.77	0.74	0.68	**0.79**
chips_can	0.74	0.72	0.55	**0.76**
cookie_box	**0.67**	0.56	0.53	0.66
gingerbread_box	0.76	0.71	0.64	**0.78**
milk_box	0.66	0.51	0.52	**0.67**
pasta_box	0.74	0.61	0.61	**0.77**
vacuum_cup	0.71	0.64	0.63	**0.74**
MEAN	0.72	0.64	0.60	**0.74**

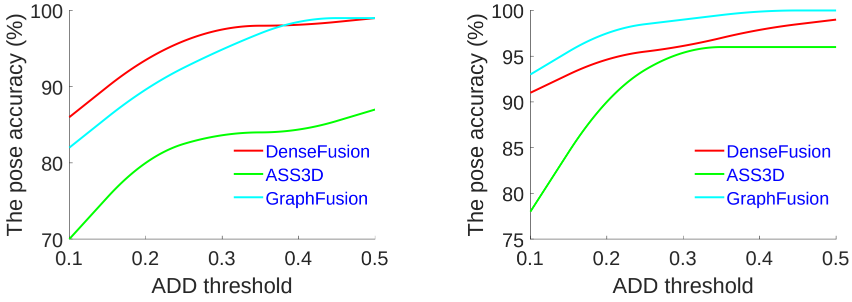
6.3.7 ABLATION STUDIES

We present ablation studies to help better understand the functionalities of different network architectures.

Effectiveness of pose refinement. From Table 6.1 and Table 6.2 we can see that compared with ASS3D and GraphFusion without pose refinement, DenseFusion and GraphFusion that perform iterative pose refinement are able to further improve the accuracy of the 6D pose. The effectiveness is further verified by Figure 6.5. In Figure 6.5 we compare the successful pose rate measured by ADD for 8 objects, which is obtained by varying the ADD threshold. As can be seen, DenseFusion and GraphFusion outperform other approaches by a large margin, especially when the threshold is small.

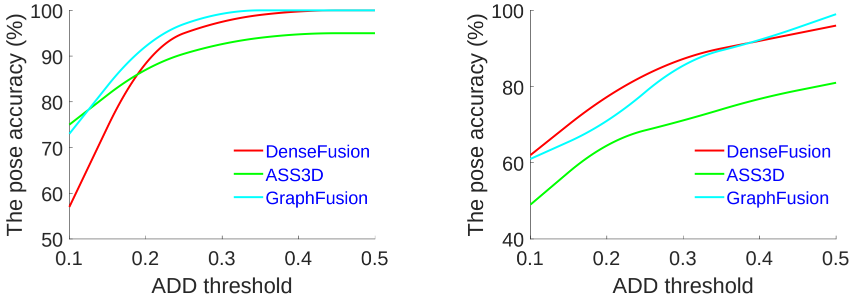
Effectiveness of multi-feature fusion. Apart from the successful pose rate generated by ADD, we calculate the successful pose rate by varying the reprojection error threshold, as shown in Figure 6.6. From Figure 6.6 we can see that GraphFusion is superior to other approaches, for the performance of ASS3D degrades significantly as the reprojection error decreases, while the performance of GraphFusion has a smaller decrease. It indicates that fusion mechanism considering the relationship between color and geometric features has a clear advantage over methods ignoring the correlation information between them.

Time efficiency and accuracy robustness of the single shot model. Compared with DenseFusion and GraphFusion, ASS3D estimates the 6D pose in a single and consecutive network. It runs faster than other approaches, as shown in Table 6.3 which compares the time efficiency among different methods. In particular, ASS3D runs at least 4 times faster than GraphFusion. Besides, for the texture-less objects such as milk box, ASS3D is more robust and has a better performance as shown in Figure 6.5.



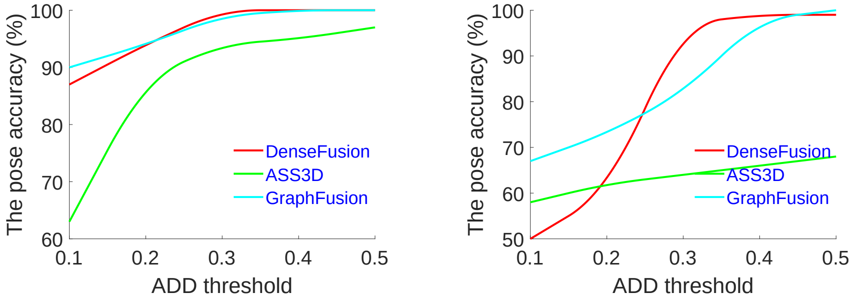
(a) *Banana*

(b) *Biscuit_box*



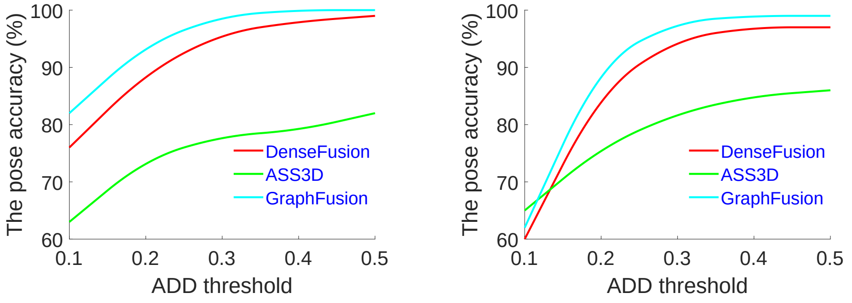
(c) *Chips_can*

(d) *Cookie_box*



(e) *Gingerbread_box*

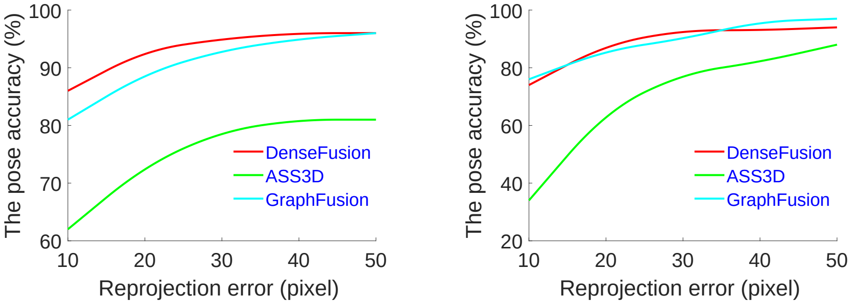
(f) *Milk_box*



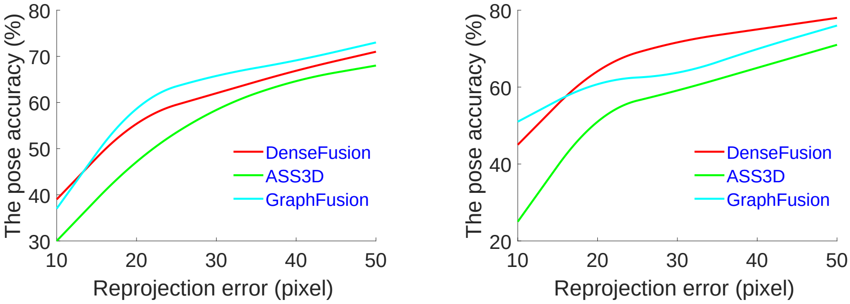
(g) *Pasta_box*

(h) *vacuum_cup*

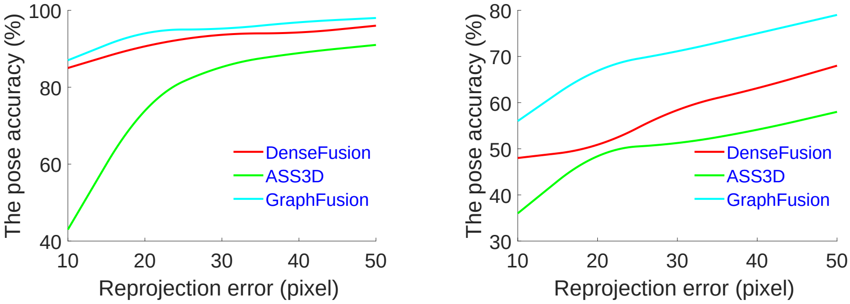
FIGURE 6.5: *The success rate of pose estimation in terms of ADD.*



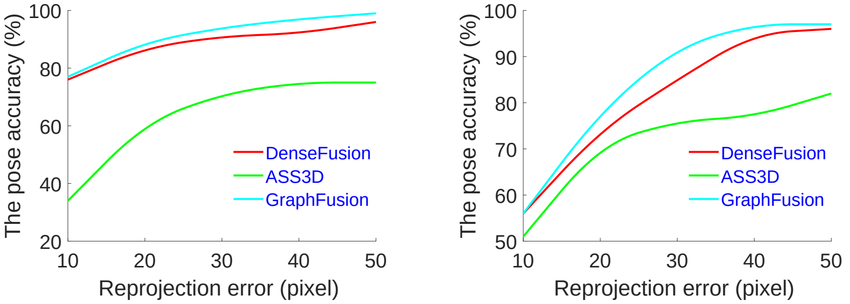
(a) *Banana* (b) *Biscuit_box*



(c) *Chips_can* (d) *Cookie_box*



(e) *Gingerbread_box* (f) *Milk_box*

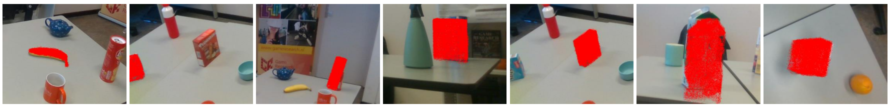


(g) *Pasta_box* (h) *Vacuum_cup*

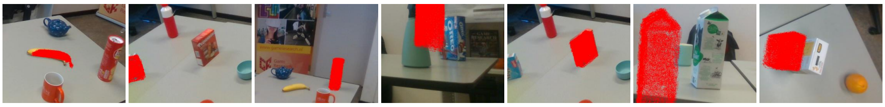
FIGURE 6.6: *The success rate of pose estimation in terms of reprojection errors.*

TABLE 6.3: *Comparison of the computational run time among different approaches (second per frame).*

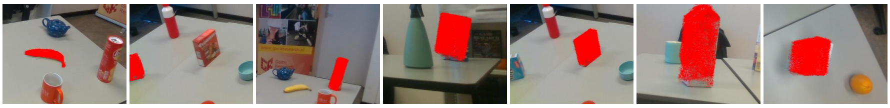
	DenseFusin	ASS3D	GraphFusion
banana	0.03	**0.01**	0.04
biscuit_box	0.03	**0.01**	0.04
chips_can	0.03	**0.01**	0.04
cookie_box	0.03	**0.01**	0.04
gingerbread_box	0.03	**0.01**	0.04
milk_box	0.03	**0.01**	0.04
pasta_box	0.03	**0.01**	0.04
vacuum_cup	0.03	**0.01**	0.04
MEAN	0.03	**0.01**	0.04



(a) *DenseFusion*



(b) *ASS3D*



(c) *GraphFusion3D*

FIGURE 6.7: *Examples of accuracy performance. Each 3D model is projected to the image plane with the estimated 6D pose.*

Furthermore, in Figure 6.7 we also visualize the comparison results. It can be seen that DenseFusion, ASS3D and GraphFusion provide more accurate 6D pose for colorful objects, such as banana, gingerbread box and chips can, while these approaches are less robust against dark color or low texture objects, such as the cookie box and milk box.

6.3.8 SUMMARY

With our benchmark, we have captured some state-of-the-art 6D object pose estimation approaches and systematically measured the progress of pose estimation research. As open problems, our analysis takes varying texture and shape objects, occlusion and object symmetries into consideration. The evaluation results indicate that the approach fully exploiting color and depth features performs best, outperforming pixel fusion based method and the approach with multimodal supervision. Besides, the single shot model outperforms other approaches in terms of time efficiency.

These insights are able to be used as a variety of purposes. For example, we can focus our pose estimation research efforts particularly on the more challenging objects and scenarios. It would be interesting to develop novel methods to find better feature representations for estimating poses of these challenging objects. Apart from that, we can focus our dataset collection efforts. For example, the synthetic dataset needs to be expanded by adding more reflective objects, occlusion, varying lighting conditions. Besides, more accurate depth images and 3D models need to be provided.

6.4 OBJECT POSE ESTIMATION WITH COLOR/GEOMETRY ATTENTION FUSION

6.4.1 OVERVIEW

Our goal is to achieve accurate 6D pose estimation for objects with a variety of sizes, shapes and textures using RGB-D images as input. Handcrafted features such as SIFT [NH03] and ORB [MAMT15] are key factors for classical methods to estimate 6D poses. However, these features are not sufficient to obtain precise poses for texture-less objects. Instead of designing new approaches to improve the robustness of handcrafted features, we learn more robust features and semantic cues using deep learning-based models.

We first apply a CNN-based encoder-decoder architecture to learn features from color images. Rather than directly extracting geometric features from the depth image, we first convert it to a point cloud which contains rich and scalable geometric information using the camera intrinsic matrix. To process point cloud data, previous methods first convert it into regular grids by projecting 3D data into 2D images or splitting 3D data into 3D voxel grids. Then they process the transformed data using approaches designed for regular data. Other methods directly extract features from raw point clouds by processing each point independently.

However, all the above approaches either introduce quantization artifacts or result in missing local features. To overcome these limitations, we use a graph neural network (GNN) to aggregate feature information from input data. GNN is especially suitable for data lying on irregular or non-Euclidean domains, such as point clouds. It has

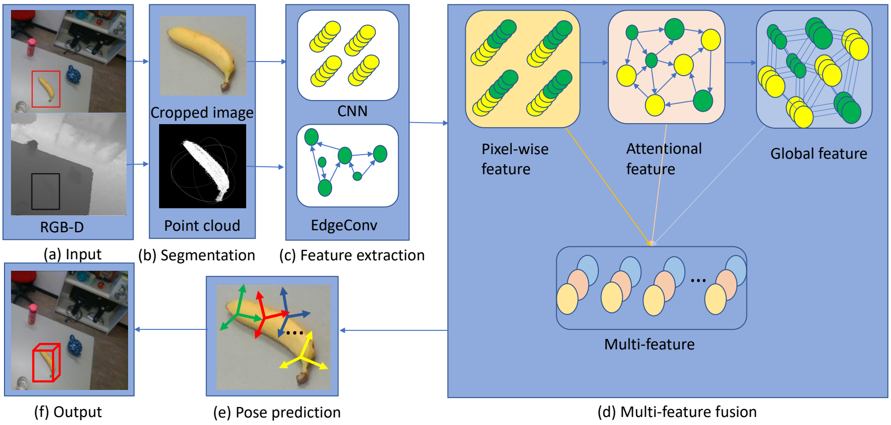


FIGURE 6.8: *Overview of our architecture: (a) The input are captured color-and-depth image pairs. (b) A semantic segmentation architecture is used to segment the target object. (c) The visual features are extracted by a CNN-based network and geometric representations are computed from the point cloud converted by the depth image. (d) A graph attention network is introduced to perform the fusion between color and geometric features. (e) The 6D object pose and its corresponding confidence score are predicted by the fused features. (f) The pose with the highest confidence is chosen as the final pose.*

been successfully applied in many areas, including semantic segmentation [LLS+17] and physics systems [WZW+17a]. To effectively extract geometric information, we use graph based edge convolution to process the point cloud. Edge convolution [WSL+19] generates edge features that describe the relationships between a point and its neighbors. It avoids the fundamental limitation that leads to loss of local features produced by previous approaches.

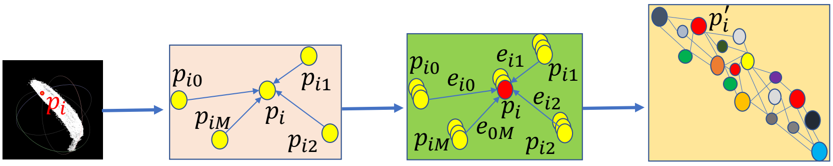
For high-precision pose estimation, fusing features is crucial to estimate accurate 6D poses. To effectively fuse features, we introduce a graph self-attention based framework (GAT) to exploit relationship between visual and geometric features, as opposed to prior works which just concatenates these features. Attention mechanisms have been used together with many neural network architectures that operate on regular and graph-structured data. Self-attention which is used to compute representations of sequence-based data attracts many attentions. It has been applied successfully in tasks such as machine translation [LPM15] and object detection [LLS+17] on Euclidean domains.

Figure 6.8 shows our overall framework. We first perform semantic segmentation to extract the target object from color-and-depth image pairs (Figure 6.8(b), Section 6.4.2). Next, we extract color and geometric features, separately, retaining the native structure of each data (Figure 6.8(c), Section 6.4.2). We apply the CNN-based network to aggregate visual information in the color image. To extract local and global geometric features from the depth image, we first convert the depth image to a point cloud and then build the local graph map for each point with the k-nearest neighbors algorithm (kNN). After that, the geometric features are computed by edge convolution on each local graph map. Furthermore, we fuse visual and geometric features with the GAT (Figure 6.8(d), Section 6.4.3). Finally, we train the network to predict the 6D pose for chosen pixels and then apply an iterative refinement method to estimate the final pose (Figure 6.8(e), Section 6.4.4).

6.4.2 SEMANTIC SEGMENTATION AND FEATURE EXTRACTION

Semantic segmentation. We detect objects in the input image using semantic segmentation from [XSNF17]. It generates a per-pixel segmentation map which classifies each pixel into a known object class. From the segmentation map, we get a 2D bounding box for the target object, and then we use the 2D bounding box to crop the input color-and-depth image pairs.

Feature extraction. In order to effectively extract information from color and depth images, we process the cropped color-and-depth image pairs separately. This is because the color data can be represented in a grid-like structure, while the geometric information residing in the depth image is defined in a continuous vector space. The cropped color image is fed into a CNN-based encoder-decoder architecture to extract visual information. Specially, given a color image of size $H \times W \times 3$, the network generates a feature image of size $H \times W \times d_{rgb}$ which contains the d_{rgb}-dimensional hidden representation of each pixel in the color image.



(a) Point cloud $p_i \in R^{N \times F}$ (b) Graph construction $p_i = \{p_{i0}, p_{i1}, \dots p_{iM}\}$ (c) Edge convolution using MLP (d) Geometric features $p_i' \in R^{N \times F'}$

FIGURE 6.9: *Geometric feature extraction: (a) The input is a point cloud converted from the depth image. (b) Each point of the point cloud is clustered by kNN to produce graph maps. (c) Geometric information is extracted by edge convolution. (d) A new set of features are produced as the output.*

To extract features from the depth image, we first project the cropped depth image to a point cloud based on the camera intrinsic matrix. For a 3D point cloud, features learned from each point are able to encode the neighboring geometric structure of each point. However, such features suffer from sensor noises. In contrast, multiple point-pair features are robust to occlusion and noises. To take advantage of the point-pair feature, we build a graph map for each point using the kNN (Figure 6.9(b)). Then we use edge convolution [WSL+19] that applies convolution-like operations on the local graph to extract features. The extracted features can effectively describe the relationship between a point and its neighbors (Figure 6.9(c)). The geometric feature extraction process is described as follows.

The input to the edge convolution layer is a local graph with M points, denoted by $p_i = \{p_{i0}, p_{i1}, ..., p_{iM}\}, p_i \in R^{N\times F}$, N is the number of points in the point cloud, F is the dimension of each point. The edge feature is defined as $e_{i,j} = f(p_i, p_j - p_i)$, where $f : R^F \times R^F \to R^{F'}$ is parametric non-linear function parameterized by a learnable weight matrix. F' is the new dimension of each point. We compute the edge feature by applying a multi-layer perceptron (MLP). After that, we get the output, $p_i' \in R^{N\times F'}$ shown in Figure 6.9(d).

6.4.3 MULTI-FEATURE FUSION

Fusing color and geometric features is important for 6D pose estimation. Concatenating color and geometric features directly is not able to effectively exploit the relationships between these features for more accurate 6D pose estimation. The key idea of our multi-feature fusion is to apply GAT to compute the hidden representations of each feature by attending over its neighbors. Unlike GraphFusion proposed in Section 6.3.5, our network has more attention layers and our fusion module is able to generate more semantic cues.

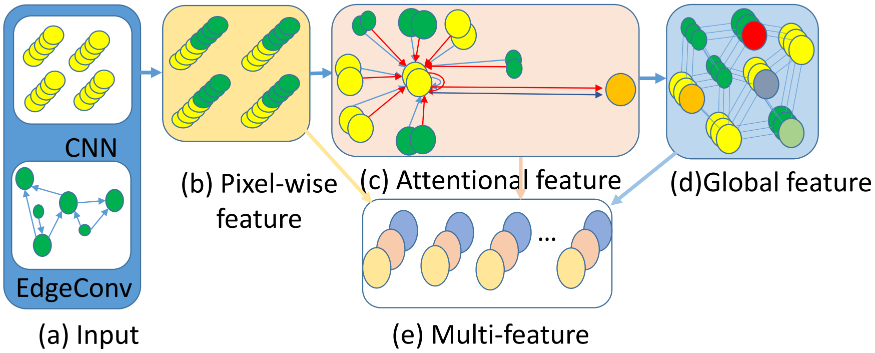


FIGURE 6.10: *Multiple feature fusion: (a) Color and geometric features are inputs. (b) We combine these two types of features to generate pixel-wise features. (c) The GAT is applied to compute attentional features. (d) The global features are generated by attentional features. (e) Features generated from (b), (c) and (d) are concatenated to produce multiple features.*

Our multiple feature fusion procedure first concatenates the per-pixel color and per-point geometric features as node features (6.10(b)) and then feeds them to the GAT. The graph attention layer updates the node features based on the other nodes. Concretely, the input nodes are firstly transformed by a linear transformation, parameterized by a weight matrix, $H \in R^{D' \times D}$, to achieve a higher representation. D is the number of input features in a node and D' is the number of new features in a node. Then a shared attention mechanism $GA : R^{D'} \times R^{D'} \rightarrow R$ is applied to the transformed node to compute attention coefficients $e_{i,j} = GA(Hx_i, Hx_j)$. The coefficient represents the importance of node j's features to node i.

In our experiments, we use a single-layer feedforward neutral network as the attention mechanism GA, parameterized by a set of learnable parameters:

$$ga = ga_1, ga_2, ..., ga_N, ga_i \in R^{2D'}, \tag{6.6}$$

where N is the number of nodes. The LeakyReLU is used as the activation function and the softmax function is used to normalize the attention coefficients. The attention mechanism is expressed as:

$$ga_{ij} = softmax_j(LeakyReLU(ga^T[Hx_i||Hx_j])), \tag{6.7}$$

where T is the transposition and $||$ is the concatenation operation.

After obtaining the learnable parameter for each node, it is multiplied with the node features by a nonlinearity, Θ, to produce the output node features. In order to obtain stable features, K attention mechanisms are implemented. Next, all the output features are concatenated:

$$x'_j = ||_{k=0}^{K} \Theta(\sum_{j \in N} ga_{ij}^k H^k x_j])), \tag{6.8}$$

where the attention parameter, ga_{ij}^k, is computed by the k-th attention mechanism (ga^k), and H^k is the corresponding transformation matrix applied to the input node. In our experiment, K is set to be two, as shown in Figure 6.10(c).

The resulting attentional features are fed into a CNN-based network to generate a global feature vector using maxing-pooling reduction function (Figure 6.10(d)). Finally, we concatenate pixel-wise, attentional and global features together as our multi-fusion features(6.10(e)).

6.4.4 POSE ESTIMATION AND REFINEMENT

Pose estimation. We predict the pixel-wise 6D object pose with our fusion features by MLP. We also predict a confidence score for the corresponding pose. It indicates the possibility of the corresponding pose to be the best pose. During inference, we choose the pose with the highest score as the final predicted pose. The loss function is defined based on the Euclidean distance between points transformed by ground truth pose and those transformed by predicted pose:

$$l_i = \frac{1}{N} \sum_{j \in N} ||(Rx_j + t) - (R'_i x_j + t'_i||), \tag{6.9}$$

where $[R'_i|t'_i], i \in N$, and $[R|t]$ are the estimated and ground truth 6D pose respectively, x_j is the selected point from the 3D model and N is the number of selected points.

Pose refinement. The performance of 6D pose estimation can be further improved by iterative refinement. We adopt the refinement module from [WXZ+19] to improve the pose prediction. Concretely, the input of this step are color features computed from the cropped color image and geometric features computed from the new point cloud transformed by the predicted 6D pose. The idea behind this transformation is that the transformed point cloud implicitly encodes the predicted pose. Then the two kinds of features are fused and fed into the refinement network to predict a residual pose. The final pose is obtained by M iterations:

$$RT = [R_M|t_M].[R_{M-1}|t_{M-1}]...[R_0|t_0]. \tag{6.10}$$

We can train the pose refinement network and the main network together. In order to reduce the training time, we start the refinement network after the main network has converged.

6.4.5 EXPERIMENTAL RESULTS

6.4.5.1 SETTINGS

Datasets. We use three datasets including our own dataset (RobotP), LineMOD dataset and YCB-Video datasets to evaluate our method. In our own dataset [dat], there are eight objects covering a variety of shapes, sizes and textures. LineMOD contains 13 textureless objects and YCB-Video dataset has 21 objects of varying shapes and textures. We follow the same training and testing settings as prior learning based approaches [TSF18, WXZ+19].

Implementation Details. We use the CNN-based network Resnet-18 [HZRS16] encoder followed by 4 up-sampling layers as decoder to extract color features. The edge convolution is a MLP with the number of layer neurons defined as $\{3, 64, 64, 64\}$. The graph is constructed using $k = 10$ the nearest neighbors. A single-layer GAT model with two attention heads is used for the feature fusion. We implement the networks within the PyTorch framework, train our model using Adam optimizer and set the learning rate to 0.0001. Furthermore, we refine the pose predicted from the main work with 2 iterations.

6.4.5.2 OVERALL PERFORMANCE

The overall performance compared with other state-of-the-art approaches is shown in Table 6.4, Table 6.5 and Table 6.6. We use ADD(-S), including ADD for non-symmetric objects and ADD-S for symmetric objects, to measure the prediction. If ADD-S is smaller than $2cm$ which is the minimum tolerance for robot grippers, the predicted pose is considered to be correct. The evaluation results compared with PoseCNN and Densefusion in terms of ADD-(S) and AUC on the YCB-Video dataset are shown in Table 6.4. In Table

6.5, we compare the percentage of ADD-(s) ($< 2cm$) with those of SSD-6D [KMT+17], Implicit+ICP [SMD+18] and DenseFusion [WXZ+19] on the LineMOD dataset.

TABLE 6.4: *Quantitative evaluation of the 6D pose (ADD(-S)) on the YCB-Video dataset (objects with bold name are symmetric).*

	PoseCNN		DenseFusion		C/G-AF	
	AUC	ADD(-S) <2 cm	AUC	ADD(-S) <2 cm	AUC	ADD(-S) <2 cm
002_master_chef_can	68.1	51.1	73.3	72.3	**88.2**	**88.9**
003_cracker_box	83.4	73.3	**94.2**	**98.2**	93.5	94.2
004_sugar_box	**97.5**	99.5	96.5	**100.0**	96.7	**100.0**
005_tomato_soup_can	81.8	76.6	85.5	83.0	**93.0**	**95.8**
006_mustard_bottle	**98.0**	98.6	94.7	96.1	95.1	**98.9**
007_tuna_fish_can	83.9	72.1	81.9	62.2	**89.2**	**85.7**
008_pudding_box	**96.6**	**100.0**	93.2	98.6	95.6	99.3
009_gelatin_box	**98.1**	**100.0**	96.7	**100.0**	**98.1**	**100.0**
010_potted_meat_can	83.5	77.9	83.6	79.9	**87.5**	**84.6**
011_banana	91.9	88.1	83.7	88.4	**92.1**	**97.9**
019_pitcher_base	**96.9**	97.7	**96.9**	**100.0**	95.9	**100.0**
021_bleach_cleanser	**92.5**	**92.7**	89.7	90.8	90.0	89.5
024_bowl	81.0	54.9	89.5	95.1	**89.9**	**96.7**
025_mug	81.1	55.2	88.9	88.8	**93.5**	**97.1**
035_power_drill	**97.7**	92.2	92.7	**96.5**	89.9	91.1
036_wood_block	87.6	80.2	92.8	**100.0**	**93.4**	98.2
037_scissors	78.4	49.2	77.5	48.6	**91.9**	**89.3**
040_large_marker	85.3	87.2	93.0	**100.0**	**94.7**	99.8
051_large_clamp	75.2	74.9	72.5	**78.7**	**75.0**	78.2
052_extra_large_clamp	64.4	48.8	69.9	74.9	**73.9**	**76.8**
061_foam_brick	97.2	**100.0**	91.9	**100.0**	94.1	**100.0**
Mean	86.6	79.9	87.6	88.2	**91.0**	**93.4**

TABLE 6.5: *Quantitative evaluation of the 6D pose (ADD(-S)) on the LineMOD dataset (objects with bold name are symmetric).*

	ape	ben.	cam	can	cat	driller	duck	**eggbox**	**glue**	hole.	iron	lamp	phone	MEAN
DenseFusion	92.3	93.2	94.4	93.1	96.5	87.7	92.3	99.8	**100.0**	92.1	97.0	95.3	92.8	94.3
SSD-6D	65.0	80.0	78.0	86.0	70.0	73.0	66.0	**100.0**	**100.0**	49.0	78.0	73.0	79.0	77.0
Implicit+ICP	20.6	64.3	63.2	76.1	72.0	41.6	32.4	98.6	96.4	49.9	63.1	91.7	71.0	64.7
C/G-AF	**96.3**	**97.4**	**97.8**	**97.6**	**98.5**	**96.8**	**97.4**	99.8	**100.0**	**95.3**	**97.3**	**98.8**	**98.6**	**97.8**

We can see that our proposed approach applying color/geometry attention fusion (C/G-AF) achieves the best performance on both datasets. It is also verified by Table

TABLE 6.6: *Quantitative evaluation of the 6D pose (ADD and ADD-S).*

	DenseFusion		ASS3D		GraphFusion		C/G-AF	
	ADD	ADD-S	ADD	ADD-S	ADD	ADD-S	ADD	ADD-S
banana	**0.86**	0.86	0.70	0.75	0.83	0.87	0.85	**0.91**
biscuit_box	0.91	0.95	0.78	0.88	0.93	0.96	**0.94**	**0.98**
chips_can	0.56	0.94	**0.75**	0.85	0.69	**0.97**	0.74	**0.97**
cookie_box	**0.62**	0.74	0.49	0.66	0.61	0.75	**0.62**	**0.79**
gingerbread_box	0.87	0.94	0.63	0.86	0.90	0.95	**0.92**	**0.96**
milk_box	0.50	**0.81**	0.58	0.62	0.66	0.77	**0.69**	0.80
pasta_box	0.77	0.91	0.63	0.72	0.84	0.96	**0.86**	**0.97**
vacuum_cup	0.61	0.90	**0.65**	0.75	0.63	0.97	**0.65**	**0.98**
MEAN	0.71	0.88	0.65	0.76	0.76	0.90	**0.78**	**0.92**

6.6 which shows the comparison results with benchmarking methods. These results demonstrate that our fusion strategy is superior to those that do not exploit the relationship between color and geometric features or do not effectively extract geometric features. For the LineMOD dataset, our method outperforms Implicit+ICP and DenseFusion 33.1% and 3.5%, respectively.

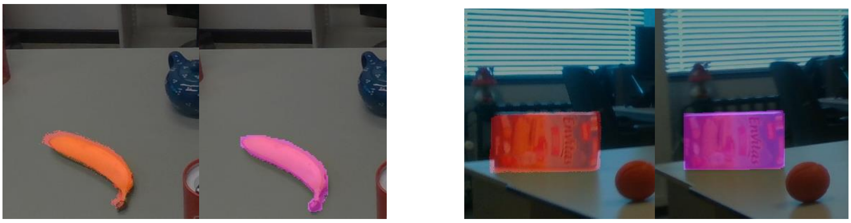
Furthermore, in Figure 6.11 we also visualize the comparison results between DenseFusion and our method on our own new dataset (RobotP). The overlap is generated by projecting 3D object models to the image plane with the estimated 6D poses. We can see that the overlap generated by our approach is larger than DenseFusion, which indicates our method is more robust against occlusion and low texture objects.

6.4.5.3 ABLATION STUDY

To verify the effectiveness of each module of our proposed network, we perform the ablation study on the LineMOD and RobotP datasets.

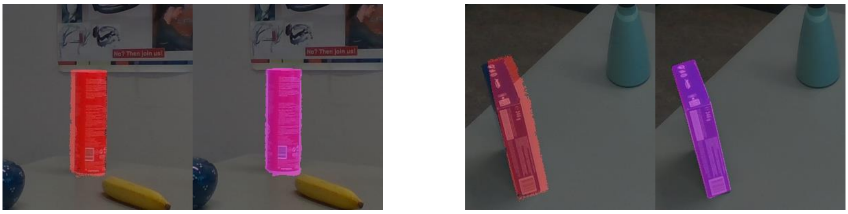
Effectiveness of geometric feature extraction. By varying the reprojection error threshold on our RobotP dataset, we plot the accuracy-threshold curves, as shown in Figure 6.12. It can be seen that our method using geometric features extracted from point pairs outperforms the approach which extracts geometric features by processing each point separately by a large margin, especially for low texture objects, such as the milk box.

Effectiveness of multi-feature fusion. Table 6.7 summarizes the comparison results with and without graph attention mechanism in terms of ADD(-S). As can be seen, compared with DenseFusion, the graph attention mechanism increases the accuracy of estimated poses significantly (8.7%), and our approach predicts more accurate poses for symmetric objects, like glue. Compared with our method without multi-feature fusion module, the performance is also increased by a large margin (5.9%).



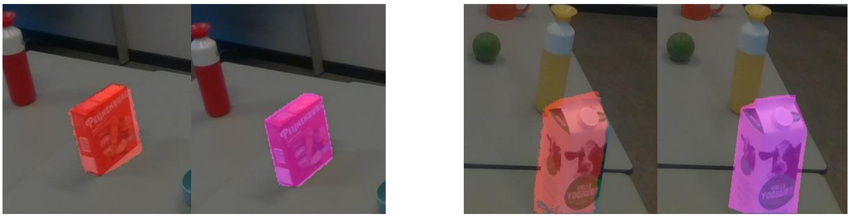
(a) *Banana*

(b) *Biscuit_box*



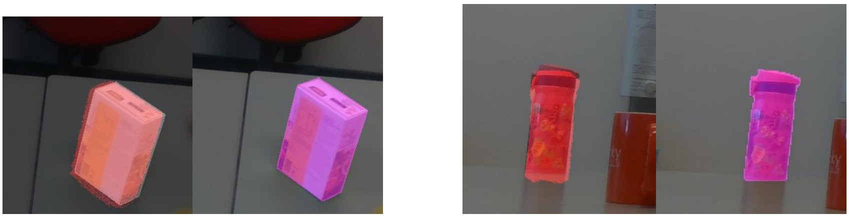
(c) *Chips_can*

(d) *Cookie_box*



(e) *Gingerbread_box*

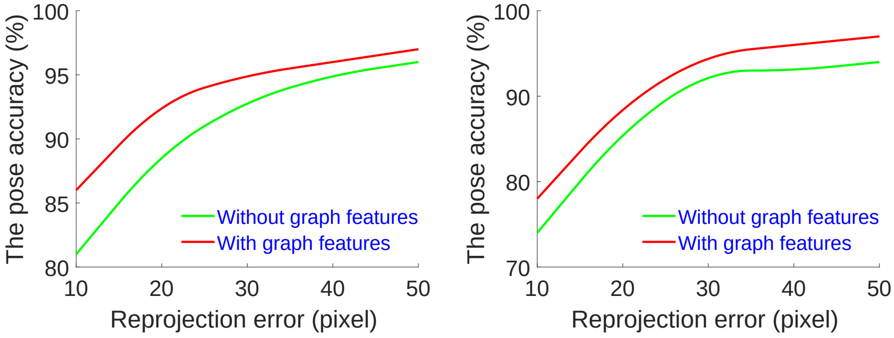
(f) *Milk_box*



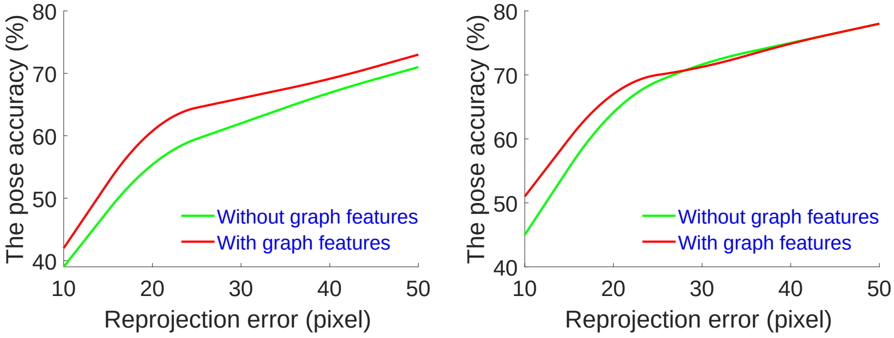
(g) *Pasta_box*

(h) *vacuum_cup*

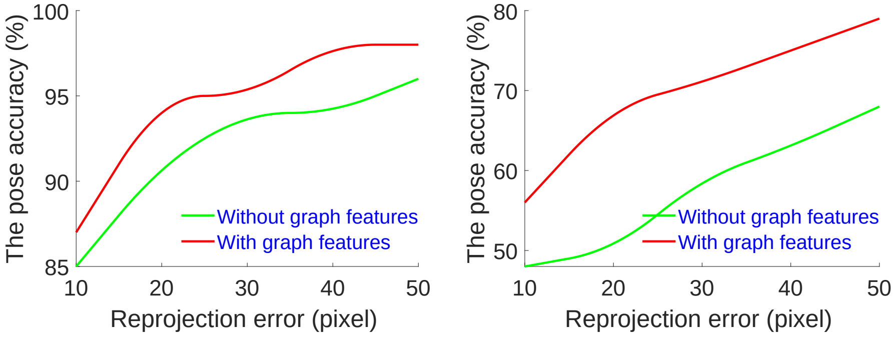
FIGURE 6.11: *Examples of accuracy differences between DenseFusion and our approach on RobotP dataset. The first image is the result of DenseFusion and the second image is the result of our approach. Each 3D model is projected to the image plane with the estimated 6D poses to generate the overlap.*



(a) *Banana* (b) *Biscuit_box*



(c) *Chips_can* (d) *Cookie_box*



(e) *Gingerbread_box* (f) *Milk_box*

FIGURE 6.12: *The accuracy-threshold curves (AUCs) generated by reprojection error on RobotR dataset.*

TABLE 6.7: *Accuracy comparison with and without attention in terms of ADD(-S) on the LineMOD dataset (objects with bold name are symmetric).*

	ape	ben.	cam	can	cat	driller	duck	**eggbox**	**glue**	hole.	iron	lamp	phone	MEAN
DenseFusion	80.5	83.2	78.5	88.1	89.3	79.9	77.9	99.6	99.1	79.5	93.2	93.5	89.2	87.1
C/G-AF(w/o attention)	85.3	88.5	88.7	88.9	90.1	87.5	89.3	98.2	98.9	84.9	88.6	90.4	89.9	89.9
C/G-AF	**93.8**	**95.4**	**95.6**	**95.8**	**96.5**	**92.7**	**95.7**	**99.7**	**100.0**	**91.3**	**95.9**	**96.6**	**96.8**	**95.8**

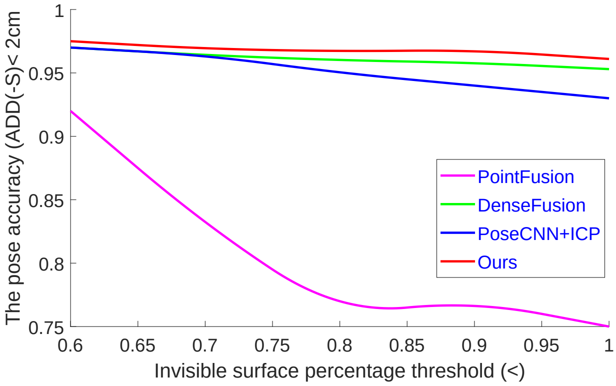


FIGURE 6.13: *Accuracy comparison with different degree of occlusion.*

Robustness against occlusion. In Figure 6.13 we compare our approach with DenseFusion, PointFusion and poseCNN + ICP in terms of the robustness against occlusion. We first calculate the visible surface ratio of each sampled object when the object is projected to the image plane. Then we calculate the number of successful predictions among all the testing frames. If the ADD(-S) is smaller than $2cm$, we consider the prediction is correct. In detail, we sample a set of points from the 3D object model and project these points to the image plane to synthesize a depth image using the ground truth 6D pose and camera intrinsic matrix. Next, we compare the pixel value in the synthesized depth image with the ground truth depth image. If the calculated pixel value is smaller than ground truth value, we consider its corresponding 3D point is invisible. This is because only the front-most pixels are shown in the depth image. After that, we calculate the number of invisible points from our sampled points and then obtain the invisible ratio. As shown in figure 6.13, our approach performs best among these methods. The increasing invisible ratio does not reduce our method's performance significantly, while the performance of PoseCNN degrades greatly.

6.5 CONCLUSION AND FUTURE WORK

The 6D object pose estimation is a challenging but important research direction for virtual reality, robotic grasping and visual navigation. With our benchmark, we have analyzed some state-of-the-art approaches in this field and summarized the success and failure modules of these algorithms. The evaluation results indicate that data-driven methods are the current trend in 6D object pose estimation and the approach fully exploiting the relationships between color and depth features performs best. This benchmark and comparative evaluation results have the potential to further enrich and boost the research of 6D object pose estimation and its applications.

We have proposed a novel 6D object pose estimation framework that first extracts discriminative color and geometric features from RGB-D images. We then fuse these features based on a graph attention mechanism to predict the object pose. Experimental results have demonstrated that the feature extraction and fusion modules can increase the overall accuracy of estimated 6D poses, and the proposed approach can be used for robotic grasping tasks. However, some limitations are worth noting. Although our method is robust to objects with different sizes and shapes, when the object is reflective and under the changing lighting condition, our method still fails to predict the accurate pose, as shown in Figure 6.14. It would be interesting to explore more efficient approaches to estimate the 6D pose of the object with reflective property under more complicated conditions in the future.

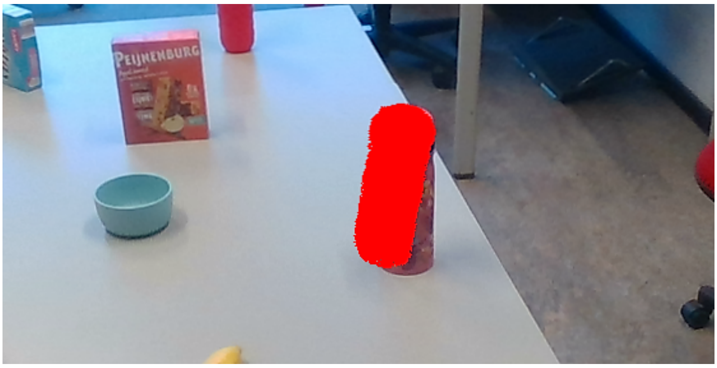


FIGURE 6.14: *The typical limitations of our approach: the estimated pose for the reflective object is not accurate, as the object projected by the predicted pose does not completely overlap onto the ground truth object. This is because it is hard to extract reliable features from reflective objects, which are used to estimate the pose.*

7

CONCLUSIONS AND FUTURE WORK

In this thesis, we explore methods to build a photo-realistic simulation environment for mobile robots and develop robotic vision tasks based on this simulator. While many recent simulators successfully provide physical environments by parameter settings of scene details, including geometry, texture and lighting, our focus is on producing photo-realistic simulated environments with a sparse set of RBG-D images to avoid precise modeling and time-consuming parameter settings. At the same time, we aim at a deeper understanding of developing 6D object pose estimation algorithms based on synthesized data.

7.1 REACHING OUR GOALS

In order to provide evidence for our work, we first revisit our goals from Chapter 1:

1. provide free-viewpoint photo-realistic rendering of real scenes, using a collection of RGB-D images,

2. allow developing robotics applications and seamlessly interfacing with Robot Operating System (ROS),

3. easily control the movement of robots, and provide real-time positions and whole trajectories of the moving robot, and a global 3D map,

4. generate representative datasets with rich data for training and evaluating algorithms designed for robotic vision tasks, and

5. enable robots to learn artificial intelligence algorithms (e.g., object recognition and pose estimation) in simulation and allow transforming knowledge learned from simulation to the real world without domain adaptation.

In Chapter 3, we focus on providing photo-realistic rendering (goal 1). A novel image based rendering approach is proposed to generate photo-realistic imagery of real scenes. We first introduce a pixel-to-pixel multi-view depth refinement method to produce pixel-accurate alignment between color-and-depth image pairs and correct inaccurate depth values in the depth image. Based on the refined depth images, we combine an adaptive view selection approach and layered 3D warping which warps images in different depth layers to lower the rendering complexity and improve the quality of synthesized image. The experimental results demonstrate that the proposed approach provides plausible and photo-realistic rendering on a variety of complex indoor scenes.

Based on the view synthesis algorithm proposed in Chapter 3, in Chapter 4 we design a 3D environment simulator (PreSim) for synthesizing free-viewpoint RGB-D views and developing robotic vision applications under ROS (goal 2 and 3). The main characteristics of PreSim are: (i) it provides a photo-realistic 3D environment which allows seamlessly integrating multisensory models in the virtual world and enables them to perceive and navigate scenes, (ii) it has an internal view synthesis module which allows transforming algorithms developed and tested in simulation to physical platforms without domain adaption, (iii) it can generate an infinite amount of data for vision-based applications, such as depth estimation and object pose estimation. We demonstrate three applications of this virtual environment and show that our simulator narrows the reality gap between the virtual environment and the real scene.

Taking advantage of our simulator described in Chapter 4, Chapter 5 proposes a representative dataset (RobotP) for various vision-based tasks, with a special focus on 6D object pose estimation (goal 4). The RobotP consists of extremely photo-realistic indoor scenes, and the objects in it cover a variety of shapes, rigidity, sizes, weight and textures. We use our simulator to mimic the physical behavior of the sensors and provide high-quality synthesized color-and-depth image pairs with ground truth 6D poses.

To obtain captured color-and-depth image pairs used for synthesizing new image and object modeling, we present an extensive analysis for objects and 3D camera selections, scenario design, and trajectory generation. As the accuracy of the 6D pose used for synthesizing new image plays an important role in high-quality view synthesis, we propose a pose refinement method to refine the poses estimated from SfM. To improve the quality of our captured depth images, we first introduce a depth alignment method to align the captured depth image to its corresponding color image. Then we propose a depth fusion approach to fuse the captured depth image and estimated depth image generated by multi-view stereo. Based on the captured data, we generate not only virtual images but also 3D modes for our chosen objects. From the 3D models, we also propose an algorithm to automatically and cost-effectively generate masks and corresponding 2D bounding boxes for the objects in our dataset.

Based on our dataset, we organize the Shape Retrieval Challenge benchmark on 6D pose estimation. It consists of two components: (1) an openly accessible dataset, and (2) an annual competition and corresponding workshop. The competition and workshop provide a way to measure and track the pose estimation progress and discuss the lessons learned from research groups. This benchmark and comparative evaluation results have the potential to further enrich and boost the research of 6D object pose estimation and its applications.

In Chapter 6, we further investigate how different pose estimation approaches perform in terms of various object properties, such as shapes, sizes, textures and occlusion using different evaluation metrics (goal 5). It gives us insight into the current state of the field of pose estimation. We learn important lessons from the current pose estimation algorithms: (1) approaches that fully exploit the color and geometric features are more robust for 6D pose estimation of reflective and texture-less objects and occlusion; (2) the single shot model outperforms other approaches in terms of time efficiency and is more robust to occlusion. We also investigate where researchers' attention should be paid to make progress and propose insights for designing the next generation of general 6D object pose estimation algorithms.

Apart from that, we propose a novel network to further improve the performance of 6D object pose estimation (goal 5). We first apply a feature extraction network which effectively extracting local and global geometric features from point clouds. Next, a new multi-feature fusion network is proposed to improve 6D pose prediction performance, which applies a graph attention network to fully exploit the relationship between visual and geometric features and compute hidden feature representations between these features. Experiments indicate that our approach is robust to handle heavy occlusion, low texture and sensor noise for 6D object pose estimation.

At the end of the day, the simulator and approaches presented in this thesis strike a compromise between our five goals. As discussed in above chapters, improvements can be made towards each goal without sacrificing the others. Besides, we believe our simulator does not yet provide enough functionality modules for robotic vision tasks. We discuss a few possible improvements below.

7.2 BEYOND OUR GOALS

7.2.1 PHOTO-REALISTIC RENDERING

Our view synthesis approaches provide photo-realistic rendering by blending layered color-and-depth image pairs. This works well when the depth image has high quality. However, generating high-quality depth images is challenging, especially when there are many texture-less, transparent and reflective objects (e.g., white walls, windows, light and mirrors) in the scene. This can be addressed with an approach that processes each special object separately and generates more accurate depth images.

7.2.2 FUNCTIONALITY MODULES

Even though our simulator provides different functionality modules for robotic vision tasks, it is only designed for static scenes which do not include dynamic contents, such as moving people and objects. A system that allows training robotic tasks such as autonomous navigation in dynamic environment or is able to simulate a variety of object changes would benefit the development of robotics applications. An approach that models different objects separately is likely to provide dynamic rendering of the scene. Besides, training a robot from scratch is extremely challenging in the complicated environment. A system that allows human participation in the simulated environment would facilitate the training process of the robotics research, for the robot could learn from demonstrations given by human users.

7.2.3 DATASETS

When collecting the dataset, we take many practical issues such as the cost and property of objects into consideration. However, in order to further improve the accuracy of vision-based algorithms, the dataset needs to be expanded by adding more challenging objects with various properties. Besides, more details about the objects including the mass, sizes and inertia of the objects would benefit a variety of motion planners and optimizers. They could use these models either as collision or manipulation objects.

For we collect the data by a hand-hold 3D depth camera, it requires patience and a steady hand to avoid blurring and noisy images. This makes capture tedious and labor-intensive, especially for good 3D modeling results. A system that allows capturing a much larger view of the scene would reduce the number of captured images. Apart from that, a system which is robust to motion blur and noise would enable users to capture images by simply waving a camera around in the scene without worrying about the quality of captured images.

7.2.4 6D OBJECT POSE ESTIMATION

Current algorithms including our method proposed in this thesis estimate the 6D object pose by extracting features from RGB-D images. This works well for colorful objects (e.g., the cookie box and chips can shown in Chapter 6). However, they are not able to provide accurate 6D poses for certain types of objects. For example, the reflective and texture-less objects tend to be particularly challenging. An approach that focuses on the challenging objects would further enrich and boost the research of 6D object pose estimation and its applications. Even though color and geometric features are extracted and fused for pose estimation, the extraction and fusion strategies are not fully explored, which would be interesting research direction.

7.3 CONCLUDING REMARKS

In this thesis, we present a 3D environment simulator for mobile robots and provide evidence that algorithms developed and tested in simulation are able to be transformed to the real world without domain adaption for many robotic vision tasks. Our simulator creates photo-realistic color-and-depth image pairs with ground truth poses, which can be used for vision-based tasks such as depth estimation, object recognition and 6D pose estimation. Based on our simulator, we generate a 3D object dataset and organize the Shape Retrieval Challenge benchmark on 6D pose estimation. It provides a way to measure and track the progress in 6D object pose estimation and discuss the lessons learned from research groups. We further improve the accuracy of the pose estimation by a graph attention network. There is plenty of interesting future work left to explore, including further improvements towards the goals we stated in Chapter 1 and research outside the scope of this thesis.

The proposed 3D environment simulator designed for mobile robots facilitates many robotics applications, such as object recognition, depth estimation, robotic grasping, obstacle avoidance and visual navigation. For example, as our simulator can generate an infinite amount of photo-realistic data, the deep learning-based visual AI algorithms (e.g., object recognition and robotic grasping algorithms) can be developed and tested in it. This makes it easier to take advantage of deep learning to further boost robot manipulation ability. Apart from that, our virtual environment provides a variety of information including color-and-depth image pairs, sensor poses and robot trajectories, which can be used to design visual navigation approaches. This provides the possibility for robots to learn to localize and navigate themselves to a target position automatically. In the near future, we may even see some of these applications become available to consumers and the goal that equips robots with most of human-like abilities is achieved.

CURRICULUM VITAE

Honglin Yuan was born in Shandong, China in 1990. She obtained her bachelor's in Automation from Wuhan Textile University. After graduating from Wuhan Textile University, she moved to Beijing to study Precision Instruments and Mechanology in Beihang University. After obtaining her master's degree, she joined in the Multimedia group at Utrecht University in 2017 to pursue a PhD degree under the supervision of Prof. Remco.C. Veltkamp. Her research focused on developing virtual reality environments and deep learning-based applications for mobile robots.

BIBLIOGRAPHY

[AB+91] Edward H Adelson, James R Bergen, et al. *The plenoptic function and the elements of early vision*, volume 2. MIT Press, 1991.

[ACD+09] Ery Arias-Castro, David L Donoho, et al. Does median filtering truly preserve edges better than linear filtering? *The Annals of Statistics*, 37(3):1172–1206, 2009.

[AW18] Ibraheem Alhashim and Peter Wonka. High quality monocular depth estimation via transfer learning. *arXiv preprint arXiv:1812.11941*, 2018.

[AWT+18] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould, and Anton van den Hengel. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3674–3683, 2018.

[BBM+01] Chris Buehler, Michael Bosse, Leonard McMillan, Steven Gortler, and Michael Cohen. Unstructured lumigraph rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 425–432, 2001.

[BCM18] Simone Bianco, Gianluigi Ciocca, and Davide Marelli. Evaluating the performance of structure from motion pipelines. *Journal of Imaging*, 4(8):98, 2018.

[BE01] Gabriel J Brostow and Irfan Essa. Image-based motion blur for stop motion animation. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 561–566, 2001.

[BK08] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, 2008.

[BKM$^+$14] Eric Brachmann, Alexander Krull, Frank Michel, Stefan Gumhold, Jamie Shotton, and Carsten Rother. Learning 6D object pose estimation using 3D object coordinates. In *European conference on computer vision*, pages 536–551. Springer, 2014.

[BNVB13] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[BPA$^+$17] Simon Brodeur, Ethan Perez, Ankesh Anand, Florian Golemo, Luca Celotti, Florian Strub, Jean Rouat, Hugo Larochelle, and Aaron Courville. Home: A household multimodal environment. *arXiv preprint arXiv:1711.11017*, 2017.

[BRR11] Michael Bleyer, Christoph Rhemann, and Carsten Rother. Patchmatch stereo-stereo matching with slanted support windows. In *Bmvc*, volume 11, pages 1–11, 2011.

[BSCB00] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. Image inpainting. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 417–424, 2000.

[BSFG09] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. Graph.*, 28(3):24, 2009.

[CCL$^+$05] Wan-Yu Chen, Yu-Lin Chang, Shyh-Feng Lin, Li-Fu Ding, and Liang-Gee Chen. Efficient depth image based rendering with edge dependent depth filter and interpolation. In *2005 IEEE International Conference on Multimedia and Expo*, pages 1314–1317. IEEE, 2005.

[CDF$^+$17] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3D: Learning from RGB-D data in indoor environments. *arXiv preprint arXiv:1709.06158*, 2017.

[CDLN07] Frederique Crete, Thierry Dolmiere, Patricia Ladret, and Marina Nicolas. The blur effect: perception and estimation with a new no-reference perceptual blur metric. In *Human vision and electronic imaging XII*, volume 6492, page 64920I. International Society for Optics and Photonics, 2007.

[CDSHD13] Gaurav Chaurasia, Sylvain Duchene, Olga Sorkine-Hornung, and George Drettakis. Depth synthesis and local warps for plausible image-based navigation. *ACM Transactions on Graphics (TOG)*, 32(3):1–12, 2013.

[Che95] Shenchang Eric Chen. Quicktime VR: An image-based approach to virtual environment navigation. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 29–38, 1995.

[CHJH02] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134(1-2):57–83, 2002.

[CT19] Chao Chen and Llewellyn Tang. Bim-based integrated management workflow design for schedule and cost planning of building fabric maintenance. *Automation in Construction*, 107:102944, 2019.

[CW93] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 279–288, 1993.

[dat] 3D object datasets. `https://yhldrf.github.io/Datasets.github.io/`. Accessed: 2020-05-31.

[DCLT18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[DDB$^+$15] Paul Debevec, Greg Downing, Mark Bolas, Hsuen-Yueh Peng, and Jules Urbach. Spherical light field environment capture for virtual reality using a motorized pan/tilt head and offset camera. In *ACM SIGGRAPH 2015 Posters*, pages 1–1. 2015.

[DFI$^+$15] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. FlowNet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2758–2766, 2015.

[DHH$^+$20] Matt Deitke, Winson Han, Alvaro Herrasti, Aniruddha Kembhavi, Eric Kolve, Roozbeh Mottaghi, Jordi Salvador, Dustin Schwenk, Eli VanderBilt, Matthew Wallingford, et al. RoboTHOR: An open simulation-to-real embodied AI platform. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3164–3174, 2020.

[DN17] Ji Dai and Truong Nguyen. View synthesis with hierarchical clustering based occlusion filling. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 1387–1391. IEEE, 2017.

[DP11] Amaël Delaunoy and Emmanuel Prados. Gradient flows for optimizing triangular mesh-based surfaces: Applications to 3D reconstruction problems dealing with visibility. *International journal of computer vision*, 95(2):100–123, 2011.

[dPMM06] Angel P del Pobil, Rad Madhavan, and Elena Messina. Benchmarks in robotics research. In *Workshop IROS*. Citeseer, 2006.

[DRC$^+$17] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. *arXiv preprint arXiv:1711.03938*, 2017.

[DS98] Norman R Draper and Harry Smith. *Applied regression analysis*, volume 326. John Wiley & Sons, 1998.

[DT05] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005.

[DUB$^+$17] Bertram Drost, Markus Ulrich, Paul Bergmann, Philipp Hartinger, and Carsten Steger. Introducing MVTec ITODD - A for 3D object recognition in industry. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 2200–2208, 2017.

[ED04] Elmar Eisemann and Frédo Durand. Flash photography enhancement via intrinsic relighting. *ACM transactions on graphics (TOG)*, 23(3):673–678, 2004.

[EdDM$^+$08] Martin Eisemann, Bert de Decker, Marcus A. Magnor, Philippe Bekaert, Edilson de Aguiar, Naveed Ahmed, Christian Theobalt, and Anita Sellent. Floating textures. *Comput. Graph. Forum*, 27(2):409–418, 2008.

[EVGW$^+$10] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (VOC) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[FBAS16] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. RotorS—a modular Gazebo MAV simulator framework. In *Robot Operating System (ROS)*, pages 595–625. Springer, 2016.

[Feh04] Christoph Fehn. Depth-image-based rendering (DIBR), compression, and transmission for a new approach on 3D-TV. In *Stereoscopic Displays and Virtual Reality Systems XI*, volume 5291, pages 93–105. International Society for Optics and Photonics, 2004.

[FGW$^+$18] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. Deep ordinal regression network for monocular depth estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2002–2011, 2018.

[FH15] Yasutaka Furukawa and Carlos Hernández. Multi-view stereo: A tutorial. *Foundations and Trends® in Computer Graphics and Vision*, 9(1-2):1–148, 2015.

[FHT01] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[FIS14] Dario Floreano, Auke Jan Ijspeert, and Stefan Schaal. Robotics and neuroscience. *Current Biology*, 24(18):R910–R920, 2014.

[FNPS16] John Flynn, Ivan Neulander, James Philbin, and Noah Snavely. Deepstereo: Learning to predict new views from the world's imagery. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5515–5524, 2016.

[FP09] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE transactions on pattern analysis and machine intelligence*, 32(8):1362–1376, 2009.

[GAF$^+$10] Michael Goesele, Jens Ackermann, Simon Fuhrmann, Carsten Haubold, Ronny Klowsky, Drew Steedly, and Richard Szeliski. Ambient point clouds for view interpolation. In *ACM Transactions on Graphics (TOG)*, volume 29, page 95. ACM, 2010.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[Ger03] Deborah Levine Gera. *Ancient Greek ideas on speech, language, and civilization*. Oxford University Press, USA, 2003.

[GGS$^+$19] Xiaofeng Gao, Ran Gong, Tianmin Shu, Xu Xie, Shu Wang, and Song-Chun Zhu. Vrkitchen: an interactive 3D virtual environment for task-oriented learning. *arXiv preprint arXiv:1903.05757*, 2019.

[GGSC96] Steven J Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F Cohen. The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54, 1996.

[GHTC03] Xiao-Shan Gao, Xiao-Rong Hou, Jianliang Tang, and Hang-Fei Cheng. Complete solution classification for the perspective-three-point problem. *IEEE transactions on pattern analysis and machine intelligence*, 25(8):930–943, 2003.

[Gir15] Ross Girshick. Fast R-CNN. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[GIRL03] Natasha Gelfand, Leslie Ikemoto, Szymon Rusinkiewicz, and Marc Levoy. Geometrically stable sampling for the ICP algorithm. In *Fourth International Conference on 3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings.*, pages 260–267. IEEE, 2003.

[GLS15] Silvano Galliani, Katrin Lasinger, and Konrad Schindler. Massively parallel multiview stereopsis by surface normal diffusion. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 873–881, 2015.

[GLSU13] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The KITTI dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[GLU12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE, 2012.

[GM04] Mike Goslin and Mark R Mine. The panda3D graphics engine. *Computer*, 37(10):112–114, 2004.

[GR71] Gene H Golub and Christian Reinsch. Singular value decomposition and least squares solutions. In *Linear Algebra*, pages 134–151. Springer, 1971.

[GS05] Pau Gargallo and Peter Sturm. Bayesian 3D modeling from images using multiple depth maps. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 885–891. IEEE, 2005.

[GVS18] Silvio Giancola, Matteo Valenti, and Remo Sala. *A survey on 3D cameras: Metrological comparison of time-of-flight, structured-light and active stereoscopy technologies*. Springer, 2018.

[GZD$^+$20] Albanis Georgios, Nikolaos Zioulis, Anastasios Dimou, Dimitris Zarpalas, and Petros Daras. Dronepose: Photorealistic uav-assistant dataset synthesis for 3D pose estimation via a smooth silhouette loss. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, August 2020.

[Har97] Richard I Hartley. In defense of the eight-point algorithm. *IEEE Transactions on pattern analysis and machine intelligence*, 19(6):580–593, 1997.

[HDS$^+$20] David Hall, Feras Dayoub, John Skinner, Haoyang Zhang, Dimity Miller, Peter Corke, Gustavo Carneiro, Anelia Angelova, and Niko Sünderhauf. Probabilistic object detection: Definition and evaluation. In *The IEEE*

Winter Conference on Applications of Computer Vision, pages 1031–1040, 2020.

[HHC$^+$11] Stefan Hinterstoisser, Stefan Holzer, Cedric Cagniart, Slobodan Ilic, Kurt Konolige, Nassir Navab, and Vincent Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *2011 international conference on computer vision*, pages 858–865. IEEE, 2011.

[Hir06] Heiko Hirschmuller. Stereo vision in structured environments by consistent semi-global matching. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 2386–2393. IEEE, 2006.

[HLI$^+$12] Stefan Hinterstoisser, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary Bradski, Kurt Konolige, and Nassir Navab. Model based training, detection and pose estimation of texture-less 3D objects in heavily cluttered scenes. In *Asian conference on computer vision*, pages 548–562. Springer, 2012.

[HMB$^+$18] Tomas Hodan, Frank Michel, Eric Brachmann, Wadim Kehl, Anders GlentBuch, Dirk Kraft, Bertram Drost, Joel Vidal, Stephan Ihrke, Xenophon Zabulis, et al. BOP: benchmark for 6D object pose estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[HMBLM08] Gary B Huang, Marwan Mattar, Tamara Berg, and Eric Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. University of Massachusetts, 2008.

[HPP$^+$18] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM Transactions on Graphics (TOG)*, 37(6):1–15, 2018.

[HRDB16] Peter Hedman, Tobias Ritschel, George Drettakis, and Gabriel Brostow. Scalable inside-out image-based rendering. *ACM Transactions on Graphics (TOG)*, 35(6):1–11, 2016.

[HST12] Kaiming He, Jian Sun, and Xiaoou Tang. Guided image filtering. *IEEE transactions on pattern analysis and machine intelligence*, 35(6):1397–1409, 2012.

[HZ03] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[Int] Intel RealSense Depth Camera D415. https://www.intelrealsense.com/depth-camera-d415/. Accessed: 2020-06-18.

[IZU13] Umit Isikdag, Sisi Zlatanova, and Jason Underwood. A bim-oriented model for supporting indoor navigation requirements. *Computers, Environment and Urban Systems*, 41:112–123, 2013.

[JHHB16] Matthew Johnson, Katja Hofmann, Tim Hutton, and David Bignell. The Malmo platform for artificial intelligence experimentation. In *IJCAI*, pages 4246–4247, 2016.

[JMP$^+$18] Omid Hosseini Jafari, Siva Karthik Mustikovela, Karl Pertsch, Eric Brachmann, and Carsten Rother. iPose: instance-aware 6D pose estimation of partly occluded objects. In *Asian Conference on Computer Vision*, pages 477–492. Springer, 2018.

[JP11] Michal Jancosek and Tomás Pajdla. Multi-view reconstruction preserving weakly-supported surfaces. In *CVPR 2011*, pages 3121–3128. IEEE, 2011.

[KDW$^+$17] Petr Kellnhofer, Piotr Didyk, Szu-Po Wang, Pitchaya Sitthi-Amorn, William Freeman, Fredo Durand, and Wojciech Matusik. 3DTV at home: eulerian-lagrangian stereo-to-multiview conversion. *ACM Transactions on Graphics (TOG)*, 36(4):1–13, 2017.

[KGC15] Alex Kendall, Matthew Grimes, and Roberto Cipolla. PoseNet: A convolutional network for real-time 6-DOF camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, 2015.

[KH04] Nathan Koenig and Andrew Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. IEEE, 2004.

[KK11] Mohamed A Khamsi and William A Kirk. *An introduction to metric spaces and fixed point theory*, volume 53. John Wiley & Sons, 2011.

[KK17] Ms Shakeeba S Khan and AS Khan. A brief survey on robotics. *International Journal of Computer Science and Mobile Computing*, 6(9):38–45, 2017.

[KLS14] Laurent Kneip, Hongdong Li, and Yongduek Seo. UPnP: An optimal O (n) solution to the absolute pose problem with universal applicability. In *European Conference on Computer Vision*, pages 127–142. Springer, 2014.

[KMH+17] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: an interactive 3D environment for visual AI. *arXiv preprint arXiv:1712.05474*, 2017.

[KMT+16] Wadim Kehl, Fausto Milletari, Federico Tombari, Slobodan Ilic, and Nassir Navab. Deep learning of local RGB-D patches for 3D object detection and 6D pose estimation. In *European conference on computer vision*, pages 205–220. Springer, 2016.

[KMT+17] Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. SSD-6D: Making rgb-based 3D detection and 6D pose estimation great again. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1521–1529, 2017.

[Kut00] Kiriakos N Kutulakos. Approximate N-view stereo. In *European Conference on Computer Vision*, pages 67–83. Springer, 2000.

[KW16] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[KWR+16] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. ViZDoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2016.

[KXD12] Alexander Kasper, Zhixing Xue, and Rüdiger Dillmann. The KIT object models database: An object model database for object recognition, localization and manipulation in service robotics. *The International Journal of Robotics Research*, 31(8):927–934, 2012.

[KZ04] Vladimir Kolmogorov and Ramin Zabin. What energy functions can be minimized via graph cuts? *IEEE transactions on pattern analysis and machine intelligence*, 26(2):147–159, 2004.

[LAE+16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[LH96] Marc Levoy and Pat Hanrahan. Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42, 1996.

[LK15] Maziar Loghman and Joohee Kim. Segmentation-based view synthesis for multi-view video plus depth. *Multimedia Tools and Applications*, 74(5):1611–1625, 2015.

[LLF$^+$16] Jing Liu, Chunpeng Li, Xuefeng Fan, Zhaoqi Wang, Min Shi, and Jie Yang. View synthesis with 3D object segmentation-based asynchronous blending and boundary misalignment rectification. *The Visual Computer*, 32(6-8):989–999, 2016.

[LLS$^+$17] Xiaodan Liang, Liang Lin, Xiaohui Shen, Jiashi Feng, Shuicheng Yan, and Eric P Xing. Interpretable structure-evolving LSTM. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1010–1019, 2017.

[LMNF09] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. EPnP: An accurate O (n) solution to the PnP problem. *International journal of computer vision*, 81(2):155, 2009.

[Low04] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[LPK07] Patrick Labatut, Jean-Philippe Pons, and Renaud Keriven. Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cuts. In *2007 IEEE 11th international conference on computer vision*, pages 1–8. IEEE, 2007.

[LPM15] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[LRB$^+$16] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks. In *2016 Fourth international conference on 3D vision (3DV)*, pages 239–248. IEEE, 2016.

[LXX12] Shiqi Li, Chi Xu, and Ming Xie. A robust O(n) solution to the perspective-n-point problem. *IEEE transactions on pattern analysis and machine intelligence*, 34(7):1444–1450, 2012.

[LZS18] Shuai Li, Ce Zhu, and Ming-Ting Sun. Hole filling with multiple reference views in DIBR view synthesis. *IEEE Transactions on Multimedia*, 20(8):1948–1959, 2018.

[LZW$^+$17] Jianjun Lei, Cuicui Zhang, Min Wu, Lei You, Kefeng Fan, and Chunping Hou. A divide-and-conquer hole-filling method for handling disocclusion in single-view rendering. *Multimedia Tools and Applications*, 76(6):7661–7676, 2017.

[MAMT15] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[MB95] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 39–46, 1995.

[MB19] Patrick Mania and Michael Beetz. A framework for self-training perceptual agents in simulated photorealistic environments. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 4396–4402. IEEE, 2019.

[MFY$^+$09] Yuji Mori, Norishige Fukushima, Tomohiro Yendo, Toshiaki Fujii, and Masayuki Tanimoto. View generation with 3D warping using depth information for FTV. *Signal Processing: Image Communication*, 24(1-2):65–72, 2009.

[MHN13] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, volume 30, page 3, 2013.

[MHW$^+$13] Ziyang Ma, Kaiming He, Yichen Wei, Jian Sun, and Enhua Wu. Constant time weighted median filtering for stereo matching and beyond. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 49–56, 2013.

[MKB$^+$17] Frank Michel, Alexander Kirillov, Eric Brachmann, Alexander Krull, Stefan Gumhold, Bogdan Savchynskyy, and Carsten Rother. Global hypothesis generation for 6D object pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 462–471, 2017.

[MLKN09] Raj Madhavan, Rolf Lakaemper, and Tamás Kalmár-Nagy. Benchmarking and standardization of intelligent robotic systems. In *2009 International Conference on Advanced Robotics*, pages 1–7. IEEE, 2009.

[MML$^+$18] Jeffrey Mahler, Matthew Matl, Xinyu Liu, Albert Li, David Gealy, and Ken Goldberg. Dex-Net 3.0: Computing robust vacuum suction grasp targets in point clouds using a new analytic model and deep learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2018.

[Mor78] Jorge J Moré. The Levenberg-Marquardt algorithm: implementation and theory. In *Numerical analysis*, pages 105–116. Springer, 1978.

[MSOC$^+$19a] Ben Mildenhall, Pratul P. Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 2019.

[MSOC$^+$19b] Ben Mildenhall, Pratul P Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. *ACM Transactions on Graphics (TOG)*, 38(4):1–14, 2019.

[MST$^+$20] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing scenes as neural radiance fields for view synthesis. *arXiv preprint arXiv:2003.08934*, 2020.

[NH03] Pauline C Ng and Steven Henikoff. SIFT: Predicting amino acid changes that affect protein function. *Nucleic acids research*, 31(13):3812–3814, 2003.

[NH10] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[OCDD15] Rodrigo Ortiz-Cayon, Abdelaziz Djelouah, and George Drettakis. A Bayesian approach for selective image-based rendering using superpixels. In *International Conference on 3D Vision-3DV*, 2015.

[PCS$^+$15] François Pomerleau, Francis Colas, Roland Siegwart, et al. A review of point cloud registration algorithms for mobile robotics. *Foundations and Trends® in Robotics*, 4(1):1–104, 2015.

[PL15] Sudeep Pillai and John Leonard. Monocular slam supported object recognition. *arXiv preprint arXiv:1506.01732*, 2015.

[QCG$^+$09] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[QSMG17] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[QZZ⁺17] Weichao Qiu, Fangwei Zhong, Yi Zhang, Siyuan Qiao, Zihao Xiao, Tae Soo Kim, and Yizhou Wang. UnrealCV: Virtual worlds for computer vision. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 1221–1224, 2017.

[RC11] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[RDGF16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[RDS⁺15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. ImageNet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[RL01] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the ICP algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152. IEEE, 2001.

[Rob16] Adam Roberts. *The history of science fiction*. Springer, 2016.

[ros] Understanding ROS nodes. `http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes`. Accessed: 2020-05-30.

[RPD08] Edward Rosten, Reid Porter, and Tom Drummond. Faster and better: A machine learning approach to corner detection. *IEEE transactions on pattern analysis and machine intelligence*, 32(1):105–119, 2008.

[RRKB11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *2011 International conference on computer vision*, pages 2564–2571. IEEE, 2011.

[RS10] Robert W Rydell and Laura Burd Schiavo. *Designing Tomorrow: America's World's Fairs of the 1930s*. Yale University Press, 2010.

[RSEG20] Christoph B Rista, David Schmidta, Markus Enzweilera, and Dariu M Gavrilab. SCSSnet: Learning Spatially-Conditioned Scene Segmentation on LiDAR Point Clouds. In *Proc. of the Intelligent Vehicles Symposium (Best Paper Award)*, 2020.

[Rus10] Radu Bogdan Rusu. Semantic 3D object maps for everyday manipulation in human living environments. *KI-Künstliche Intelligenz*, 24(4):345–348, 2010.

[SA12] Mashhour Solh and Ghassan AlRegib. Hierarchical hole-filling for depth-based view synthesis in FTV and 3D video. *IEEE Journal of Selected Topics in Signal Processing*, 6(5):495–504, 2012.

[SCD+17] Manolis Savva, Angel X Chang, Alexey Dosovitskiy, Thomas Funkhouser, and Vladlen Koltun. MINOS: Multimodal indoor simulator for navigation in complex environments. *arXiv preprint arXiv:1712.03931*, 2017.

[SCK08] Heung-Yeung Shum, Shing-Chow Chan, and Sing Bing Kang. *Image-based rendering*. Springer Science & Business Media, 2008.

[SDLK18] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. AirSim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics*, pages 621–635. Springer, 2018.

[SF16a] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4104–4113, 2016.

[SF16b] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[SGHSK20] Caner Sahin, Guillermo Garcia-Hernando, Juil Sock, and Tae-Kyun Kim. A review on object pose recovery: From 3D bounding box detectors to full 6D pose estimators. *Image and Vision Computing*, page 103898, 2020.

[SGT+08] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[SH99] Heung-Yeung Shum and Li-Wei He. Rendering with concentric mosaics. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 299–306, 1999.

[SHKF12] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from RGBD images. In *European conference on computer vision*, pages 746–760. Springer, 2012.

[shr] SHREC2020 - 3D Shape Retrieval Challenge 2020. `http://www.shrec.net/`. Accessed: 2020-06-13.

[SJ15] Michael Schmeing and Xiaoyi Jiang. Faithful disocclusion filling in depth image based rendering using superpixel-based inpainting. *IEEE Transactions on Multimedia*, 17(12):2160–2173, 2015.

[SK00] Harry Shum and Sing Bing Kang. Review of image-based rendering techniques. In *Visual Communications and Image Processing 2000*, volume 4067, pages 2–13. International Society for Optics and Photonics, 2000.

[SKM+19] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, et al. Habitat: A platform for embodied ai research. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 9339–9347, 2019.

[SMD+18] Martin Sundermeyer, Zoltan-Csaba Marton, Maximilian Durner, Manuel Brucker, and Rudolph Triebel. Implicit 3D orientation learning for 6D object detection from rgb images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 699–715, 2018.

[Sof] Software for view synthesis. Software for view synthesis. `http://www.fujii.nuee.nagoya-u.ac.jp/multiview-data/mpeg2/VS.htm`.

[SSN08] Ashutosh Saxena, Min Sun, and Andrew Y Ng. Make3D: Learning 3D scene structure from a single still image. *IEEE transactions on pattern analysis and machine intelligence*, 31(5):824–840, 2008.

[SSS09] Sudipta Sinha, Drew Steedly, and Rick Szeliski. Piecewise planar stereo for image-based rendering. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 1881–1888, 2009.

[Ste99] Charles V Stewart. Robust parameter estimation in computer vision. *SIAM review*, 41(3):513–537, 1999.

[SWS+17] Pratul P Srinivasan, Tongzhou Wang, Ashwin Sreelal, Ravi Ramamoorthi, and Ren Ng. Learning to synthesize a 4D RGBD light field from a single image. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2243–2251, 2017.

[SYZ+17] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1746–1754, 2017.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[SZFP16] Johannes L Schönberger, Enliang Zheng, Jan-Michael Frahm, and Marc Pollefeys. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision*, pages 501–518. Springer, 2016.

[SZZJ17] Zhiqiang Sui, Zheming Zhou, Zhen Zeng, and Odest Chadwicke Jenkins. SUM: Sequential scene understanding and manipulation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3281–3288. IEEE, 2017.

[THo19] T.R. THoogenkamp. *A simulation environment for robot depth and color sensors*. Master's thesis, 2019.

[TMHF99] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment—a modern synthesis. In *International workshop on vision algorithms*, pages 298–372. Springer, 1999.

[TSF18] Bugra Tekin, Sudipta N Sinha, and Pascal Fua. Real-time seamless single shot 6D object pose prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 292–301, 2018.

[UKA+11] Stefan Ulbrich, Daniel Kappler, Tamim Asfour, Nikolaus Vahrenkamp, Alexander Bierbaum, Markus Przybylski, and Rüdiger Dillmann. The OpenGRASP benchmarking suite: An environment for the comparative analysis of grasping and dexterous manipulation. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1761–1767. IEEE, 2011.

[UM20] Claudio Urrea and Rodrigo Matteoda. Development of a virtual reality simulator for a strategy for coordinating cooperative manipulator robots using cloud computing. *Robotics and Autonomous Systems*, 126:103447, 2020.

[VCC+17] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[VS18] Sai Vemprala and Srikanth Saripalli. Vision based collaborative path planning for micro aerial vehicles. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–7. IEEE, 2018.

[WGSJ20] Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. SYNSIN: End-to-end view synthesis from a single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7467–7477, 2020.

[WOC+07] Jianxin Wu, Adebola Osuntogun, Tanzeem Choudhury, Matthai Philipose, and James M Rehg. A scalable approach to activity recognition based on object use. In *2007 IEEE 11th international conference on computer vision*, pages 1–8. IEEE, 2007.

[WRM+08] Daniel Wagner, Gerhard Reitmayr, Alessandro Mulloni, Tom Drummond, and Dieter Schmalstieg. Pose tracking from natural features on mobile phones. In *2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 125–134. IEEE, 2008.

[WSH+19] He Wang, Srinath Sridhar, Jingwei Huang, Julien Valentin, Shuran Song, and Leonidas J Guibas. Normalized object coordinate space for category-level 6D object pose and size estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2642–2651, 2019.

[WSL+19] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph CNN for learning on point clouds. *ACM Transactions on Graphics (TOG)*, 38(5):1–12, 2019.

[WWGT18] Yi Wu, Yuxin Wu, Georgia Gkioxari, and Yuandong Tian. Building generalizable agents with a realistic and rich 3D environment. *arXiv preprint arXiv:1801.02209*, 2018.

[WXZ+19] Chen Wang, Danfei Xu, Yuke Zhu, Roberto Martín-Martín, Cewu Lu, Li Fei-Fei, and Silvio Savarese. Densefusion: 6D object pose estimation by iterative dense fusion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3343–3352, 2019.

[WZW+17a] Nicholas Watters, Daniel Zoran, Theophane Weber, Peter Battaglia, Razvan Pascanu, and Andrea Tacchetti. Visual interaction networks: Learning a physics simulator from video. In *Advances in neural information processing systems*, pages 4539–4547, 2017.

[WZW+17b] Gaochang Wu, Mandan Zhao, Liangyong Wang, Qionghai Dai, Tianyou Chai, and Yebin Liu. Light field reconstruction using deep convolutional network on EPI. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6319–6327, 2017.

[XCJ19] Zelin Xu, Ke Chen, and Kui Jia. W-PoseNet: Dense correspondence regularized pixel pair pose regression. *arXiv preprint arXiv:1912.11888*, 2019.

[XSNF17] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes. *arXiv preprint arXiv:1711.00199*, 2017.

[XZH$^+$18] Fei Xia, Amir R Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese. Gibson env: Real-world perception for embodied agents. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9068–9079, 2018.

[YMB$^+$18] Claudia Yan, Dipendra Misra, Andrew Bennnett, Aaron Walsman, Yonatan Bisk, and Yoav Artzi. Chalet: Cornell house agent learning environment. *arXiv preprint arXiv:1801.07357*, 2018.

[YSW$^+$19] Wang Yifan, Felice Serena, Shihao Wu, Cengiz Öztireli, and Olga Sorkine-Hornung. Differentiable surface splatting for point-based geometry processing. *ACM Transactions on Graphics (TOG)*, 38(6):1–14, 2019.

[YVA$^+$20] Honglin Yuan, Remco C. Veltkamp, Georgios Albanis, Nikolaos Zioulis, Dimitrios Zarpalas, and Petros Daras. SHREC 2020 Track: 6D Object Pose Estimation. In Tobias Schreck, Theoharis Theoharis, Ioannis Pratikakis, Michela Spagnuolo, and Remco C. Veltkamp, editors, *Eurographics Workshop on 3D Object Retrieval*. The Eurographics Association, 2020.

[YYL$^+$14] Jingyu Yang, Xinchen Ye, Kun Li, Chunping Hou, and Yao Wang. Color-guided depth recovery from RGB-D data using an adaptive autoregressive model. *IEEE transactions on image processing*, 23(8):3443–3458, 2014.

[ZCZ$^+$18] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

[ZDdW10] Sveta Zinger, Luat Do, and PHN de With. Free-viewpoint depth image based rendering. *Journal of visual communication and image representation*, 21(5-6):533–541, 2010.

[Zha94] Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. *International journal of computer vision*, 13(2):119–152, 1994.

[ZK07] C Lawrence Zitnick and Sing Bing Kang. Stereo for image-based rendering using image over-segmentation. *International Journal of Computer Vision*, 75(1):49–65, 2007.

[ZKU$^+$04] C Lawrence Zitnick, Sing Bing Kang, Matthew Uyttendaele, Simon Winder, and Richard Szeliski. High-quality video view interpolation using a layered representation. In *ACM transactions on graphics (TOG)*, volume 23, pages 600–608. ACM, 2004.

[ZPQL04] Gang Zeng, Sylvain Paris, Long Quan, and Maxime Lhuillier. Surface reconstruction by propagating 3D stereo data in multiple 2D images. In *European Conference on Computer Vision*, pages 163–174. Springer, 2004.

[ZTF$^+$18] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. *arXiv preprint arXiv:1805.09817*, 2018.

[ZTXM19] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 6(1):11, 2019.