Imperial College London

Department of Computing

Scene Understanding for 3D Multi-Object Scenes: Labelling, Reasoning and Decomposing

Zoe Landgraf

5th February 2023

Supervised by Prof. Andrew Davison and Dr. Stefan Leutenegger

Copyright declaration

Statement of originality

The research presented in this thesis is my own and all external content and information has been referenced appropriately.

Abstract

Scene understanding is a crucial component for intelligent computer vision systems, which are an integral part of most robotic devices, including augmented reality glasses, autonomous cars and future household robots. With the tremendous progress computer vision methods have experienced since the introduction of machine learning and in particular, deep learning techniques to the field, intelligent vision systems seem to be within reach. Some successful applications such as face detection and augmentation have already found their way into our smartphones. However, many challenges remain unsolved, especially in the 3D domain. This thesis addresses three areas of 3D scene understanding with a focus on scenes composed of multiple small objects, which can pose particular challenges and are overall less explored. *Labelling* In a first study, two common deep learning based methods to add semantic labels to a 3D reconstruction in real-time SLAM are compared and evaluated, providing valuable insights on the settings which favour one or the other approach. The experiments are conducted for a table-top setting of small objects and under the assumption, that a mobile device can extensively scan the scene. *Reasoning* However, such an ideal setting is not always possible and the second challenge addressed in this thesis, is that of estimating the content of multi-object scenes when only one or few views are available. To this end, a novel, generative neural network architecture is proposed which can generate the full 3D shape and instance segmentation of a scene from a single depth image. *Decomposing* The method proposed in the second study encodes all features jointly, which leaves no room for scene manipulation within the representation itself. The last experiments explore the question on how to best represent 3D scenes in a compositional way. Compositional, object-level encodings have several advantages including the control over individual features and objects within a scene, the ability to produce novel compositions, as well as a suitable structure to model interactions between scene components. To this end, a novel method to generate a factorised latent representation is proposed, which encodes a scene into a set of per-object latent codes. The proposed method is able to decompose a scene of convex shapes into its components, even from a single viewpoint. We hope that the research results of this thesis can provide insights for solving some of the open problems of 3D scene understanding and that the proposed solutions are relevant for future work and applications.

Acknowledgements

This thesis would not have been possible without the guidance of my supervisor Prof. Andrew Davison and my co-supervisor Dr. Stefan Leutenegger, who taught me many valuable things about research, critical analysis and thinking. I would also like to thank my fellow PhD students and the PostDocs in the Dyson Robotics Lab for their advice and the inspiring discussions and collaborations throughout the duration of my PhD. I would like to thank all my dear friends who supported and accompanied me during this journey and who brightened up even the darkest and rainiest winter days in London. Finally, I want to express special gratitude to my family Annette, Martin, Liora and Victor and to Christiaan for their love, motivation, inspiration and guidance during this important chapter of my life.

Contents

Chapter 1

Introduction

Contents

1.1 Intelligent systems

The last years have brought about technological changes which are said to be driving a fourth industrial revolution: increasing automation, fusion between the digital and real worlds, rapid scientific progress across countries and disciplines, and the merging of technologies within domains such as medicine, genetics, quantum computing, and a newly emerging field: artificial intelligence (AI). This progress also appears to be advancing at unprecedented speed and breadth; compared to previous industrial revolutions which progressed at a linear rate, the current one is evolving at exponential rate [Schwab, 2016]. Connectivity and accessibility may be the driving factors here. People around the world are increasingly connected to one another, through the internet and mobile devices that are slowly becoming a tool for everything: from booking cultural events or flights over transferring money, buying food or goods online, to holding meetings with remote business partners. Most aspects of life have moved online and daily life has become easier and is moving at a faster pace.

Eventually, this revolution will completely transform the way we live. In day-to-day life, one of the most visible aspects of this change will be brought about by robotics and automation, but in particular, by *artificial intelligence*. Simple forms of intelligent systems are already present in various aspects of our daily lives, online and in the physical world.

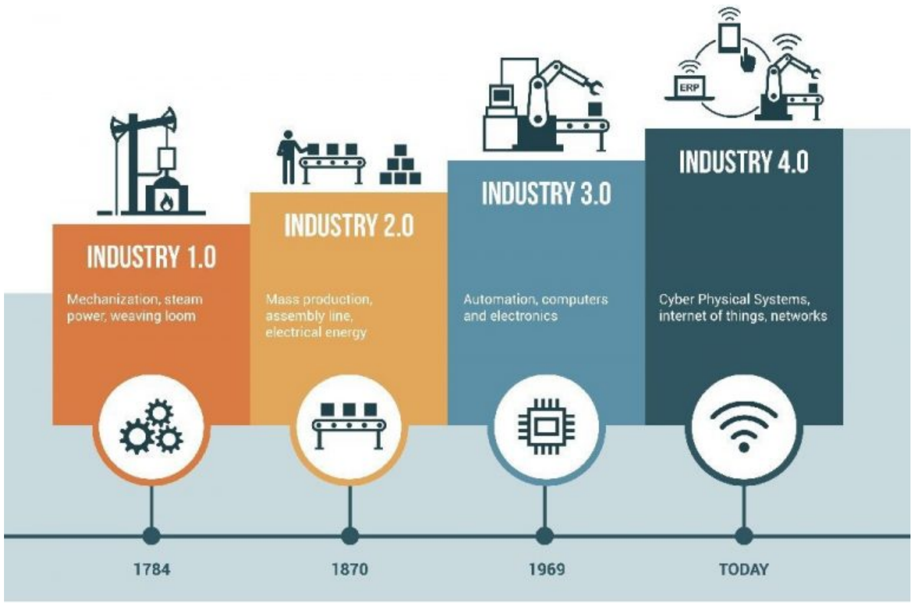

INDUSTRY 1.0
Mechanization, steam power, weaving loom
INDUSTRY 2.0
Mass production, assembly line, electrical energy
INDUSTRY 3.0
Automation, computers and electronics
INDUSTRY 4.0
Cyber Physical Systems, internet of things, networks
ERP
1784
1870
1969
TODAY

Figure 1.1: Current technological progress has been attributed to a fourth industrial revolution which will be characterised by connectivity, merging technologies and intelligent systems [Rose, 2018].

1.1.1 The current use of AI

Online Most automation visible to the general public, takes place online. Smart algorithms parse our e-mails and clear them from spam and phishing content; we get new movie, music and news article suggestions which are tailored to our taste. Most purchases and venue reservations can now be made fast and efficiently, without any human intermediary, through online websites. Overall, we are provided with increasingly automated services which make our lives easier. However, while very effective at data crunching tasks, the realms of the digital world constrain the possibilities of what AI algorithms can achieve. To develop its full potential, AI requires the ability to move and interact.

Embodied Algorithms that operate in the digital world have to understand and process digitised signals — sequences of bits. When embedded into a machine, these algorithms have to understand and process information coming from their surroundings such as light, sound, pressure or thermal measurements. They are connected to the physical world through sensors recording these external signals: cameras, laser scanners, ultrasound sensors, etc. Once processed, the extracted information from the data can be used to navigate within and interact with the environment — the machine becomes a robot.

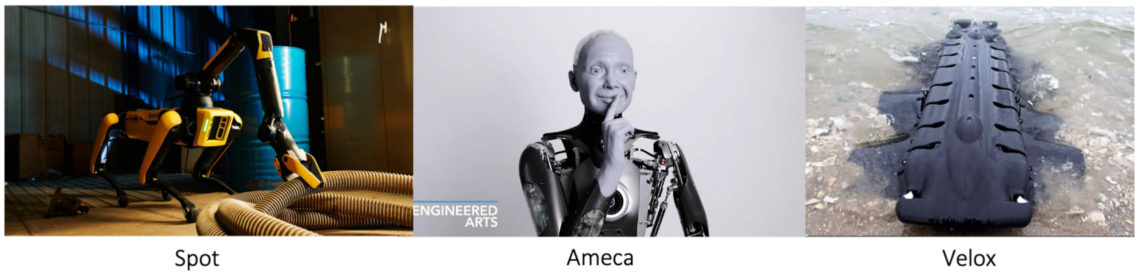


Figure 1.2: Examples of some of the most advanced robots with human- and animal-like agility and advanced sensing capabilities. **Left to right**: *Spot*, a Boston Dynamics robot with self-charging and advanced sensing capabilities, ideal to live on remote industrial sites and perform maintenance and safety checks [Porter, 2021]. *Ameca*, a humanoid robot by Engineered Arts showing human-like face dynamics [Engineered Arts, 2022]. *Velox*, an amphibian-looking ice and water rescue robot with fins whose agility is inspired from various locomotions in the animal kingdom [Pliant Energy Systems, 2022].

Some simple robots are already present in our everyday lives. Surgeries are increasingly supported by robots which enable more precision and minimally invasive surgeries. Although most of these robots are still manually controlled, some already act autonomously [Svoboda, 2019]. A certain degree of autonomy is also present inside modern cars, where assistance systems provide additional safety and comfort features. Through processing signals from our physical world through sensors including cameras and laser scanners, these systems provide features such as lane-keeping assistants, drowsiness detection and in some cases, auto-pilot settings. Another domain where robots are already widely in use is industry: robots are gradually taking over repetitive tasks such as part assembly and conveyer belt operations [Conveyer Concepts Inc, 2020], but also more advanced tasks such as inspection and surveillance by air and on land [Boston Dynamics Team, 2021b] (see Figure 1.2, left). Inside our households, automatic vacuum cleaners such as Roomba [iRobot, 2022], and the Dyson 360 line [Dyson UK, 2022] provide autonomous floor cleaning. Their systems are equipped with sensors and software that allow the robot to build a map of the environment it is cleaning using visual SLAM (Simultaneous Localisation and Mapping). This map helps reason about things such as where to clean next and how to reach a charging station. However, these vacuum cleaners don't have the ability to detect and differentiate objects, nor can they interact with their environment or make complex decisions based on their surroundings (e.g stop cleaning when guests enter the apartment or to clean more thoroughly under the dinner table after dinner). On the engineering side, incredible progress has been made on the mobility and capabilities of robotic systems. With their research platform, the humanoid robot *Atlas*, Boston Dynamics has demonstrated human agility and control, performing parkour walks and even backflips [Boston Dynamics Team, 2021a]. In the domain of human-robot interaction, incredibly realistic facial models have been demonstrated on the humanoid robot *Ameca* by Engineering Arts (see Figure 1.2, centre). Robots also increasingly support rescue missions. *Velox*, an amphibian-bodied water and ice rescue robot by Pliant Energy Systems has fins that mimic various locomotions from the animal kingdom (see Figure 1.2, right).

Despite these incredible technical advances, most of the automated software of currently deployed robots is still limited to specific environments and specific sets of instructions. For instance, one of the leading industrial robotic companies, KUKA, offers separate robots and task-specific software for part assembly, food and pharmaceutical manipulation, welding, and laser cutting [KUKA AG, 2022]. What are the missing features to create more versatile and adaptive robots which can become more general assistants?

1.1.2 Understanding and adapting

Currently, most intelligent systems follow instructions and operate in the specific environments they were programmed for. Some, such as autonomous vacuum cleaners or surveillance robots and drones already have a certain degree of autonomy. More advanced systems will be expected to support us in our everyday lives, such as household robots or personalised Augmented Reality (AR) devices that can help locate lost items, perform tasks online and augment the reality around you (e.g provide information about the building you see or translate the traffic signs when you travel to a foreign country). As these systems accompany us in daily life, they will have to be able to *understand* the world around us and *adapt* to it.

Understanding our physical world involves acquiring knowledge about the geometry of space, but also its semantics (labels of material things such as chair, human, tree, sky) and a concept of *objectness* - the physical space belonging to one single entity, often referred to as an *instance*. Beyond definitions and spatial awareness, knowledge about object properties and their relations will also be crucial for advanced interaction. For instance, to estimate whether or not to lift or push an item to move it, a robot will have to be able to estimate its weight. To tidy a room, a household robot will have to understand that books usually belong into shelves and chairs beneath tables. Adapting to novel environments will require general and versatile state representations, the ability to differentiate novel from previously seen states and to transfer and relate these representations. Furthermore, a robust system will also need the ability to deal with uncertainty and handle unknown situations. The authors of [Davison and Ortiz, 2019] provide an outlook on the overall requirements of future autonomous systems, termed *Spatial AI systems*.

The understanding of the physical world around us is usually referred to as *scene understanding* and, when obtained through visual sensing, *visual scene understanding*. It is an integral part of the algorithmic requirements for intelligent systems and will be the main topic of this body of work.

1.2 Motivation

How to design an intelligent system which can understand its surroundings in a versatile and ad-hoc way remains an open question and is among the most active research areas within computer vision and machine learning. The difficulties are manifold, arising from the complexity of designing robust perception in varying lighting conditions, highly complex and dynamic scenes, dynamically growing state representations or defining the correct way of handling uncertainty. Many problems remain unsolved. While much research focuses on large-scale outdoor and indoor scenes, a particularly challenging setting are scenes with collections of small objects, typical of a table-top scene. Such scenes can be challenging to analyse, because a collection of small objects is more difficult to segment and differentiate due to their size and the partial or full occlusions arising in cluttered scenarios. Developing algorithms that improve 3D scene understanding for small object collections naturally complements the reasoning abilities for larger-scale scenes and is particularly useful for interactive tasks, which require the segmentation and reconstruction of individual (graspable) objects and will, in many cases, address settings with smaller, household-sized objects.

This thesis addresses several aspects of scene understanding, with a focus on scenes with collections of small objects (see Figure 1.3 for a summary). It starts by exploring the question of how to semantically annotate a scene of small objects from a stream of 2D images, a common form of data collected by a moving robot. Despite the progress in semantic segmentation of images using deep learning methods, several questions on how to best propagate semantic segmentation to the 3D domain remain open, in particular, in a real-time setting. Through a principled comparison and analysis of two commonly used methods for semantic scene annotation, Chapter 4 provides some answers for a table-top setting where a scene of small objects is exhaustively scanned.

However, in many scenarios only few views of a scene are available, from which a scene model with annotations should be obtained. This becomes particularly difficult for cluttered scenes, where objects in close proximity generate many occlusions. Chapter 5 is concerned with reasoning about occluded space and how deep generative methods can provide plausible estimates in such scenarios.

Finally, an efficient and descriptive representation of a scene will likely contain a factorised lower dimensional form of all scene components (i.e., an object-centric representation). Understanding a scene through its components will likely facilitate a deeper understanding about object interactions and the ability to independently compose novel scenes. This will be particularly relevant for cluttered scenes of small objects where object interactions have to be modelled and object segmentation and reconstruction is challenging. In the last part of this thesis we explore how to obtain such a representation using deep learning.

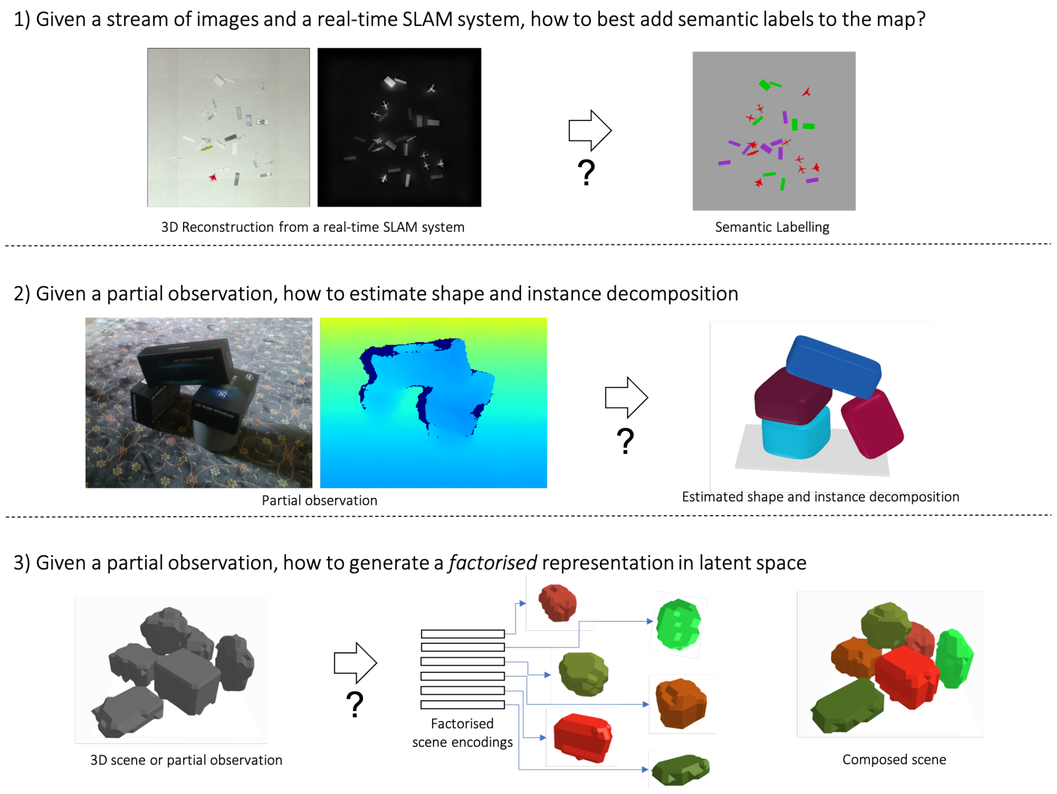

1) Given a stream of images and a real-time SLAM system, how to best add semantic labels to the map?
?
3D Reconstruction from a real-time SLAM system
Semantic Labelling
2) Given a partial observation, how to estimate shape and instance decomposition
?
Partial observation
Estimated shape and instance decomposition
3) Given a partial observation, how to generate a *factorised* representation in latent space
?
3D scene or partial observation
Factorised scene encodings
Composed scene

Figure 1.3: Visual summary of the questions addressed in this thesis.

1.3 Thesis structure and contributions

The rest of this thesis is structured in the following way. Chapter 2 provides some background about the technical field, including visual perception and deep learning. Chapter 3 provides the theoretical and technical preliminaries necessary to understand the main content (Chapters 4 – 6). Chapter 7 concludes with a discussion and summary.

The research produced in Chapters 4 and 5 has been published as follows:

Zoe Landgraf, Fabian Falck, Michael Bloesch, Stefan Leutenegger, Andrew Davison. *Comparing View-based and Map-based semantic labelling in real-time SLAM*, International Conference on Robotics and Automation (ICRA), 2020.

Zoe Landgraf, Raluca Scona, Tristan Laidlow, Stefan Leutenegger, Andrew Davison. *SIMstack: A Generative Shape and Instance Model for unordered object stacks*, International Conference on

Computer Vision (ICCV), 2021.

CHAPTER 2

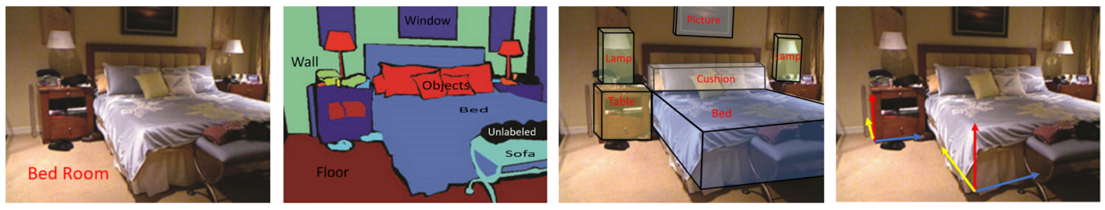
Background

Contents

2.1 Visual scene understanding

To understand a scene, systems can leverage multiple data modalities, including sound and touch; however, one of the most immediate and expressive ways to understand a scene is through images. As humans, we can look around us and quickly understand where we are, which objects are around us and how these objects relate to ourselves and to each other. We can also estimate physical properties such as weight, speed, stability, all from a single glance. Although our understanding will be augmented by sounds and smells, our main source of information comes through our sight [Hutmacher, 2019]. Being able to replicate our ability of visual scene understanding in computer systems will greatly increase their ability to understand, interact and adapt.

Scene understanding through visual data — *visual scene understanding* — involves collecting data through sensors such as cameras, depth sensors or laser scanners and processing the recorded data to extract information about the geometry and the content of the observed scene. Extracting scene geometry from images is one of the main goals for algorithms such as Structure from Motion (SfM) or Simultaneous Localisation and Mapping (SLAM), that generate a 3D map from a sequence of recorded images or laser scans (a crucial component in applications such as autonomous driving or augmented reality applications). Understanding the content of the scene (e.g. detecting objects and assigning semantic labels) is crucial to enable automatic detection of pedestrians in the vision systems of an autonomous car, but also in other domains, such as detecting tumours when processing medical images. Figure 2.1 illustrates some examples of tasks in visual scene understanding.



(a) Scene Classification (b) Semantic Segmentation (c) Object Detection (d) Pose Estimation

Figure 2.1: Examples of scene understanding tasks. **Right to left:** scene classification, semantic segmentation, object detection and pose estimation [Naseer et al., 2019]

Visual scene understanding is not a new field and decades of research have produced impressive and robust solutions for some of the problems involved. In the following, the areas of visual scene understanding most relevant to this thesis will be introduced.

2.1.1 Image and scene segmentation

The goal of segmentation is to partition an image or a scene into different regions for the purpose of better understanding the properties and the content of the captured information. Among the most common goals of segmentation are semantic segmentation — assigning a label to every pixel — and object segmentation — segmenting the region belonging to one object. This is commonly referred to as *instance* segmentation. Early image segmentation methods relied on simple thresholding methods (dividing an image into regions based on pixel intensity) such as Otsus' algorithm and extensions [Otsu, 1979, Cheriet et al., 1998], edge detection methods that find discontinuities in the image using edge detectors such as the Canny algorithm [Canny, 1986]) and region-based methods (e.g. region growing – segmentation seed points are grown into neighbouring regions following a set of rules). After the 2000s, techniques were increasingly based on graph theory and clustering [mei Zhou et al., 2008, Cigla and Alatan, 2008]. In the last decade, the introduction of deep learning based methods, in particular Convolutional Neural Networks (CNNs), revolutionised the field. One of the first methods to use a CNN for per-pixel semanic labelling was the work of Farabet *et al*. [Farabet et al., 2013] and one year later, R-CNN became one of the first methods to surpass hand-engineered semantic segmentation methods [Girshick et al., 2014]. The real breakthrough however, was brought about by the introduction of the Fully Convolutional Network (FCN) [Long et al., 2015], an elegant architecture that allowed for per-pixel prediction, ideal for semantic segmentation and instance segmentation tasks. State of the art semantic segmentation methods such as DeepLabv3 [Chen et al., 2017] or PSPNet [Zhao et al., 2017a] all derive from this architecture and achieve over 80% by the Intersection over Union metric on public datasets. For instance segmentation, state of the art architectures either employ single stage methods (based on the FCN architecture) such as BlendMask [Chen et al., 2020a] and SOLOv2 [Wang et al., 2020b], or two-stage approaches that build on R-CNN, such as Mask-RCNN [He et al., 2017]. Recently, combining semantic and instance segmentation in one task — *panoptic segmentation* [Kirillov et al., 2019] — has gained increasing attention in the community.

With the advancement of segmentation in images, a variety of works started looking into adding segmentation results to 3D reconstructions [McCormac et al., 2016, Nie et al., 2020a], as well as directly segmenting 3D scenes [Dai et al., 2018]. Compared to 2D segmentation, 3D segmentation

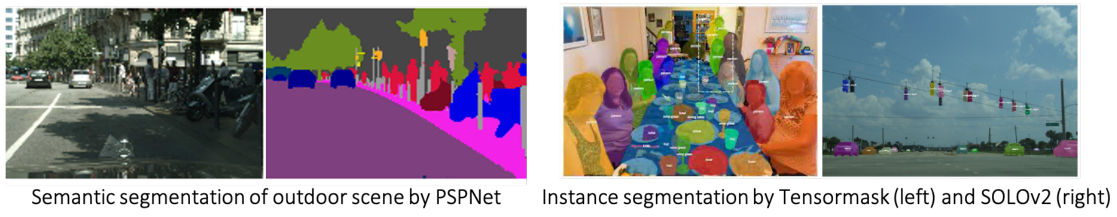

Semantic segmentation of outdoor scene by PSPNet
Instance segmentation by Tensormask (left) and SOLOv2 (right)

Figure 2.2: Examples of state of the art semantic and instance segmentation methods for images. **Left**: outdoor scene segmentation by PSPNet [Zhao et al., 2017a]. **Right**: instance segmentation of two images by Tensormask [Chen et al., 2019] and SOLOv2 [Wang et al., 2020b]

is a lot more difficult and despite great progress achieved in the last few years, for segmentation and annotation in 3D, several challenges remain unsolved and results lack accuracy and completeness (see Figure 2.3). Some of the currently unsolved problems in segmenting 3D scenes will be explored in this thesis.

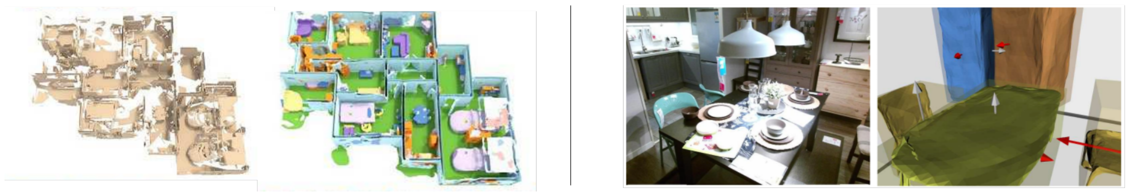


Figure 2.3: Examples of segmenting 3D scenes. **Left**: semantics predicted by the ScanComplete system [Dai et al., 2018]. **Right**: instance segmentation output of Total3D [Nie et al., 2020a]. Both systems are state of the art, however, segmentation in 3D still lacks accuracy.

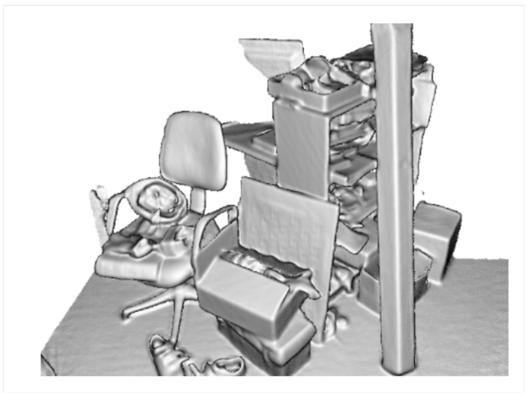
2.1.2 SfM and SLAM

Estimating 3D structure from 2D images is a long-studied problem in computer vision and an integral part of scene understanding. The difficulty of the problem lies in the image formation process being inherently uninvertible: from its projected position in the image plane, a scene point can only be recovered as an infinite line - scale information is lost. The geometrical theory of SfM allows to simultaneously estimate the 3D structure from two image views and their camera positions. The prerequisite is having solved the 'correspondence problem' - establishing sparse or dense pixel correspondences between the viewpoints. For multiple views, images can be processed in batches or sequentially, to obtain the full reconstruction. A final optimisation step is usually performed using *bundle adjustment* [Pritchett and Zisserman, 1998]. SLAM extends the concept of SfM to a real-time and dynamic algorithm which is essential for autonomous robot motion. The goal of a system running SLAM is to localise itself in the environment in real time while building a map and navigating through it. The built map improves localisation, provides information about geometry and can be augmented with information such as semantics. SLAM can in principle be run with a variety of sensors; when using cameras, it is usually referred to as Visual SLAM.

Origins and early work The earliest methods to estimate structure from a set of images were already applied in photogrammetry in the early 1800*s* to estimate the shape of terrain from aerial images [Micheletti et al., 2015]. Among the first approaches in the computer vision community is the seminal work of Longuit-Higgins *et al.* [Longuet-Higgins, 1981] in 1981, proposing the eight point algorithm. Based on projective geometry, this algorithm allows to estimate the fundamental matrix – which can be used to recover the scene – from two images with unknown correlation and a set of eight or more point matches. Much work was then put into extending these initial ideas for more images and making the algorithms robuster and faster. Some important works include the *factorisation algorithm* proposed by Tomasi & Kanade, a fast and elegant solution to the SfM problem based on Singular Value Decomposition (SVD) [TomasiCarlo and KanadeTakeo, 1992], as well as several iterative solutions based on Levenberg-Marquardt non-linear optimisation [Raja Kumar et al., 1989, Szeliski and Kang, 1994, Weng et al., 1989]. Nowadays, most multi-view SfM methods use bundle adjustment techniques which optimise pose and structure estimates based on the reprojection error [Özyesil et al., 2017]. Addressing the correspondence problem, several approaches were proposed to improve edge and corner detection [Canny, 1986], [Harris and Stephens, 1988] and later, more advanced feature-based detectors of which the most famous include SIFT [LoweDavid, 2004], its faster version SURF [Bay et al., 2008], and, optimised for real-time performance: ORB [Rublee et al., 2011].

Research in SLAM developed around the same time. Early work explored the statistical relationships between observed landmarks and established the problem as a joint estimation of landmarks and vehicle positions [Smith et al., 1986, Smith et al., 1988]. The acronym 'SLAM' was first introduced in 1995 along with the structure of the problem [Durrant-Whyte and Bailey, 2006]. Early

SLAM algorithms used Extended Kalman Filter (EKF) based methods and Rao-Blackwellised Particle Filter methods, introduced by Montemerlo *et al*. [Montemerlo et al., 2002]. Most modern SLAM systems derive from methods based on maximum likelihood estimation introduced by Thrun *et al*. [Thrun et al., 1998]) that were further improved upon by several works including iSAM [Kaess et al., 2008] and GraphSLAM [ThrunSebastian and MontemerloMichael, 2006]. Other nominal works include MonoSLAM [Davison et al., 2007], the first system to move SLAM to a pure visual domain, PTAM [Klein and Murray, 2007] a system particularly designed for efficient reconstruction and tracking using a hand held camera and ORB-SLAM [Mur-Artal et al., 2015a], which extends PTAM to include loop-closures and uses ORB-features for efficient feature matching. As the field matured and more compute power became available, *direct* SLAM methods, also called *dense* methods were proposed, which directly optimise the map and robot position from image intensity values [Pizzoli et al., 2014, Newcombe et al., 2011b, Izadi et al., 2011, Engel et al., 2014b]. Using dense mapping not only allows for pleasing visual renderings (see Figure 2.4) and useful properties for augmented reality applications, but also improves the robustness of real-time tracking during fast motion [Newcombe et al., 2011b].



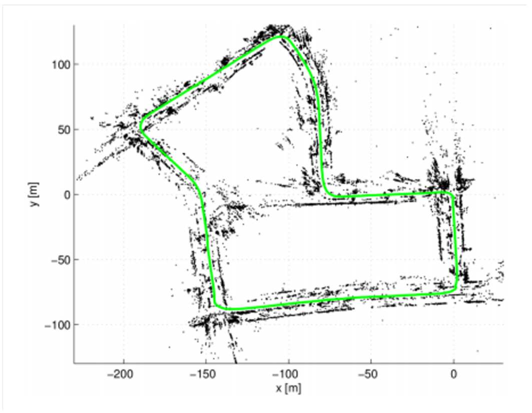


Figure 2.4: Dense vs. Sparse SLAM. **Left:** dense reconstruction of KinectFusion [Newcombe et al., 2011a]. **Right:** sparse landmarks and trajectory of ORB-SLAM [Mur-Artal et al., 2015a]

Semantics in SLAM As the field of geometric SLAM matured, research started looking into annotating the reconstructions with informative labels and concepts. Many early methods that add semantics to the map, apply offline segmentation: first the scene is reconstructed geometrically, then segmented into semantic concepts [Herbst et al., 2011, Pillai and Leonard, 2015, Koppula et al., 2011, Pham et al., 2015]. Other methods semantically label in an online fashion [Pronobis and Jensfelt, 2012, Cadena et al., 2015]. Many approaches that pre-dated the wide adaptation of deep learning, such as [Pham et al., 2015, Valentin et al., 2013], use Conditional Random Fields to infer labels for the reconstructed map. [Vineet et al., 2015] use a dense reconstruction system where semantic labels are inferred using a densely connected CRF. [Häne et al., 2013] leverage the complementary nature of 3D geometry and image-based segmentation and jointly predict per-voxel semantics and occupancy using a CRF. Other approaches still, choose an object-level representation [Castle et al., 2007, Salas-Moreno et al., 2013].

Deep Learning in SLAM With the recent success of deep learning methods for a variety of computer vision problems, several approaches have replaced individual components of the SLAM algorithms with learning based methods [Bloesch et al., 2018, Czarnowski et al., 2020, Mohanty et al., 2016] or even attempt to learn end-to-end SLAM with a deep neural network [Bruno and Colombini, 2021]. Other applications leverage deep learning to augment the geometric map with additional information such as semantic labels or object level segmentation [McCormac et al., 2017a, McCormac et al., 2018, Sünderhauf et al., 2017].

However, many questions on how to robustly add semantic and instance labels to 3D reconstructions, as well as how to deal with uncertainty remain unsolved and are among the questions being addressed in this thesis.

2.2 AI, machine learning and deep learning

The field of computer vision has advanced rapidly in the last few years and a large part of this success can be attributed to breakthroughs in domains such as image classification and semantic segmentation achieved by machine learning and in particular by deep learning methods.

AI is generally considered the umbrella term for everything ranging from logic-based reasoning, over computer vision to natural language processing. It also includes machine learning and deep learning within its sub-fields (see Figure 2.5). While AI includes systems which are more biologically inspired, such as spiking neural networks, machine learning is rooted in statistical theory and predominantly used in computer science for algorithms that are optimised with respect to a set of rules, or data. It includes classical machine learning models such as support vector machines and kernel methods, decision forests and clustering methods, as well as neural networks. Deep learning is usually linked to neural network models with more than one hidden layer (deep neural networks).

Whether based on classical architectures or neural networks, the ways in which machine learning algorithms learn depends on how they experience the data they see or the environment they are placed into. Learning methods can be loosely divided into *Supervised Learning*, where the algorithm learns from correct examples, *Unsupervised Learning*, where the algorithm has to discover patterns in the data and *Reinforcement Learning*, where the algorithm learns a policy (per-

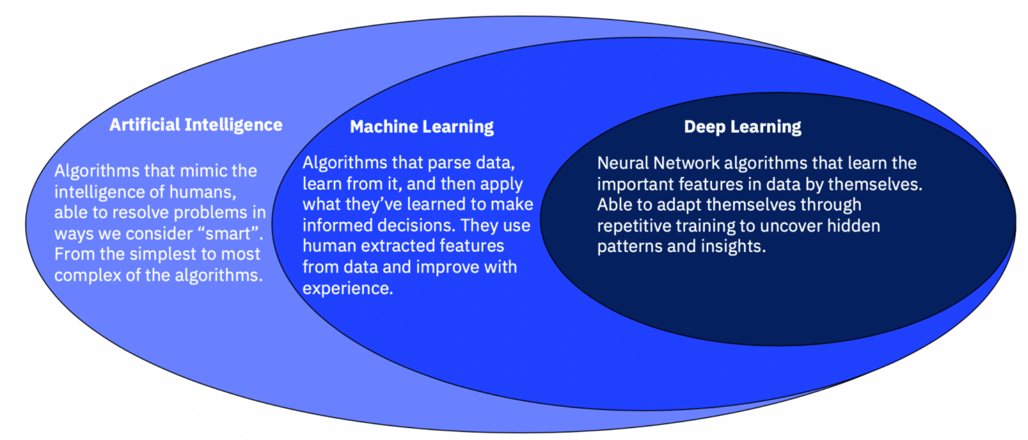

Artificial Intelligence
Algorithms that mimic the intelligence of humans, able to resolve problems in ways we consider "smart". From the simplest to most complex of the algorithms.
Machine Learning
Algorithms that parse data, learn from it, and then apply what they've learned to make informed decisions. They use human extracted features from data and improve with experience.
Deep Learning
Neural Network algorithms that learn the important features in data by themselves. Able to adapt themselves through repetitive training to uncover hidden patterns and insights.

Figure 2.5: Definitions of AI, machine learning and deep learning and how they are related [Ceron, 2019]

forming a set of instructions or behaviour) based on feedback they obtain from an environment (such as a game). See Figure 2.6 for an illustration.

A brief history The origins of machine learning and AI can be traced back to a few key events in the 1940*s* and 1950*s*. The first mathematical model of a neural network was proposed by McCulloch and Pitts [McCulloch and Pitts, 1943]. A decade later, Arthur Samuel, often considered the first one to have popularised the term machine learning, wrote the first championship level checkers computer. It used the minimax algorithm to find the best move, alpha-beta pruning to avoid searching all possible paths and remembered previous moves to improve its strategy — it learned from experience [History Computer, 2021]. Often considered the birthplace of AI, another key event was the Dartmouth College workshop in 1956, where scientists of engineering, maths and cognitive sciences gathered and discussed the fields of AI and machine learning. Not long after, the first multi-layer perceptron [Rosenblatt, 1958] was proposed and in 1969, Fukushima published the *neocognitron*, a hierarchical neural network which inspired today's convolutional neural networks [Fukushima, 1969]. In 1979, the Stanford Car becomes the first autonomous vehicle to navigate a room full of chairs without human help [Stanford University, 1979]. 1986 sparked new interest in neural networks thanks to the backpropagation algorithm proposed by Rumelhard *et al*. [Rumelhart et al., 1986], which is at the basis of modern deep learning methods. However, it wasn't until the last decade, when the required computing power and hardware became available, that neural networks became the powerful tool they are today. Current breakthroughs achieved with methods based on deep learning span a variety of domains such as game playing (AlphaGo beats the world's Go playing champion [Deep Mind, 2015]), biology (AlphaFold solved the 50 year grand challenge of the protein folding problem [Deep Mind, 2021]), and nuclear physics (A deep reinforcement learning method controls nuclear fusion plasma [Deep Mind, 2022]).

Figure 2.6: Types of learning: supervised, where correct examples are shown during training; unsupervised, where the machine learning model discovers patters independently; reinforcement learning, where the machine learning model operates within a simulated environment (such as a game) and learns a policy through receiving rewards for correct actions or reaching goals. Image references [Arora, 2020, Lee, 2019].

2.3 Deep learning for computer vision

The modern age of deep learning in vision is usually said to have begun with AlexNet, a CNN which achieved impressive accuracy in the ImageNet Large Scale Visual Recognition Challenge in 2012 [Krizhevsky et al., 2012]. Since then, CNN's have surpassed human performance in image classification [He et al., 2015] and have been extended for various other scene understanding tasks (e.g. semantic segmentation, instance segmentation, pose estimation). Popular descendent architectures include the fully convolutional network (FCN) [Long et al., 2015], an elegant solution for per-pixel prediction, the residual network [He et al., 2016] which enabled very deep networks and U-Net [Ronneberger et al., 2015], which introduced the concept of skip-connections, that can transfer fine detail for per-pixel predictions. Several adaptations for 3D modalities such as voxels [Maturana and Scherer, 2015], pointclouds [Qi et al., 2017]and meshes [Hanocka et al., 2019] have been proposed. On the content generation side, variational autoencoders (VAEs) and generative adversarial networks (GANs) have become popular training methods. GANs in particular, have revolutionised image generation, with current state of the art models [Park et al., 2019b, Karras et al., 2019] being able to generate highly realistic natural images and even transfer styles between images (see Figure 2.7). Note that VAE and GAN models usually have a CNN backbone and can also be extended to different model architectures; their difference lies in the training method and loss function.

Current frontiers Although CNN's remain the main architecture for vision tasks, they are currently being challenged by a new architecture, the *transformer*. Originally introduced for natural language processing [Vaswani et al., 2017], this architecture was recently adapted for vision

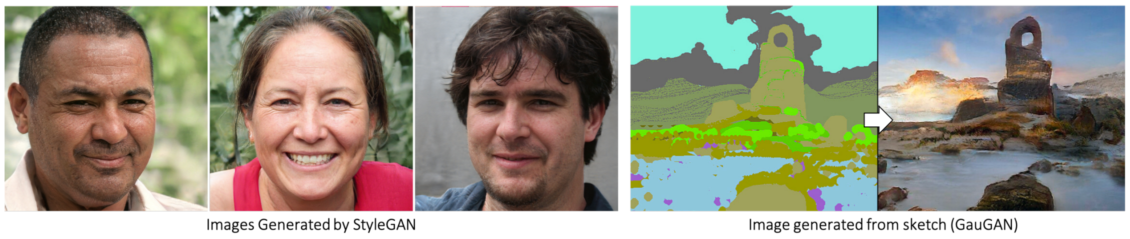

Images Generated by StyleGAN
Image generated from sketch (GauGAN)

Figure 2.7: **Left**: realistic faces generated by StyleGAN [Karras et al., 2019]. **Right**: Sketch to image translation by GauGAN [Park et al., 2019b].

tasks [Dosovitskiy et al., 2020]. Transformers are based on a mechanism called *attention*, which establishes and learns correlations between different components of an encoded signal. As supervised learning is reaching maturity as a field and the inherent problems of learning from labelled data or examples (expensive labelling, covariate shift, etc.) persist, the community is starting to embrace the challenges of semi-supervised, self-supervised, unsupervised and few-shot learning. While being algorithmically more difficult and complex to design, these methods hold a lot of promise for domain adaptation and generalisation of machine learning algorithms. Other frontiers include neural scene representation, Graph neural networks and modelling causality.

For the research of this thesis, the backbone architecture of all developed models are based on CNNs. For the task of single-view 3D shape and instance generation, the model is trained as a 3D VAE. All tasks are learnt in a supervised setting. The architectures and training concepts will be introduced in more detail in section 3.

CHAPTER 3

Preliminaries

Contents

This chapter introduces the technical and mathematical details which are used in the research of this thesis. We review the algorithms of Simultaneous Localisation and Mapping (SLAM) and different methods to represent 3D space, as such a map is built. Then, the fundamentals of machine learning and deep learning, as well as the relevant neural network architectures are introduced. This is followed by a description of visual perception methods with deep learning and the relevant rendering techniques. Finally, software and implementation details are provided.

3.1 SLAM

SLAM is the process of concurrently estimating the state of a robot with on-board sensors and building a representation of the environment the robot is moving in — *mapping*. The robot state is usually described by quantities such as position and orientation, but can also include velocity and sensor information. The map can be represented using sparse landmarks or using a geometric representation (see Section 3.2 for an overview of different 3D geometric representations). The need for a map arises from the problem of drift in dead-reckoning, which inevitably happens as errors accumulate without external re-localisation. Furthermore, a map can provide information for other applications such as path planning, interactions such as grasping, or visualisations. Generally, SLAM finds application in all situations where a map cannot be obtained a priori (in cases where a map infrastructure is present, such as factory floors, SLAM might not be necessary) and has become particularly popular with the development of indoor robotics where the localisation error cannot be bound using GPS signals [Cadena et al., 2016b]. The structure of a SLAM system can be broadly separated into a *front-end*, which collects and processes sensor-data and a *back-end*, which solves the bulk of the optimisation problem (see Figure 3.1 for a visualisation by [Cadena et al., 2016b]).

In the following, we will provide more details on the structure of the SLAM optimisation problem and data association methods, such as keypoint feature detection and matching.

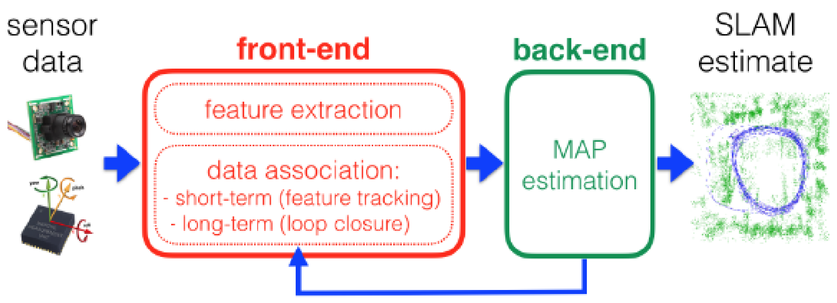


Figure 3.1: Diagram illustrating the overall structure of a SLAM system pipeline. The front-end collects and processes sensory data while the backend solves the optimisation problem (in modern SLAM systems usually a maximum a posteriori estimation (MAP)). The backend can provide feedback to the front end for refinement processes such as loop-closure [Cadena et al., 2016b].

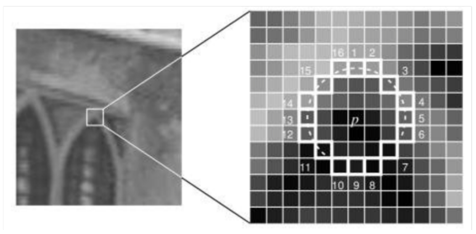
3.1.1 Odometry and visual odometry

An initial estimate of a robot's location can be obtained from odometry, e.g. using the data from motion sensors on the robot such as wheel encoders. By counting the relative number of rotations with respect to a timepoint, displacement can be estimated. The errors on such measurements quickly accumulate and need correction through external measurements. In visual SLAM, motion is usually estimated using images only — *visual odometry*. Here, the collected measurements consist of image data from which the camera poses can be estimated. The first steps in this process are keypoint feature detection and feature matching across frames (a form of data association) and subsequently, refinement and outlier detection using Random Sample Consensus (RANSAC) [Fischler and Bolles, 1981]. Once feature matches have been established, the transformation T between consecutive frames can be estimated.

3.1.2 Keypoint features

Data association can take multiple forms depending on the sensor modality; for visual SLAM it involves keypoint feature detection and matching across consecutive frames.

In order to find correspondences between two images for visual odometry, characteristic keypoint features such as corners and edges are detected and transformed into feature descriptors that are then matched across views with feature matching algorithms. Feature descriptors are a collection of characteristic points, usually high contrast points or edges, which will remain similar under different viewpoints and lighting conditions. This allows for robust matching with the same points in another view of the same region. Apart from SIFT [Lowe, 1999] and BRISK [Leutenegger et al., 2011], one of the most popular such methods is Oriented FAST and rotated BRIEF (ORB) [Rublee et al., 2011]. As the name suggests, ORB builds on the FAST keypoint detector and the BRIEF feature descriptor, adding several improvements such multi-scale features and the addition of rotation invariance. The main concept of FAST, is to find pixels which are surrounded by neighbours that have a strong difference in intensity — i.e. represent corners, edges. BRIEF descriptors are then created with patches surrounding selected keypoints. After applying a Gaussian blur for robustness, a number of pixel pairs are sampled from the patch. These pixel pairs are used to create a feature descriptor based on a binary selection function τ: Depending on whether the first or second sampled pixel is brighter, a 0 or 1 is saved in a *binary vector*, from which the feature descriptor is generated as: $f = \sum 2^{i-1}\tau(p_i)$ [Viswanathan, 2011]. ORB adds an orientation to the descriptor, by computing the intensity centroid of the patch and computing the vector between the patch center



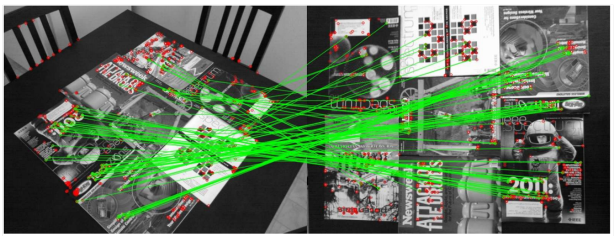


Figure 3.2: **Left**: FAST feature detection. The highlighted pixels are those used in corner detection and the pixel at p is the candidate pixel of the corner detection. If a set number of contiguous pixels in the arc (dashed line) are brighter than p by a set threshold, p is selected as keypoint [Viswanathan, 2011]. **Right**: ORB feature matches [Rublee et al., 2011]

and its intensity centroid. This information is then included into the descriptor, effectively making it rotation invariant. An example of detected ORB features is given in Figure 3.2, right. To match feature descriptors across images, ORB uses Locality Sensitivity Hashing (LSH) [Rublee et al., 2011].

3.1.3 The SLAM backend

Once the sensor data has been collected and processed during the data-association step, the pre-processed data can be passed to the SLAM backend for optimisation.

The setup the SLAM backend has to solve, can be visualised as a graph, such as the one displayed in Figure 3.3. The nodes u_t represent the location estimates provided by odometry and z_t are the external measurements of the environment, based on the robot locations. x_t and m are the true robot locations and landmark positions in the environment; the latent variables to be estimated. The edges which connect the individual measurements are characterised by measurement models such as motion estimation through wheel encoders and camera projection models. The SLAM problem can be solved offline as a full estimation of the posterior over all poses and external measurements: $P(X_t, m|Z_t, U_t)$, or online, estimating the current pose x_t based on all previous measurements: $P(x_t, m|Z_t, U_t)$. There are different methods to estimate this posterior, of which most derive either from Extended Kalman Filter (EKF) methods, Particle Filter approaches or Expectation Maximisation [Stachniss et al., 2016, Aulinas et al., 2008].

MAP estimation and factor graphs Although there exist state of the art methods based on EKF [Mourikis and Roumeliotis, 2007, Kottas et al., 2012, Hesch et al., 2014], MAP estimation has proven to be more accurate and efficient in most cases and has become the standard modern formulation of SLAM [Cadena et al., 2016b]. To formulate SLAM as a MAP estimation, one

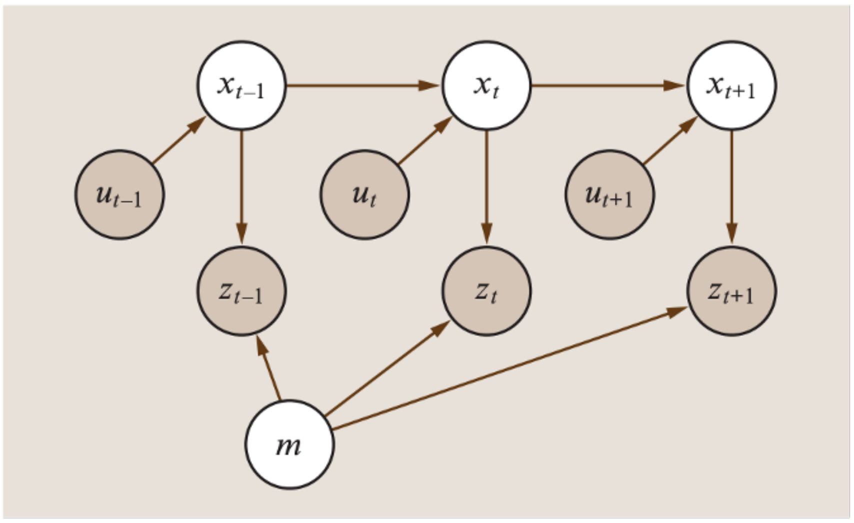


Figure 3.3: Diagram illustrating the observed (shaded) and estimated (white) variables in SLAM. x_t are the true robot locations and m is a true landmark location. u_t and z_t are the estimated robot locations and landmark positions, respectively [Stachniss et al., 2016].

considers a set of variables X_k, which include the trajectory of the robot and a set of discrete landmarks in the scene. The set of measurements $Z = \{z_k : k = 1,\ldots,m\}$ can then be expressed as a function of X: $z_k = h_k(X_k) + \varepsilon_k$, where $h_k(\cdot)$ is the measurement or observation function and ε_k is random measurement noise [Cadena et al., 2016b]. Note that following the conventions in literature, we use x for latents and z for measurements here, while these definitions are swapped once we describe Neural Networks in the following chapters, where latent variables are commonly referred to by z. During MAP estimation, the assignment X^* to the variables X is found, which maximises the posterior $p(X|Z)$:

$$X^* = \underset{X}{\operatorname{argmax}}\, p(Z|X)p(X), \tag{3.1}$$

where $p(Z|X)$ is the likelihood of the measurements Z given X and $p(X)$ represents any prior knowledge about X. Equation 3.1 can be rewritten as a negative log-likelihood minimsation problem:

$$X^* = \underset{X}{\operatorname{argmin}} -log(p(Z|X)p(X)) \tag{3.2}$$

MAP estimation is usually solved using iterative linearisation techniques such as the Gauss-Newton or Levenberg-Marquardt methods.

Recently, combining MAP estimation with a *factor graph* representation has become a popular SLAM formulation. Factor graphs are a class of graphical models where a set of variables (the unknown quantities to be estimated) are connected through factors which represent functions on a subset of the variables. The interdependency between factors and variables is modelled through edge connections. An illustration of a factor graph representation applied to the SLAM problem can be seen in Figure 3.4.

Under the assumption that the measurements Z are independent, Equation 3.1 can be factorised:

$$X^* = \underset{X}{\operatorname{argmax}}\, p(X)\Pi_{k=1}^{m} p(z_k|X_k), \tag{3.3}$$

where every measurement z_k only depends on a subset of the variables X_k. This can be interpreted in terms of inference over a factor graph [Cadena et al., 2016b]. In fact, factor graphs a very suitable representation to leverage the sparsity and locality in the structure of the SLAM graph [Dellaert, 2021].

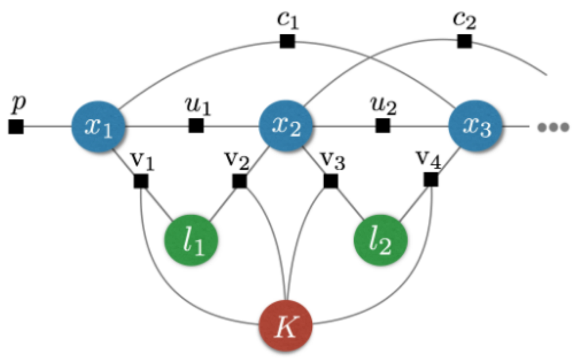


Figure 3.4: The SLAM problem modelled as a factor graph. Blue nodes represent robot poses at incremental time steps, green nodes denote landmarks and the red node is the variables associated with the intrinsic camera parameters. Black squares denote factors; u_n are factors representing odometry constraints, v_n model camera measurements and c_n are loop closure constraints and p models the prior knowledge [Cadena et al., 2016b].

3.1.4 Loop closure

Even when obtaining position and motion estimates using external measurements and using a robust back-end optimisation algorithm, a robot's trajectory will eventually drift. For large-scale SLAM systems it is therefore important to perform regular loop-closures, whereby the system attempts to detect whether or not a place has already been visited before (place recognition) and if so, adds an additional constraint (e.g. a factor) to the graph (this is where the SLAM back-end communicates with the front-end). Robust place recognition is a particularly challenging and long-studied problem. It lies outside the scope of this thesis and we refer the reader to [Lowry et al., 2016] for an excellent overview.

3.2 3D representations of space

In sparse SLAM systems, the environment is represented through sparse 3D feature points, that are mainly useful for relocalisation of the autonomous agent. For other applications such as path plan-

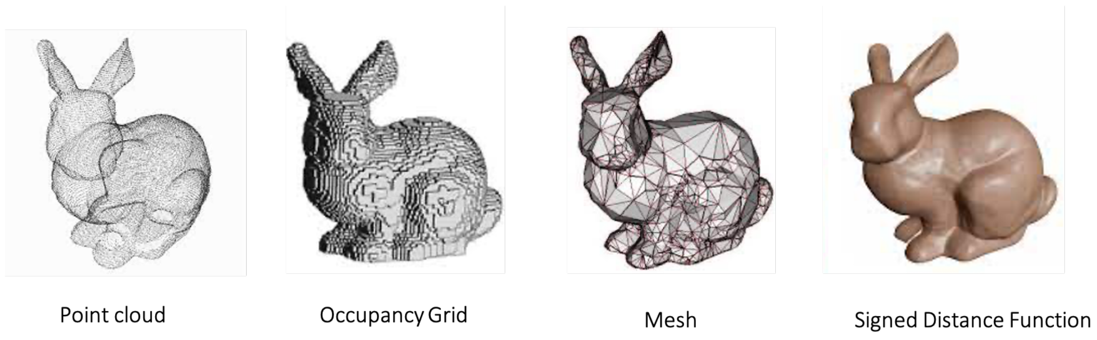


Figure 3.5: **Left to right**: Examples of point cloud, occupancy grid, polygon mesh and signed distance function (SDF) representations. To obtain the smooth surface representation, the SDF is processed using Marching Cubes.

ning or more advanced interactive tasks, a more complete model of the environment is required. A number of different approaches exist for modelling the 3D world around us.

One of the simplest representations are **topological maps**, which model different points of interest relative to each other using a graph representation. An example of such a map can be a set of road segments and intersections to represent the environment for autonomous cars. Topological representation are particularly memory efficient, however, they are limited in descriptiveness since they don't contain much information about the geometric structure of the environment. Alternatively, a few representations exist, which more explicitly model detailed geometry.

Point clouds Popular for their memory efficiency, point clouds are an unordered, sparse collection of points, each holding a position (x, y, z) in 3D space (see Figure 3.5, left). Usually generated by LIDAR sensors, point clouds have the advantage of inherently encoding the concept of free and occupied space. However, there is no differentiation between free and unknown space and the unstructured nature of point clouds makes it difficult to process them for visual perception tasks such as segmentation.

Polygonal meshes are another common representation of 3D space and a standard representation in computer graphics. A polygon mesh consists of a set of triangles composed by vertices which are connected through edges (see Figure 3.5, third bunny from the left). Their irregular nature makes them an ideal representation for complex shapes with very detailed as well as smooth regions. However, this irregular structure also makes them complicated to process and mainly suitable for graph-based algorithms.

Volumetric representations using voxel-grids provide regularity and allow for precise estimation of free and occupied space, which is necessary for some tasks within SLAM and computer vision such as path planning. They can be used to model binary occupancy (usually 0 for unoccupied and

1 for occupied space) or signed distance functions (SDF). SDFs model surfaces as the levelset of a continuous function $f : f(\mathbf{x}) = 0$. At any given point x in space the function outputs the distance to the closest surface. The sign of the function determines whether x is inside or outside of the surface. In practice, SDFs are stored as discrete voxelgrids whereby each voxel center stores the distance to the closest surface. Intermediate values can be estimated through interpolation. When processed using Marching Cubes, SDFs representations yield very smooth surface representations, compared to the blocky representation obtained from binary occupancy grids (see Figure 3.5). In practice, SDF representations are often truncated to a TSDF, i.e. set to a constant value after a certain distance.

Height maps Volumetric representations can be costly in terms of memory and processing time, as they scale cubically in size. Therefore, in some cases, 2.5 representations are used instead: height maps. Height maps (also called height fields) are modelled as a 2D grid with an elevation parameter along the third dimension (see Figure 3.6). They are an efficient representation for surfaces that don't require the modelling of overhanging structures. In many cases, the height map is transformed into a mesh using algorithms such as the one proposed by Garland *et al.* [Garland, 1998].

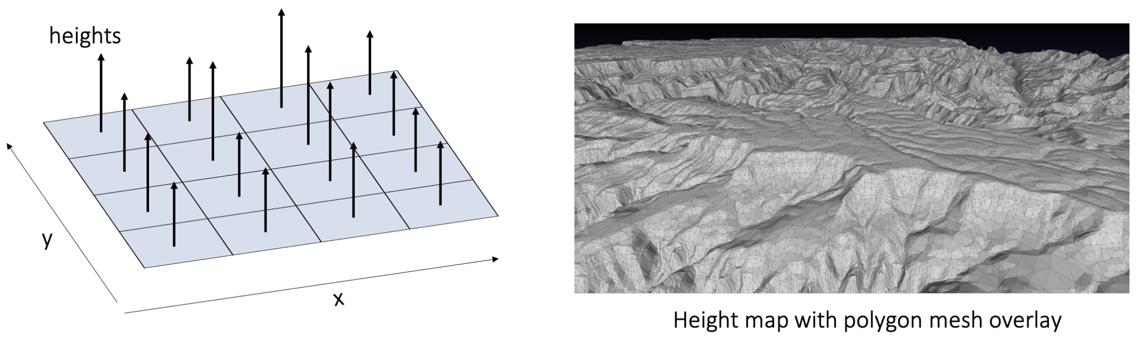


Figure 3.6: **Left**: Illustration of a height map. **Right**: Example of terrain modelled using a height map with mesh overlay [Foegleman, 2014].

When overlaying a heightmap with a mesh structure, a principled way to perform calculations (e.g interpolations) on the surface is by using *barycentric coordinates*. Given a point P that lies within the area of a triangle defined by vertices A, B and C, the position of P can be defined in terms of the subtriangle area ratios w, v and u formed by P and the sides of the triangle: $P = wA + uB + vC$ (see Figure 3.7). w, u and v are called the barycentric or areal coordinates and since they are defined as the ratios of the subtriangle and the main triangle areas, they are normalised, such that $u + w + v = 1$.

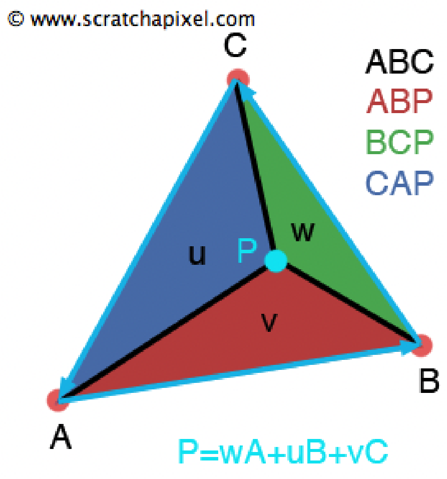


Figure 3.7: Illustration of Barycentric coordinates.

3.3 Machine learning and deep learning

With the explosion of interest and progess in AI in the past decade, many computer vision algorithms are increasingly combined with or even replaced by machine learning and in particular, deep learning methods. Below we provide a technical overview of the methods relevant to the research in this thesis.

Machine learning is the process of automatically adapting the parameters of a model based on an external signal, usually provided through evaluating the error a model is making with respect to an expected output. This process is usually referred to as *training* or *learning* and the final goal is to have a model which is able to predict correct outcomes for previously unseen datapoints. The main ingredients of ML algorithms are 1) the model to process the data by mapping an input value x to an output value y through a set of (usually non-linear) functions $y = f(x)$, 2) training data to *train* the network, 3) a cost function (often called loss function) to evaluate the performance of the model during training, 4) a training scheme which contains the details of the training procedure and 5) an evaluation metric to evaluate the model at deployment.

Machine learning models include a wide range of different methods, from clustering (K-Means, spectral clustering, DBSCAN), over kernel methods (radial basis functions, support vector machines) to graphical models which include decision trees, random forests, bayesian networks and neural networks. However, these are only coarse categories and many sub-categories exist, as well as combinations.

Although there is no one official definition, *deep learning* is commonly used to describe the machine learning algorithms designed for deep neural networks — neural networks with multiple hidden layers.

3.3.1 Neural networks

Since the machine learning work in this thesis is based on neural networks, this section will introduce the basic theory and the specific neural network models that are trained. For a detailed explanation of other machine learning models, [Bishop and Nasrabadi, 2007] and [Murphy, 2012] are excellent references.

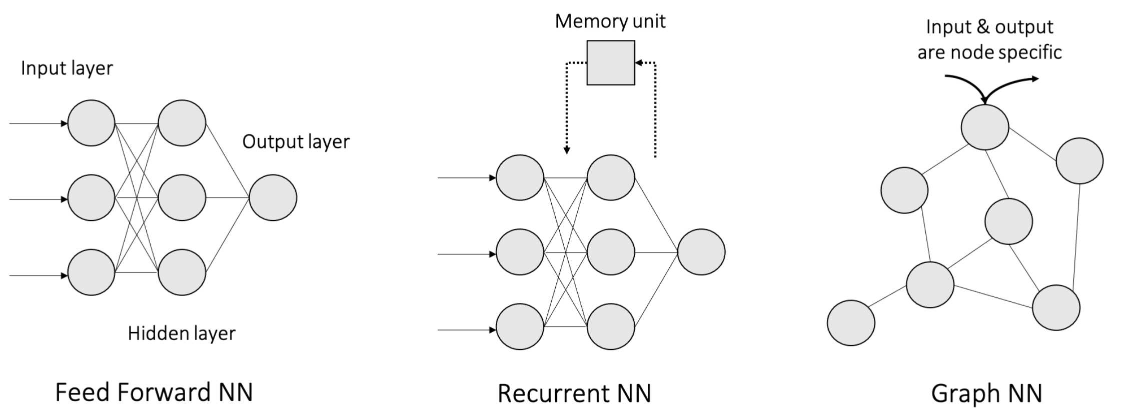


Figure 3.8: Neural network architecture model diagrams. **Left**: feed forward neural network model with one hidden layer and one output neuron. **Centre**: recurrent neural network model with one hidden layer with memory unit. **Right**: graph neural network model.

Neural networks are one of the most versatile machine learning models and have gained particular popularity in recent years in the context of deep learning research. Their structure is loosely inspired from the neural connections inside our brains and generally, they consist of a collection of neurons and their interconnections, which have learnable weights. One can broadly separate neural networks into the categories of feed-forward, recurrent and graph neural networks (see Figure 3.8), which each process the data in a different way. Feed-forward networks process the input data sequentially to produce an output, while recurrent networks repeatedly pass data through the same neurons, making use of memory units and combine stored information with new input data. Graph neural networks on the other hand, can process data in parallel and using more complicated rules based on message passing mechanism and local neighbourhoods. For the research of this thesis, feed-forward models were used (in particular CNNs and VAEs) which will be introduced in more detail below.

Convolutional neural networks (CNN)

CNNs are feed forward networks which are designed to process image data through a set of discrete 2D convolution operations by a 2D kernel K of dimension $m \times n$ on the input image I defined as:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n), \tag{3.4}$$

for a specific pixel location (i,j) [Goodfellow et al., 2016]. In practice, this operation computes a weighted average of the neighbourhood of a pixel and the pixel itself, based on the weights of the kernel K. During a forward pass through the network, multiple kernels slide over the input and repeatedly compute this convolution. The training process optimises the kernel weights that become sensitive to features such as colours, corners and edges in earlier layers, while deeper layers learn higher level, more abstract and visibly less interpretable features. The design of CNNs allows for location invariance; because the kernel weights are repeatedly applied to all image regions, patterns can be detected independent of their location. This has made the architecture particularly suitable for visual perception tasks.

Relevant architecture details Most CNN architectures use additional features such as *pooling* operations which downsample the input image based on computing the maximum or the average value in an image patch. Pooling effectively widens the *receptive field* of the network — the area it can use to extract features — and allows for more context integration. Alternatively, *strided* or *dilated* convolutions [Yu and Koltun, 2016] can be applied. Other common architecture extensions include *residual connections* [He et al., 2016], which allow for identity mappings between layers $l_{i+1} = \mathbf{I}^T l_i$ and address the vanishing gradient problem in very deep networks (see Figure 3.9 for more details).

All operations of 2D CNNs can be extended to 3D, by adding an additional dimension. While mathematically trivial, the extension to 3D incurs a high memory cost, since the input size and the respective computations grow cubically.

The original CNN architectures were designed for tasks such as image classification, where image features are extracted and processed to predict the type of flower or animal present in an image. The general CNN architecture for classification consists of successive convolutional layers with downsampling components such as pooling operations or strided convolutions and a final fully connected layer that maps the extracted features to a vector of class probabilities. To date, a large number of varieties have been developed from this original architecture, including *fully convolutional networks* (FCN), designed for the per-pixel prediction that is necessary for semantic and instance segmentation tasks. The original FCN architecture [Long et al., 2015] proposed a modified CNN architecture where the extracted and downsampled features are brought back to the original resolution through deconvolutional or upsampling (interpolation) layers (see Figure 3.10) for more details). To date, FCN's exist in different variants, including 3D versions that predict semantic voxels [Song et al., 2017]. A crucial component for the prediction quality of most FCN architectures is the addition of skip-connections, first proposed by [Ronneberger et al., 2015]. Skip connections connect the early encoding layers with the late decoding layers, by skipping the downsampling layers which generate informative but coarse features, that lack representation detail.

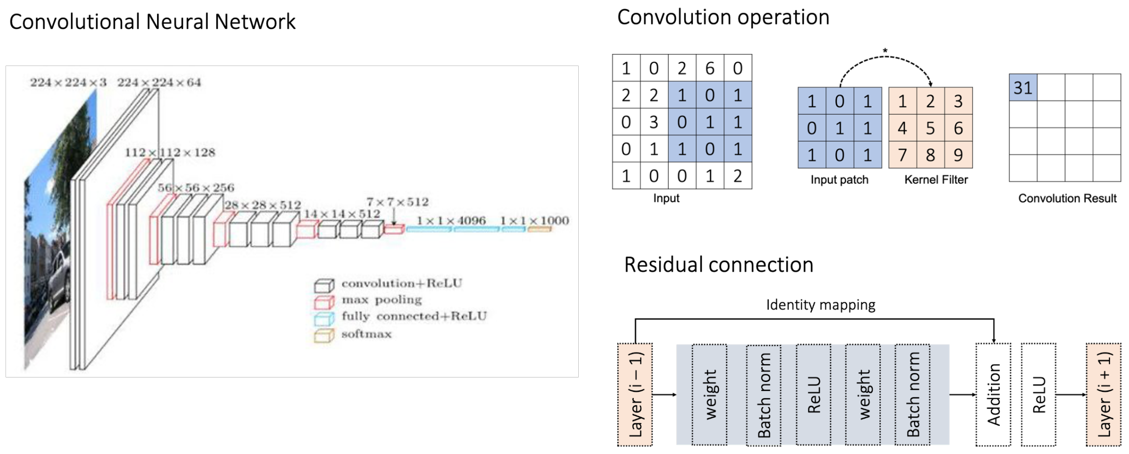


Figure 3.9: **Left**: A standard CNN architecture (the VGG-16 [Simonyan and Zisserman, 2015] network) [Gebrehiwot et al., 2019]. **Top right**: A convolutional operation on an input layer of a CNN. **Bottom right**: The architecture of a residual connection.

Variational autoencoders

Variational Autoencoders (VAE) are part of the class of *generative models*, which aim to learn the underlying distribution of a dataset in order to generate new examples from the distribution. *'A generative model simulates how the data is generated in the real world'* [Kingma and Welling, 2019]. Generative models are not necessarily different in architecture, but in the task they learn and in their training algorithm. While the goal of discriminative models is to learn a function mapping $f : x \rightarrow y$ and to be able to predict the correct result given previously unseen input data, the goal of generative models is to be able to generate new results, by sampling from the distribution that was learnt from the training data: $f : y \sim \mathcal{N}_\theta(\mu, \sigma^2)$. This form of learning is also called *(unsupervised) representation learning*, for which variational autoencoders provide a principled framework [Kingma and Welling, 2019]. VAEs are so-called likelihood-based generative models and are considered the fusion of graphical models and deep learning [Kingma and Welling, 2019].

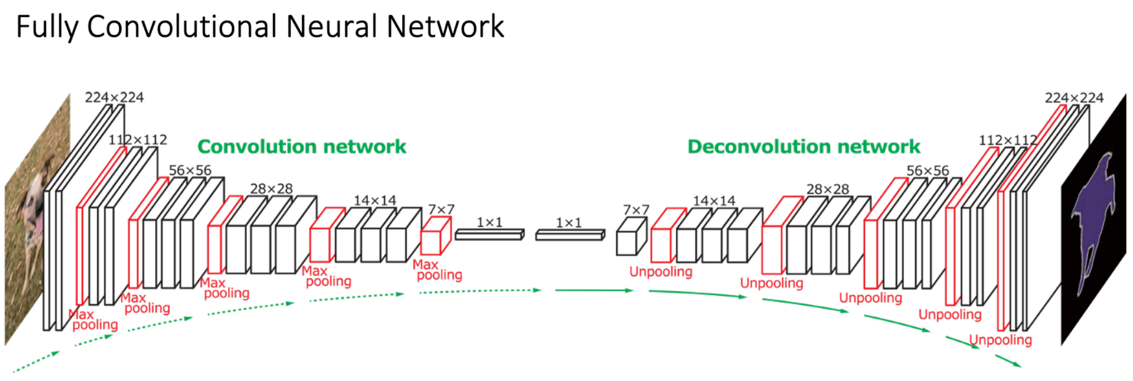


Figure 3.10: The architecture of a fully convolutional network for semantic segmentation with a mirrored encoder and deconder structure [Noh et al., 2015].

They are also considered a special type of deep latent variable model. Their theoretical framework is introduced in more detail below.

Autoencoders are a special type of deep neural network that learn how to compress data in the most informative (least lossy way) in order to then restore it as accurately as possible. Their architecture has a characteristic bottleneck through which the data has to pass during compression, usually referred to as latent code or latents. Compared to classical dimensionality reduction methods such as principal component analysis, autoencoders provide non-linear compression functions and instead of using eigendecomposition, the most important and descriptive features are learnt from data. They can be used for a variety of applications such as feature extraction for other downstream applications, image denoising, sequence to sequence prediction for NLP applications and, when combined with variational inference, they can be trained as generative models, e.g. for natural image generation.

Variational inference is a method for approximate inference in latent variable models with x observed variables and z unobserved (latent) variables. The goal is to find the posterior $p(z|x)$, i.e. estimating the latent variables from the observed ones. $p(z|x)$ could be computed with Bayes' rule:

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}, \tag{3.5}$$

but the *marginal density* $p(x)$, also called evidence, is usually intractable. Hence, one uses approximate inference techniques such as sampling based methods or variational methods.

Sampling based methods, such as such as MCMC, can be slow to converge, it is difficult to tell when they converge and they require carefully chosen sampling techniques. Variational methods on the other hand, scale better and provide an exact bound on the accuracy of the approximation. They cast inference on an intractable distribution $p(z|x)$ as an optimisation problem over a family of tractable distributions over the latent variables $\mathscr{Q}$. The goal is to find a distribution within the family $q(z) \in \mathscr{Q}$ which is similar to the true distribution p. To measure the similarity between the target posterior $p(z|x)$ and the variational approximation $q(z)$, the Kullback-Leibler (KL) divergence is used. It finds the "distance" between two probability distributions by measuring the difference in information contained in both distributions. The KL divergence between two distributions p and q with discrete support is defined as:

$$KL(q||p) = \sum_x q(x) \log(\frac{q(x)}{p(x)}). \tag{3.6}$$

However, how can $KL(q||p)$ be evaluated if p is unknown? Variational inference uses the variational lower bound, also called the evidence lower bound (ELBO). The ELBO is a bound on the probability of the data and can be derived either using Jensen's inequality or from the KL divergence itself (Equation 3.6). Starting from the KL divergence between $q(z)$ and $p(z|x)$, one can use Bayes' rule, to obtain:

$$KL(q(z)||p(z|x)) = -(\mathbb{E}_q[log(p(z,x))] - \mathbb{E}_q[log(q(z))]) + \log(p(x)), \tag{3.7}$$

where $\log(p(x))$ is the log marginal likelihood (also called *log evidence*) of the observed variables x. This can in turn be rewritten as:

$$\log(p(x)) = -(\mathbb{E}_q[log(p(z,x))] - \mathbb{E}_q[log(q(z))]) + KL(q(z)||p(z|x)) \tag{3.8}$$

Since it holds that $KL(q(z)||p(z|x)) \geq 0$, $-(\mathbb{E}_q log(p(z,x)) - \mathbb{E}_q log(q(z)))$ is essentially the lower bound $\mathscr{L}$ for $\log(p(x))$. It follows that by maximising $-\mathscr{L}$, one indirectly minimises the KL divergence between the true posterior $p_\theta(z|x)$ and the estimated posterior $p(z|x)$. At the same time, the marginal likelihood $p(x)$ of the data is maximised. The optimisation to approximate distribution q with variational parameters ϕ can be written as:

$$q_\phi(z) = \underset{q(z) \in \mathscr{Q}}{\operatorname{argmax}}(\mathbb{E}_q[log(p(z))] - \mathbb{E}_q[log(q(x,z))]) \tag{3.9}$$

Variational autoencoders (VAE) provide a framework to do variational inference on the distribution which generated a particular target dataset using deep neural networks. They consist of an encoder-decoder structure which compresses the input into a latent code. The encoder network (also called recognition network) models a conditional bayesian network of the form $p(z|x)$, while the generative decoder models another Bayesian network as $p(z)p(x|z)$. Both conditionals are parameterised through the neural network weights, which are optimised during training. The learning algorithm of VAEs is a combination of variational expectation maximisation and stochastic gradient descent (SGD) — it does stochastic gradient-based optimisation of the ELBO. To this end, gradients of the ELBO with respect to both the encoder parameters ϕ and the decoder parameters θ have to be computed. This is achieved with the help of the **reparameterisation trick** (the interested reader can refer to [Kingma and Welling, 2019] for a detailed derivation), which allows for the gradients to be computed with a simple Monte-Carlo estimator. In practice, the encoder maps the input data samples to a latent space, the conditional distribution $p_\phi(z|x)$, parameterised with variational parameters ϕ. The generative decoder samples from the estimated distribution $p(z)$ and generates new datapoints x from that distribution.

Training In order to train the VAE, the ELBO optimisation is reformulated in terms of a reconstruction loss and a KL divergence component. Equation 3.9 can be rewritten as:

$$q_\phi(z) = \underset{q(z)\in\mathscr{Q}}{\operatorname{argmin}}(\mathbb{E}_{q_\phi}[log(p_\theta(x|z))] - \mathbb{E}_{q_\theta}[\frac{p_\phi(z|x)}{p_\theta(z)}]) \tag{3.10}$$

where $\mathbb{E}_{q_\phi}(log(p_\theta(x|z))$ is the reconstruction error and $\mathbb{E}_{q_\theta}(\frac{p_\phi(z|x)}{p_\theta(z)})$ is the KL divergence between the estimated posterior and the prior over the latent variables z [Kingma and Welling, 2019].

In practice, training data examples are encoded into a mean μ and variance σ^2, parameterising the distribution $p(z|x)$. Both mean and variance are modelled with a set of continuous values, a latent code. The next step is then to sample from that encoded distribution: $z \sim \mathcal{N}(\mu, \sigma^2$. The resulting sampled latent code is decoded using the generative decoder to yield a reconstruction example. The reconstruction is evaluated with a reconstruction loss whose error is backpropagated as in discriminative learning methods. The second component of the loss is the KL divergence term which ensures that the estimated posterior $p(z,x)$ remains close to the latent prior $p(z)$.

3.3.2 Cost functions and evaluation metrics

Although cost functions can be different for every task, for supervised learning a few commonly used cost functions exist. These include the mean squared error (MSE) loss and the cross entropy loss (CE). The MSE loss, defined as $\frac{1}{m}\sum_m ||y_m - \hat{y}_m||^2$ simply calculates the euclidean distance between predicted value $\hat{y}$ and true value y and is commonly used for regression tasks. Classification tasks on the other hand, are usually trained using cross entropy, which computes the distance between two probability distributions (the probabilities of the predicted and ground truth classes). In information theoretical terms, cross entropy computes the amount of additional bits required to encode a signal when using the predicted probability distribution $\hat{\sigma}$ over the true distribution σ. If the task is to differentiate between 2 classes only, binary cross entropy is used, defined as $-(y\log(p) + (1-y)\log(1-p))$, where p and y are the predicted and true probabilities of the first class being present and $(1-p)$ and $(1-y)$ are the predicted and true probabilities of the second class being present. For multi-class problems, this is generalised to: $-\sum_c y_c \log p_c$.

Similarly to cost functions, suitable model evaluation metrics depend on the prediction task. For semantic and instance segmentation, a common evaluation metric is the intersection over union, also called the jaccard index defined as J(A,B) = $\frac{|A \cap B|}{|A \cup B|}$, where A and B are the quantities between which the amount of overlap is to be found. In practice, for pixel-wise class predictions in image segmentation tasks, one computes the intersection over union between all predicted class pixels and all true class pixels for one specific class (see Figure 3.14 for a visual example).

In 3D, some representations such as voxelgrids lend themselves to cost and evaluation metrics such as MSE or CE. For some representations such as pointclouds and meshes however, which have less structure, different metrics are used. One commonly used metric is the *Chamfer Distance*, which calculates the distance between two sets of 3D points P_1 and P_2. For every $x \in P_1$ the nearest neighbour is found in P_2. This is done in both directions, $P_1 \rightarrow P_2$ and $P_2 \rightarrow P_1$:

$$d_{CD}(P_1,P_2) = \frac{1}{P_1}\sum_{x_i=0}^{P_1}\min_{x_j \varepsilon P_1}(\mathbf{x_i}-\mathbf{x_j}) + \frac{1}{P_2}\sum_{x_j=0}^{P_2}\min_{x_i \varepsilon P_2}(\mathbf{x_j}-\mathbf{x_i}) \ . \tag{3.11}$$

[Wu et al., 2021]. Although defined for 3D points, the Chamfer Distance can be used to approximate the distance between meshes, by sampling sets P_1 and P_2 from the mesh surface.

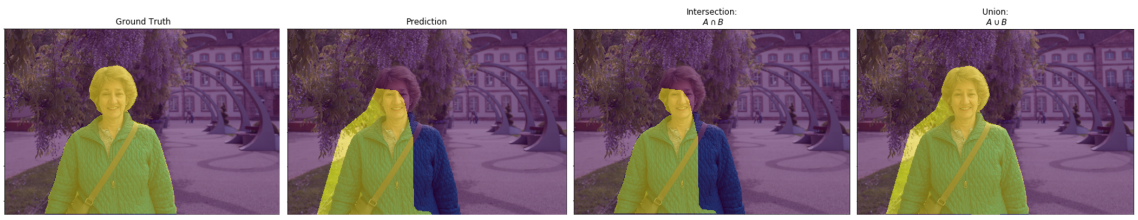


Figure 3.11: An illustration of intersection over union for semantic segmentation. [Jordan, 2018]

3.3.3 Training (optimisation) of neural networks

The way ML algorithms learn depends on how they experience data. Many different approaches exist, but they can be loosely grouped into supervised, unsupervised and reinforcement learning methods. The research in this thesis uses supervised learning methods, which are introduced in more detail below.

Supervised learning

Supervised learning consists in optimising the model parameters based on external supervision - provided through data. This supervision requires labelled training data - for a specific input x, the predicted output $\hat{y}$ is compared to the true value y using a cost function. The discrepancy between predicted and true result, the error, is used to optimise the model's parameters in such a way that the next prediction will be better (Section 3.3.3 gives a more detailed explanation on this process). In supervised learning one differentiates between *training data*, which is used top optimise the model, validation data used to assess when the model has *generalised* - is able to generate good predictions for previously unseen data points - and *test data*, used to evaluate the fully trained model.

Stochastic gradient descent

To train a network, its parameters have to be optimised with respect to a calculated cost. Second order optimisation methods are computationally expensive to apply to neural networks [Bishop, 2006] and therefore, first order optimisation is used — gradient descent. Gradient descent minimises a given objective $J_\theta(x)$ by computing the first-order gradient and stepping in the negative direction of this gradient: $\theta = \theta - \alpha_\theta \mathbb{E}[J_\theta(x)]$, where the expectation $\mathbb{E}[J_\theta(x)]$ is approximated by evaluating cost and gradient over the full training set [Maas, 2013] and the learning rate α determines how far the algorithm steps into the gradient direction. In practice it is not feasible to evaluate the full training set at once and neural networks are usually optimised using *stochastic* gradient descent (SGD). SGD computes the gradient with respect to the parameters using only a few training example pairs (x_i, y_i) at a time:

$$\theta = \theta - \alpha_\theta (J_\theta(x_i, y_i)) \tag{3.12}$$

The back-propagation algorithm

All training schemes which optimise neural networks, rely on *back-propagation*, an automatic differentiation algorithm which calculates the gradients for the weights in a neural network graph structure. It is used to perform stochastic gradient descent to minimise the error of the network output with respect to some defined loss function. Specifically, using partial differentiation, the contribution of each model parameter to the model prediction error is determined.

For a feed-forward network architecture, each unit computes a weighted sum of its inputs: $a_j = \sum_i w_{i,j} z_i$, whereby z_i is the activation of the previous unit (or the input). Biases can be included in this sum [Bishop, 2006]. For the error E_n, the gradient with respect to a weight w_{ji} can be written as $\frac{\delta E}{\delta w_{ij}} = \frac{\delta E}{\delta a_j} \frac{\delta a_j}{\delta w_{ij}}$, which can in turn be rewritten as $\frac{\delta E}{\delta w_{ij}} = \delta_j z_i$. $\delta_j = \frac{\delta E}{\delta a_j}$ is defined as the *error* of unit j. To obtain the derivative at a particular weight, the error δ_j has to be computed for all hidden and output units and be multiplied with their respective input z_j. The δ's at hidden units can be obtained through applying the chain rule: $\delta_j = \frac{\delta E}{\delta a_j} = \frac{\delta E}{\delta a_{k_0}} \dots \frac{\delta a_{k_n}}{\delta a_j}$ for any n layers between j and the output. Expanding with some further manipulation [Bishop, 2006], the general rule for back-propagation can be written as:

$$\delta_j = z_j \sum_k w_{kj} a_j \tag{3.13}$$

3.4 Visual perception with deep learning

This section introduces the visual perception components relevant for this thesis from a deep learning perspective. In particular, we describe how each task is defined and solved using deep neural networks.

3.4.1 Object detection

Object Detection consists in locating objects in an observed region of the scene. This includes estimating the volumetric extents of the object as well as its position and orientation with the help of *bounding boxes*. In deep learning, object detection consists in predicting a set of continuous values describing these bounding boxes from an input image or a 3D scene. These values are usually the object class and the width, height and center point of the bounding box, as well as its orientation when predicting in 3D. Deep learning approaches either employ single-stage or two-stage detection algorithms to produce these object descriptors. Two-stage detection methods such as the R-CNN family (Fast R-CNN [Girshick, 2015], Faster R-CNN [Ren et al., 2015] and Mask-R-CNN [He et al., 2017]) first generate a set of *region proposals*. For a set of extracted (and downsampled) feature maps of the input scene, and a predefined set of *anchors* (a set of bounding box candidates centered around every pixel of the feature map), a region proposal network (RPN) predicts which candidate holds an object. In a second step, the bounding box parameters and class label for every extracted region are refined. One-stage methods such as YOLO [Redmon et al., 2015] predict a fixed number of object descriptors for every pixel of the downsampled image and use an object existence probability p_{obj} to determine whether or not this pixel actually contains an object (see Figure 3.12). Both two-stage and one-stage methods employ *non-maximum suppression* to suppress multiple predictions per object. Non-maximum suppression selects highly overlapping bounding boxes and removes all those except for the one with the highest p_{obj}.

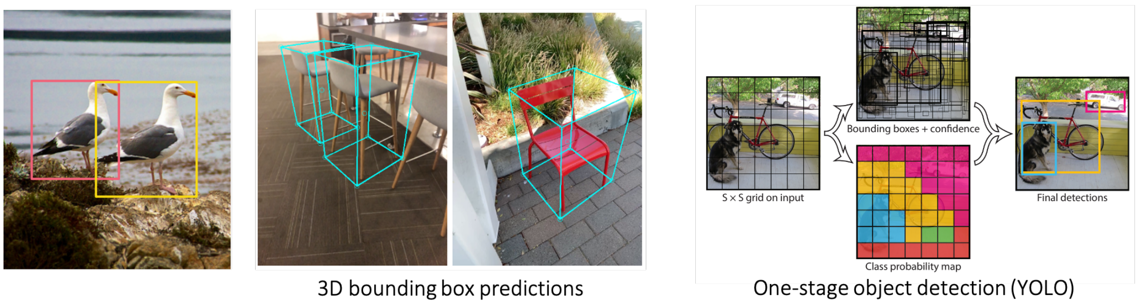

3D bounding box predictions
S × S grid on input
Bounding boxes + confidence
Class probability map
Final detections
One-stage object detection (YOLO)

Figure 3.12: **Left**: an object detection example in 2D from the DETR model [Carion et al., 2020]. **Centre**: an object detection example in 3D from Objectron MediaPipe. **Right**: illustration of the YOLO single-stage object detection pipeline [Redmon et al., 2015].

3.4.2 Semantic and instance segmentation

Semantic and instance segmentation are pixel-level prediction tasks or, when labelling in 3D, each representation component (e.g. a voxel or a point) has to be annotated. While semantic segmentation does not differentiate between individual objects in a segmentation (multiple objects can have the same label if they belong to the same class), instance segmentation assigns a semantic label and instance label to every pixel or scene entity. In deep learning, semantic and instance segmentation are predicted using FCN-style architectures where the encoder extracts descriptive features from an image or scene and the decoder generates a class prediction per pixel. Specifically, for image segmentation, the network predicts an output vector per pixel **p**, which is passed through a softmax layer to generate a per-pixel or scene element class probability distribution of which the maximum is taken as the predicted class: $c_{pixel} = \text{argmax}(Softmax(\mathbf{p}))$. For instance segmentation, in addition to the semantic class, the output needs to differentiate between different objects. State of the art methods achieve this by either building on two-stage object detection methods (e.g adding a pixel-wise binary mask delimiting the object for every extracted region of interest [Kaiming et al., 2017]) or, by predicting per-pixel instance masks on a downsampled feature image [Wang et al., 2020b].

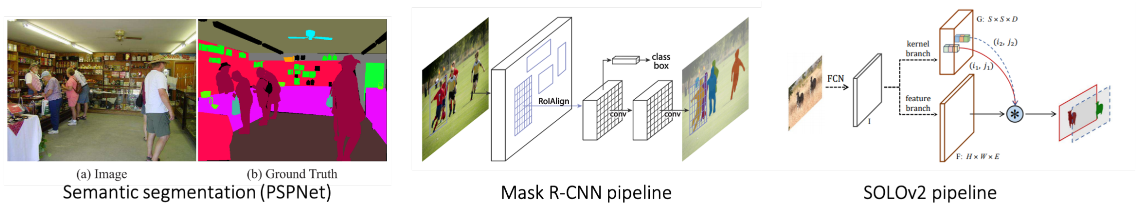


Figure 3.13: **Left**: semantic segmentation [Carion et al., 2020]. **Centre**: an object detection example in 3D from Objectron MediaPipe. **Right**: illustration of the YOLO single-stage object detection pipeline [Redmon et al., 2015].

3.5 Rendering

To obtain a 2D projection of 3D scenes, one has to use rendering techniques. In this thesis rendering is used for all projects and we introduce the theoretical basics on rendering methods and some relevant technical details below.

Rendering is the process of generating a 2D projection of a 3D scene that is represented in simulation. It essentially simulates the process of taking a photograph of that scene and usually, the goal of rendering is to generate a photorealistic image. It is a crucial component of the computer graphics pipeline, but also very present in computer vision in general - any 2D visualisation of 3D content requires the process of rendering. There are two main methods to render: *raytracing* and *rasterisation*. In raytracing, for each image pixel, a ray is sent out from the center of the simulated camera lens through the center of the pixel and traced along the direction of travel until it is found to intersect with a surface. At the point of intersection, information about the surface properties such as colour and brightness can be computed and used to fill the value of the corresponding pixel. Rasterisation methods on the other hand project triangular surface elements onto the image plane by first projecting the triangle vertices and then filling all pixels enclosed by the projected vertices with the values contained within the 3D triangle.

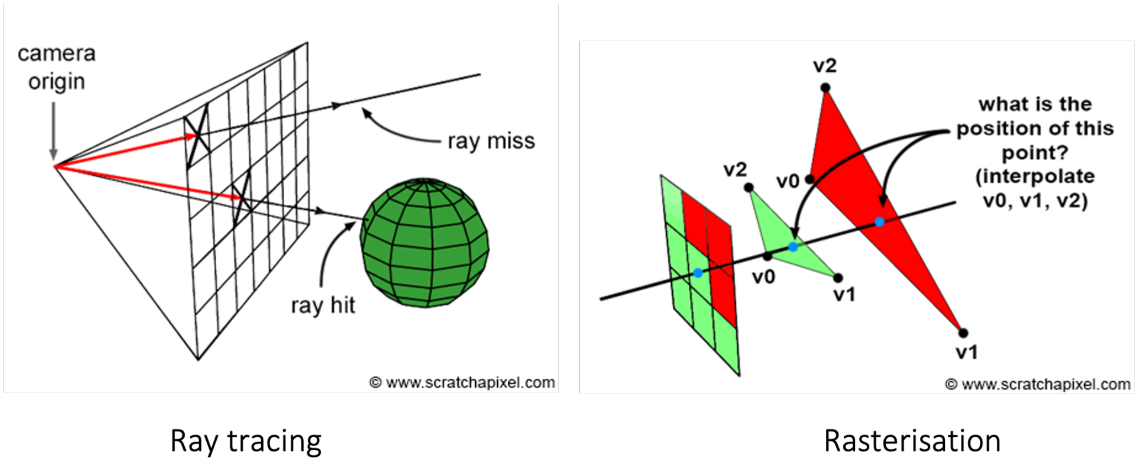


Figure 3.14: **Left**: an illustration of the raytracing operation. For every pixel, a ray is simulated from the camera origin into the scene and stepped along until a surface is encountered or the ray is found to exit the scene. **Right**: an illustration of Rasterisation. From a mesh representation of the object, vertices are backprojected onto their respective pixel locations.

3.5.1 Raytracing signed distance functions

The implementation of raytracing depends on the way the 3D scene is represented. For voxel-grid representations, one usually steps along each ray in pre-defined step sizes until the ray either exits the scene or intersects with a surface. In the former case, one registers a default value for the corresponding pixel and in the latter, the surface properties such as colour or the distance (in the case of depth rendering) is recorded at the respective pixel location. Given the discrete nature of voxelgrids, to obtain information about surface intersection at a particular point $p_{x,y,z}$ along the ray, the grid has to be interpolated. Then, intersection can be found according to some pre-defined thresholds. For occupancy grids, surface intersection occurs once the grid value is close to 1. For signed distance functions, the intersection occurs once the scene value changes sign. A particularly fast way of ray tracing for signed distance fields is sphere tracing [Hart, 1996]. Sphere tracing leverages the underlying structure of the SDF to find the surface intersection point faster. It is based on the insight, that for euclidean SDFs, at any point in space, one can trace a circle whose radius is given by the current value of the SDF, and there will be no surface intersection within that circle. Hence, at any point in space, one can safely step along the direction of the ray for at most the current value of the SDF (see Figure 3.15). Sphere tracing solves the problem of choosing the right step-size to reach the closest surface in the fastest possible way.

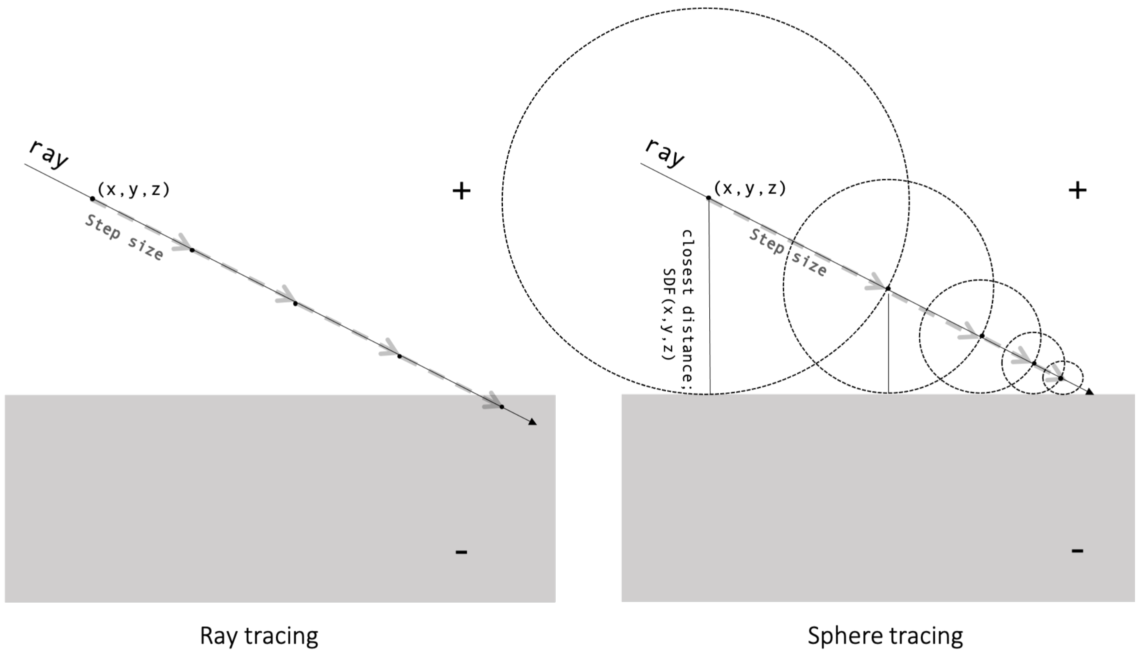


Figure 3.15: **Left**: simple raytracing. **Right**: sphere tracing [Hart, 1996].

3.5.2 Differentiable rendering

With the introduction of Machine Learning techniques to Computer Vision, it can often be useful to make the rendering process differentiable. This allows for the rendering function to be integrated into the overall optimisation and enables models which generate 3D scene representations to be trained based on image data. Differentiable rendering allows to compute the gradient $\frac{\delta p_{i,j}}{\delta \theta}$ of a rendered image pixel $p_{i,j}$ with respect to 3D scene parameters θ, which can be used in the optimisation process to adapt θ so that the rendered image matches a given target image. The parameters θ can be explicitly defining the scene, or, be encoded in the weights or latent representation of a neural network.

3.6 Software and technical details

To support the research presented in this thesis, experiments were implemented in Python, C++ as well as Cuda for parallelizing code. The deep neural networks were developed and trained using the open sourced machine learning frameworks Tensorflow and Pytorch, which will be introduced in more detail below. Some additional details on how the differentiable renderer used in Chapter 4 was implemented using Pytorch's autograd function are also provided.

3.6.1 ML frameworks

Tensorflow was first made publicly available in 2015 by developers at Google and became one of the first widely used open-source frameworks for development and end-to-end training of neural network models. In particular, it provides building blocks such as pre-implemented network layers which enable to build networks in a modular fashion. Furthermore, the forward and backward passes through a network model, including the backpropagation step are implemented as part of the framework itself and can be initiated with a simple function call. The data is packaged into *tensors*, a datastructure that extends matrices to an arbitrary number of dimensions. When building a network model, Tensorflow first creates the *computation graph*, which defines all variables, constants and computations which will be used - a series of Tensorflow operations arranged as a graph. Once the graph is defined, it is launched into a *session* which runs all operations as they were defined. This particular structure of the Tensorflow pipeline allows for parallelism and makes training particularly efficient. For more details, please refer to the official research paper [Abadi et al., 2016].

Pytorch was developed at Facebook (now Meta) and was first released in 2017. Similarly to Tensorflow, it provides an end-to-end framework for building and training neural network models. However, compared to Tensorflow, it's computation graph is *dynamic*, i.e. it is created on the fly and variables inside the graph can be easily accessed. This makes code that was built with this framework easier to debug. Furthermore, Pytorch integrates direct support for the popular python library *numpy*, which provides accelerated functionalities for array and matrix operations and is widely used in the machine learning community. Another advantage of Pytorch is it's integrated data parallelism, making it very easy to implement distributed training.

3.6.2 Differentiable rendering with Pytorch

Automatic differentiation Pytorch's framework uses *autograd*, an automatic differentiation system, which builds a directed acyclic graph of all computations (registered to autograd) within a network and produces the gradients according to the chain rule. Essentially, it is a module dedicated to computing the jacobian matrices between tensors that are registered as variables inside the graph. For operations which are non-differentiable, but should be integrated into the system's gradient flow (i.e should be part of the optimisation), one can extend *autograd* through module inheritance and manually define the forward and backward passes through the operation using the forward() and *backward()* functions respectively. Autograd functions save the variables for which gradients are computed inside a context manager *ctx*, which allows for them to be accessed during the backward pass. Please refer to the official Pytorch documentation for a more detailed explanation.

To implement the differentiable renderer used in Chapter 5, and train it together with the deep neural network (see Chapter 5), Pytorch's autograd system was leveraged and extended by manually defining the backward pass of the rendering method. To this end, the rendering functionality was writtted inside a Pytorch module that inherits from *pytorch.autograd.Function*. The forward method performs TSDF raytracing (see Section 5.3.3), which is parallelised on the GPU. In the backward pass, the gradients of the rendered image with respect to the scene parameters have to be computed. For our particular method, the scene parameters θ are defined by the latents $\mathbf{l}$ of a deep neural network that generates the discrete SDF voxelgrid from which depth images are rendered. To make the rendering process differentiable, the error on the rendered images has to be backpropagated to the voxel centers of the generated SDF. The remaining backward propagation has to go through the network, which is taken care of by Pytorch's integrated autograd function. In the implementation of the backward pass through the rendering function, the computed rendering error (see Section 5.3.3 for details on the error function) is accumulated at each voxel centre. As surface intersections of the rays don't usually coincide with the SDF voxel centers, the error at every intersection is propagated to the neighbouring voxel centers using the respective weights obtained from trilinear interpolation $w_{i=0,\dots i=8}$. Similarly to the rendering process itself, we parallelise this process on the GPU for every ray and use CUDA's atomic functions to ensure sequential reading and writing for individual voxel centers. An illustration of our implementation can be found in Figure 3.16.

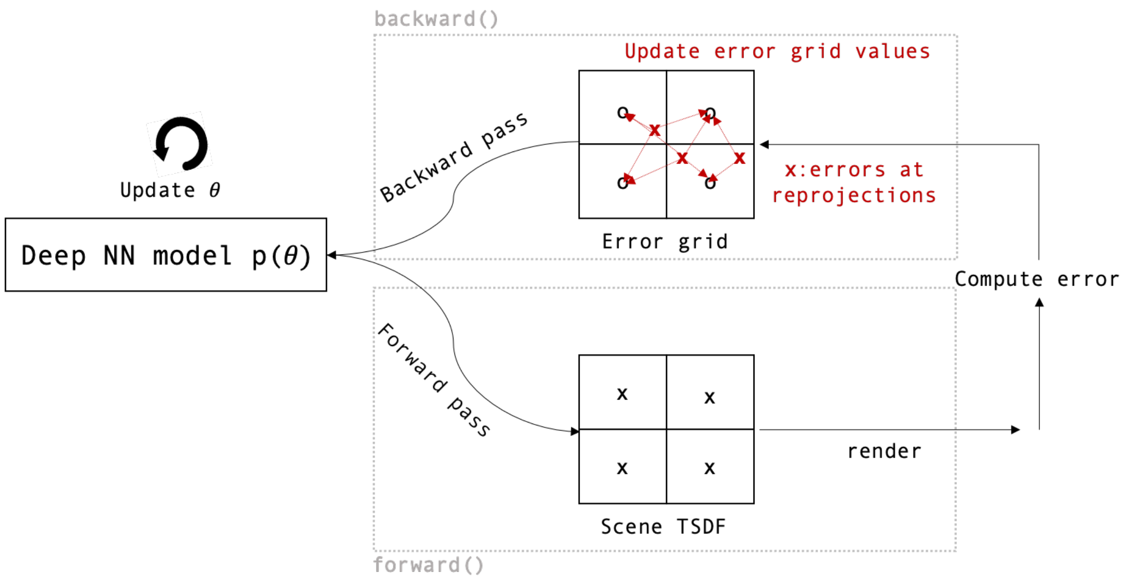


Figure 3.16: Illustration of how the differentiable renderer is implemented within Pytorch's autograd framework, by manually specifying the forward() and backward() functions.

CHAPTER 4

Comparing view-based and map-based semantic labelling in multi-view scenes

Contents

4.1 Introduction

As the cameras carried by robots or other smart devices move through scenes, their ability to operate and interact intelligently and persistently with their environments, will depend on the quality of scene representation which they can build and maintain [Davison, 2018, Cadena et al., 2016a]. But what defines a high quality scene representation? A scene can be built with several levels of information, each adding a different layer of abstraction. The most basic representation differentiates occupied from free space with an occupancy map. This allows for navigation and path planning of autonomous agents, but holds no semantic information about which areas of occupied space constitute objects, floor or ceiling. This requires semantic labels, which allow for more sophisticated navigation and planning algorithms. For instance, semantics can inform an autonomous driving system about regions which are road and pavement to detect navigable surfaces. While semantics describe the type a scene element has, it doesn't inform about ”objectness”. Recognising ”objectness” (in form of bounding boxes or instance segmentation) introduces a further level of understanding that is necessary for interactive tasks such as grasping and manipulation. Such tasks will form the basis of autonomous systems designed to support humans with tasks at home or in industrial settings, that go beyond spatial navigation. Finally, understanding and representing object relationships and affordances, as well as dynamic scenes in real-time, will allow autonomous systems to operate independently in most indoor and outdoor settings.

4.1.1 Motivation

Although some research has advanced in the direction of dynamic scene understanding [Rünz and de Agapito, 2017, Xu et al., 2019a] and instance segmentation of 3D scenes [Grinvald et al., 2019, Li et al., 2020], reliable and efficient semantic annotation of 3D space remains challenging, even when a large amount of views of the scene are available and with state of the art real-time mapping systems [McCormac et al., 2016]. Moreover, no guidelines for how to add semantic labels to 3D reconstructions exists and state-of-the art methods use a variety of approaches, which can loosely be classified into *view-based* semantic labelling, *map-based* semantic labelling and *hybrid approaches* (defined below). To our knowledge, the choice of approach is rarely based on quantitatively determined advantages for the particular use-case. This chapter is concerned with the question of how to best add semantic segmentation labels to a 3D reconstruction, when we have access to a real-time SLAM system and an arbitrary number of views. We aim to establish a baseline for comparison of view-based and map-based approaches to open up systematic and quantitative research on their use within real-time SLAM.

In particular, we aim to provide a principled study in which we compare a view-based and a map-based approach to add semantic labels to a scene reconstruction. In the context of our selected experimental setting (see below), we compare the advantages and disadvantages of both methods to provide insights into which approach can be more suitable depending on the system-requirements (e.g. are ad-hoc labels required for early semantic understanding?) and components (e.g., the semantic labelling accuracy of the neural network).

4.1.2 Semantically annotating 3D space

View-based In most state of the art methods, semantic labels are generated in image space to be then re-projected into the 3D map during SFM or SLAM as an overlay of the geometric representation, whereby each quantum of the 3D scene representation (a voxel of the voxelgrid or a point in the pointcloud) is augmented with a label vector. As each scene element is seen from multiple varying viewpoints, the independent image labels are combined using incremental fusion methods. Considering label correlations during fusion is computationally expensive and individual pixel labels are thus generally assumed to be independent. Independent pixel-wise label fusion of every frame or keyframe during reconstruction offers a scalable solution and these incremental updates contribute to robust estimates and the incremental correction of errors. The labelling CNN runs on raw data obtained directly from the camera, and can run at the sensor's natural resolution. On the downside, frame-wise label-fusion can incur unnecessary computation in areas which are easily segmented from one view. View-based approaches are generally based on well known image segmentation methods such as random forest classifiers [Hermans et al., 2014, Stückler and Behnke, 2014], or depth segmentation based on normal angles [Tateno et al., 2015] to generate semantic maps. McCormac *et al.* [McCormac et al., 2017b] introduced CNNs to incremental semantic fusion, obtaining semantic labels from RGB frames with a 2D CNN. Taking a slightly different direction, Ma *et al.* [Ma et al., 2017a] propose a self-supervised multi-view prediction method and Xiang *et al.* [Xiang and Fox, 2017] use a recurrent neural network to perform frame-wise segmentation of a sequence of RGB-D frames obtained from KinectFusion [Newcombe et al., 2011a]. Recently, approaches such as PanopticFusion [Narita et al., 2019] and [Grinvald et al., 2019] extend incremental semantic fusion to object discovery and instance-aware maps.

Map-based An alternative approach to annotating 3D space with semantic labels is to directly process the 3D representation: a single labelling network is applied to the whole reconstruction produced by a SLAM system, which contains both geometry and appearance information such as colour. This approach avoids the redundant work of labelling many overlapping input frames, and can be maximally efficient by operating on each scene element only once. Map-based methods can furthermore take advantage of global context over the whole scene when labelling each part and avoid the need for element-wise label fusion approaches which generally neglect label correlations. Finally, a network which learns and labels in canonical map space has much less scene variation to deal with due to rotation and scale changes. However, its power will be limited by the quality of the map reconstruction. Labelling the 3D representation of a scene generally involves label inference from dense 3D representations such as SDFs or voxel grids, which are known to be expensive to process. Hence, approaches for semantic, per-element labelling 3D representations have mainly been put forward for single objects [Maturana and Scherer, 2015, Wu et al., 2015]. Recently, several have focused on representing and processing 3D data more efficiently [Wurm et al., 2011, Riegler et al., 2017, Wang et al., 2017, Yu and Koltun, 2016]. Although most approaches focus on voxel grids, some approaches for classification of point clouds have been explored [Qi et al., 2017]. Efficient 3D labelling at large scale, however, remains unsolved and only few have ventured into labelling a variably-sized reconstructed 3D scene. Landrieu *et al.* [Landrieu and Simonovsky, 2018] extended the idea of superpixels to 3D point clouds and proposed a superpoint method to label large scale LIDAR scans. Dai *et al.* [Dai et al., 2018] proposed a fully convolutional, autoregressive, hierarchical coarse-to-fine 3D network to produce semantic labels together with geometry completion for a large 3D voxel grid scene. However, due to the expensive 3D nature of their input, the different levels of hierarchy in their network have to be trained separately. Roddick *et al.* [Roddick et al., 2018] project image features into an orthographic 3D space using a learned transformation, which removes the scale inconsistency, and creates a feature map with meaningful distances and without projective distortions of object appearance. They improve the efficiency of object detection from the orthographic map by collapsing voxel features along the vertical axis and then process the entire map of $80m \times 80m$ at a grid resolution of $0.5m$ at once. However, they do not address scalability in their method.

A few **hybrid approaches** also exist, which use semantics from multi-view image information and combine it with 3D geometric features for increased detail [Dai and Nießner, 2018, Hou et al., 2019].

4.1.3 Machine learning methods for processing 3D representations

Compared to 2D images whose representation is usually fixed (a regular pixel grid), 3D space can be represented in a variety of ways (see Section 3.2). For each representation, the scene labelling network required to process and extract information can vary in architecture, scalability and capacity.

When we capture and represent the world around us, the predominant format are images, optionally with a temporal dimension for videos. The variety of representing these 2D projections of space is limited - the projected information is captured on a regular grid in form of pixels composing the image. Most commonly, this image contains colour or depth information, although some alternatives exist such as events (which encode changes in the scene) captured by event cameras. The precision of the representation is determined by the image resolution and the memory requirements of image data scale quadratically with resolution. Machine learning methods to process information held in this representation have developed rapidly and in some areas such as image classification, are surpassing human level performance [He et al., 2015]. The most adept architecture to process image data is the convolutional neural network (CNN), owing to the powerful inductive bias of spatial equivariance provided by convolutional layers [Tuli et al., 2021]. Alternative architectures are being explored, such as the Transformer architecture [Sharir et al., 2021], but have not become the standard yet.

In 3D, representations can take multiple, often quite different forms (see Section 3.2), each of them requiring different neural network architectures for processing and extracting information. For occupancy grids and signed distance functions, the CNN architectures for image processing tasks can be extended to 3D (3D convolutions). Since memory requirements scale cubically in

3D, 3D CNNs can only operate at limited grid resolutions and do not scale well to large scenes. Pointclouds offer a much more space-efficient representation, but lack structure. To address this, Qi *et al.*introduced PointNet [Qi et al., 2017], proposing a spatial transformer layer, projecting irregular pointclouds into canonical space before processing them with an MLP. Follow-up works further proposed architectures which allow pointclouds to be processed by convolutions [Li et al., 2018, Wu et al., 2018a]. While poinclouds offer a memory efficient space representation, they don't provide watertight surfaces. To represent surfaces explicitly, 3D meshes are the most common representation. In addition to watertight surfaces, meshes allow for non-uniform detail — complex structures are represented with many nodes and edges to capture curvatures in high detail, while flat areas can be represented using a few triangles only. Similarly to pointclouds, meshes are unstructured and unordered and require specifically designed architecture to be processed. Some approaches have been proposed (e.g. [Hanocka et al., 2019]) which use graph neural networks to process meshes and extract features using convolutional operations on the mesh. Given the different 3D representations and network architectures to process them, the quality of direct semantic annotation of 3D reconstructions, depends on a variety of variables and settings.

4.1.4 Chapter overview

The following sections will start by introducing the experimental setup and define the task we are trying to solve. Then, the SLAM system and its scene representation which are used for the comparison setting are introduced. Section 4.4 describes our fusion algorithm for adding 2D semantic labels into a height map, one of the main components of the view-based method. In Section 4.5 a few preliminary experiments are presented for building this view-based method. Section 4.6.1 describes the synthetic dataset generation process for training the semantic segmentation network, followed by a presentation of the semantic labelling network architecture. In the final section the evaluation and comparison of both view-based and map-based approaches are presented. The chapter is concluded with a summary and an outlook on future work.

4.2 Experimental setup

Operating in a full 3D setting, we would have to choose one representation and its corresponding network architecture for the map-based approach, or, compare against all representations. The first option would bias the comparison towards one representation, whereas the second option would mean a very large amount of evaluations. Furthermore, the different nature of processing by the representation-specific neural network architectures, could make a direct comparison difficult.

Instead, we choose a height map representation (see Section 3.2) of the 3D world. Fused height maps can be labelled using a standard 2D CNN, enabling the use of a network with the same architecture for both view-based and map-based labelling. This ensures fairness in comparison and allows the results to be transferable in their interpretation beyond the choices of a specific system. Our goal is to compare both approaches in a principled manner, removing any influence and bias incurred by a 3D representation-specific architecture choice for the map-based approach. In particular, we want to provide insights into the methodical advantages and disadvantages of both approaches in the context of a real-time SLAM system and give guidance as to which settings and circumstances would be more suitable to one or the other.

Using a height map representation of 3D space constrains our application space to scenes with content that can be represented with a height field. The experimental domain is therefore set to table-top scenes with objects free of or with limited curvature along the edges. Such objects include flat rectangular objects such as books and box-shaped objects, keyboards and pencils. This setting doesn't only provide a principled environment for the comparison, but also addresses an interesting problem which has not been looked at for semantic segmentation previously: requiring reconstruction at extremely high resolution for small objects such as pens, coins and paperclips, with potentially high levels of occlusion and overlap. An example is given in Figure 4.1. Furthermore, being able to reconstruct and annotate small as well as large objects is a crucial ingredient for home-robotic applications which will need to manipulate household objects that range from chairs, over plates to cables and paperclips.

To this end we design a frame-wise labelling system with incremental label fusion (a view-based method) and a one-off reconstruction labelling system (a map-based method) to augment a height map reconstruction with semantic labels. The frameworks and the comparison are presented in the remainder of this chapter.

4.3 Height map fusion

Given the task, the comparison setting requires a real-time SLAM system with a height map mapping backend. Given the limited representational power of height fields for general real world settings, most real-time SLAM systems reconstruct 3D scenes using voxelgrids [Newcombe et al.,

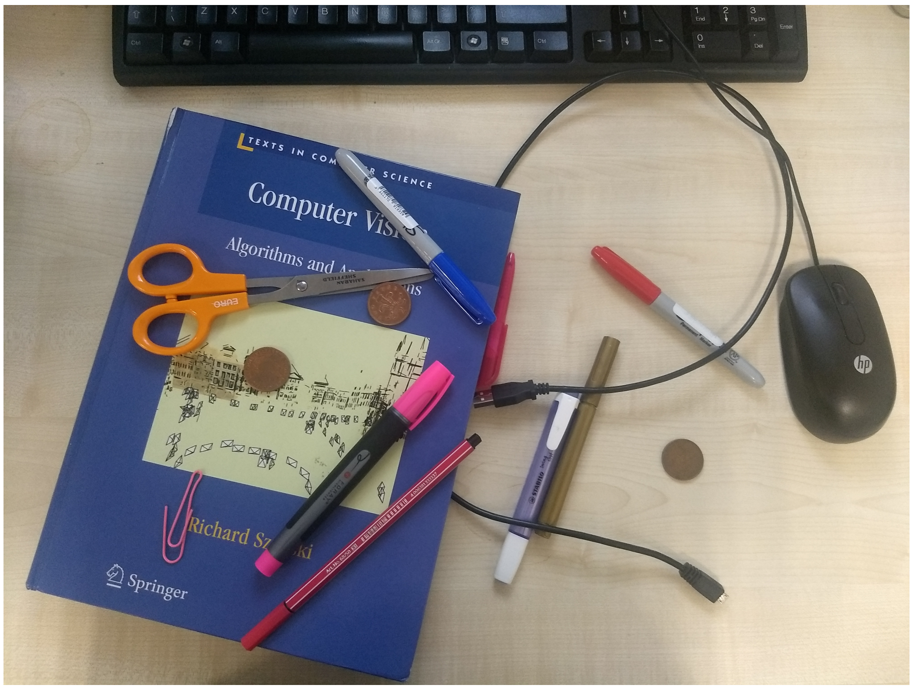


Figure 4.1: Example of a cluttered tabletop setting which represents the experimental domain of comparing view-based and map-based semantic labelling

2011a], or directly reconstruct meshes from projections [Amenta et al., 2001]. However, in some settings a 2.5D height field representation can have advantages: aiming for efficiency and extreme detail at very small scales, Zienkiewicz *et al.*propose a real-time, multi-scale height map fusion system [Zienkiewicz et al., 2016], as one of the only approaches using a height field mapping component in a real-time SLAM system. Despite the 2.5D nature of the representation, their system can reconstruct a variety of objects (see Figure 4.2) and achieve a level of detail other SLAM systems are not capable of. The height map is modelled using a triangular mesh whose vertices have horizontal coordinates on a regular grid and associated variable heights which are estimated from incremental fusion. This system uses ORB-SLAM [Mur-Artal et al., 2015b] as a camera tracker, and geometry measurements can come from either a depth camera or incremental motion stereo in the pure monocular case. The mapping component produces a fused height field as well as a fused colour map.

4.3.1 Scene reconstruction in 2.5 D

To generate a surface representation from multiple views, Zienkiewicz *et al.* [Zienkiewicz et al., 2016] start from a fixed topology mesh (see Figure 4.3), which is fit to data during reconstruction. Each vertex in the mesh only has one degree of freedom (along the z axis), which results in a height map reconstruction. To add increasing detail to the reconstruction the system uses 6 levels

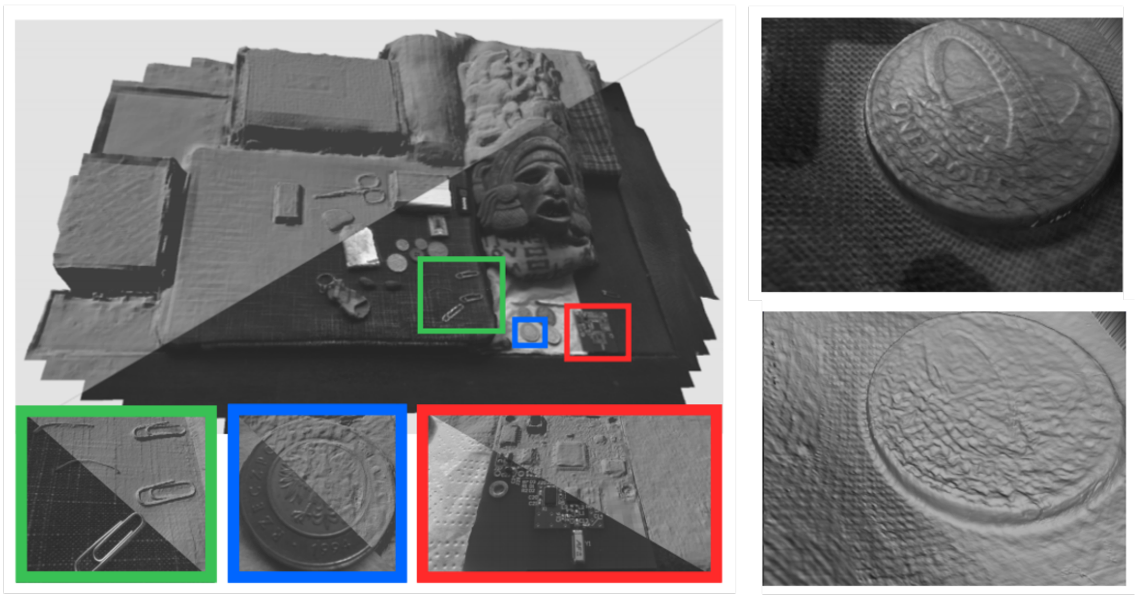


Figure 4.2: Real-time, monocular, multi-scale, height map fusion [Zienkiewicz et al., 2016]. Very small objects can be reconstructed in high detail using a 2.5D height map.

of detail within the triangular mesh, whereby for each level, triangles are subdivided using a regular tesselation factor as depicted in the bottom left of Figure 4.3. Each level of detail l chosen at each measurement depends on the depth or the on-screen area δ_B and a pre-set constant a for the tesselation factor: $l = round(log_2(\delta_B/a)$. When a depth measurement is back-projected onto the height field, it falls onto one of the meshes' triangles. To associate them with neighbouring vertices h_1, h_2 and h_3, barycentric coordinates are used:

$$z_i = \alpha_i h_1 + \beta_i h_2 + \gamma_i h_3, \tag{4.1}$$

whereby z_i is the projected height in the global frame of reference. For a set of k height measurements, all equations formed by the N measurements $i...N$ can be summarized into a system of normal equations, given by

$$J^T J h = J^T z, \tag{4.2}$$

where each row of the matrix J is given by equation 4.1 and represents the barycentric interpolation for one measurement. Solving equation 4.2 corresponds to solving the optimization problem for the surface reconstruction and this multiple equation system is solved iteratively with the Gauss-Seidel algorithm. The optimization is implemented in parallel for every triangle on the GPU.

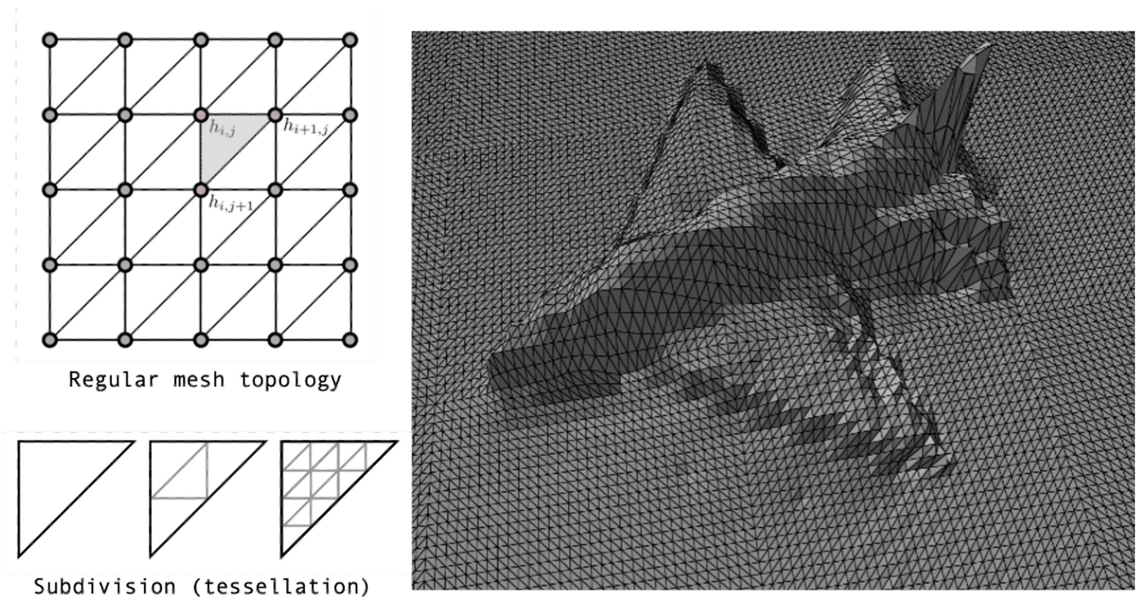


Figure 4.3: **Left**: Regular mesh structure used as a fixed topology for surface reconstruction [Zienkiewicz et al., 2016]. **Right**: The deformed topology after fitting the height map to images of a 3D scene.

4.4 Semantic height map fusion

To extend the existing real-time system to integrate semantics, the scene representation has to be extended with the ability to hold and update semantic information from multiple viewpoints. Currently, the height map consists of vertices that know about their own height, which is updated during the barycentric fusion and optimisation method of [Zienkiewicz et al., 2016]. To add semantic information to every height measurement, it has to be integrated into the barycentric fusion update. Given the semantic prediction output of a standard segmentation neural network is a probability vector over all possible classes, fusing multiple predictions is not trivial and inspiration of how to formulate this update was taken from SemanticFusion [McCormac et al., 2017b]. SemanticFusion integrates semantics into ElasticFusion [Whelan et al., 2015], leveraging their dense 3D surfel representation which they augment with semantic annotations. Each keyframe selected by ElasticFusion is run through a previously trained CNN and the resulting semantic segmentation is fused into the 3D map using the image correspondences of SLAM and a Bayesian update scheme. Based on the VGG-16 architecture [Simonyan and Zisserman, 2015], the CNN was extended into a FCN architecture for a pixelwise semantic class prediction and adapted to accept depth input. The recursive Bayesian update is formulated using the assumptions in [Hermans et al., 2014], resulting in a simple multiplicative update of each surfel probability using the current estimate of a surfel's semantic class $P(l_i|I_{l,\dots,k-1})$ and the new prediction obtained from the CNN semantic labelling output $P(O_{u(s,k)} = l_i | I_k)$:

$$p(l_i|I_{l,\dots,k}) = \frac{1}{Z} * P(l_i|I_{l,\dots,k-1}) * P(O_{u(s,k)} = l_i|I_k), \tag{4.3}$$

where $l_i \in L$ is a specific class label of L classes, I_k is the data obtained from image k and $O_{u(s,k)}$ is the network prediction for a pixel coordinate u associated with image k and surfel s.

4.4.1 Incremental label fusion

To add a semantic label fusion capability to the height map reconstruction back end, we associate a discrete distribution of semantic classes with every vertex of the mesh, and refine this distribution iteratively by projecting view-based semantic predictions onto the mesh in a per-surface-element-independent manner as in [McCormac et al., 2017b]. For every vertex, only the pixels projected onto the adjacent faces contribute to the Bayesian update. We seek to compute the posterior distribution $P(v|M^t)$ over semantic classes v for a certain vertex, given projected measurements on adjacent faces $M^t = \{m^1, m^2, \dots, m^t\}$ for all timesteps $1, \dots, t$. We define the measurement m_u^t as the network's prediction for a single pixel u given the image. We apply Bayes' rule to $P(v|M^t)$ as follows:

$$P(v|M^t) \propto P(m^t|M^{t-1}, v) P(v|M^{t-1}) \ . \tag{4.4}$$

We assume *conditional independence* of the measurements given the vertex class, i.e. $P(m^t|M^{t-1}, v) = P(m^t|v)$, and can thus rewrite the above as:

$$P(v|M^t) \propto P(m^t|v) P(v|M^{t-1}) \ , \tag{4.5}$$

describing the relation between posterior $P(v|M^t)$, measurement likelihood $P(m^t|v)$, and a-priori distribution $P(v|M^{t-1})$. Note that we dropped the normalisation constant during the derivation and thus must normalise the posterior after evaluation.

For computational reasons, we also assume *spatial independence* of the measurements and thus factorise the measurement likelihood into:

$$P(m^t|v) = \prod_{\mathbf{u} \in \mathscr{U}} P(m_u^t|v) \ , \tag{4.6}$$

where $\mathscr{U}$ denotes the set of pixels whose rays intersect with the surfaces adjacent to the given vertex and $P(m_u^t|v)$ is the measurement likelihood at pixel u and time t given the vertex class v. Since the projected measurement locations do not coincide with the location of the vertex, we model the measurement likelihood using a distance based decay $g(\cdot)$:

$$P(m_u^t|v) = g(m_u^t, v, d) \tag{4.7}$$

$$= \begin{cases} \exp(-\alpha d)a + b & \text{if } m_u^t = v \\ \frac{1-\exp(-\alpha d)a-b}{C-1} & \text{if } m_u^t \neq v \end{cases}, \tag{4.8}$$

where d is the Euclidean distance between the vertex and the projected pixel, α is a tuning parameter defining the decay rate and a and b are scaling factors based on the total number of semantic classes C which ensure that $P(m_u^t|v)$ models a uniform distribution as $d \to \infty$:

$$a = \frac{C-1}{C}, \quad b = \frac{1}{C} . \tag{4.9}$$

Note that we have dropped the class index c for readability. Intuitively, the closer the projected pixel is to the vertex, the more likely the pixel class is to coincide with the vertex class. The likelihood of a measurement being of any class other than the observed one is distributed uniformly.

Finally, the output of our network is not directly a measured class, but rather a distribution $m_u^t(c)$ over possible classes c. This can be dealt with according to Bayes by evaluating a weighted average over classes, which acts as an expectation over the projected per-pixel predictions:

$$\bar{g}(m_u^t, v, d) = \sum_c g(c, v, d) m_u^t(c) . \tag{4.10}$$

where $\bar{g}(\cdot)$ replaces the measurement function for evaluating $P(m_u^t|v)$ in Equation (4.6)

Note that in order to weight the measurements, their distribution is divided by the barycentric coefficient α_i, modeling the idea that the closer the point falls to the vertex, the stronger the distribution should contribute. As the barycentric coefficients sum to 1, it is guaranteed that the information of the projected distribution is not duplicated (see Figure 4.4).

The semantic labelling accuracy can be affected by the viewing angle of the camera with respect to the observed scene. A point observed from a flat viewing angle is more difficult to label (e.g., likely to be more affected by lighting conditions and occlusions) than a point observed from a frontal, orthogonal view. Although we do not take the viewing angle into account in our algorithm for label fusion, we see it as an interesting addition to the method for future work.

Semantic probability interpolation in image space (2D) As an alternative approach, semantic predictions could also be interpolated in image space. In this approach, the interpolation would take place in the 2D image plane and the semantic label distribution of a projected vertex of the 3D height map would be updated with the probability distributions of the adjacent pixels (see Figure

4.5). The weightings of the individual per-pixel contributions would be obtained from simple bilinear interpolation and the resulting interpolated distribution would be used with the class prior to obtain the likelihood for Equation 4.5 similarly to the method in height map space.

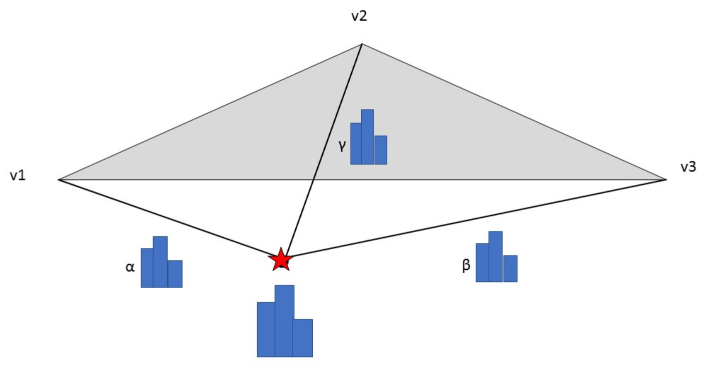


Figure 4.4: Diagram illustrating how the distribution of a projected measurement will overall only be added once to the triangular mesh.

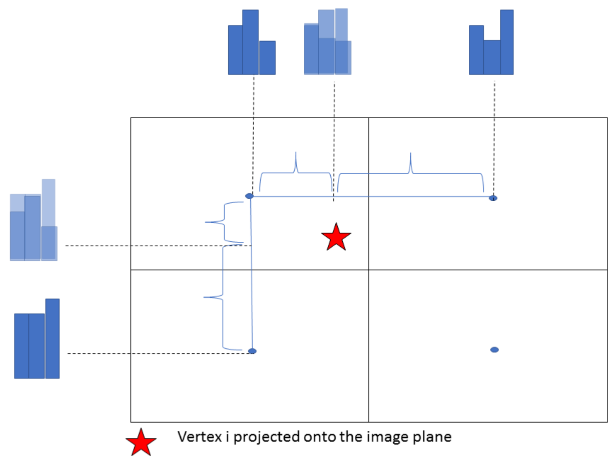


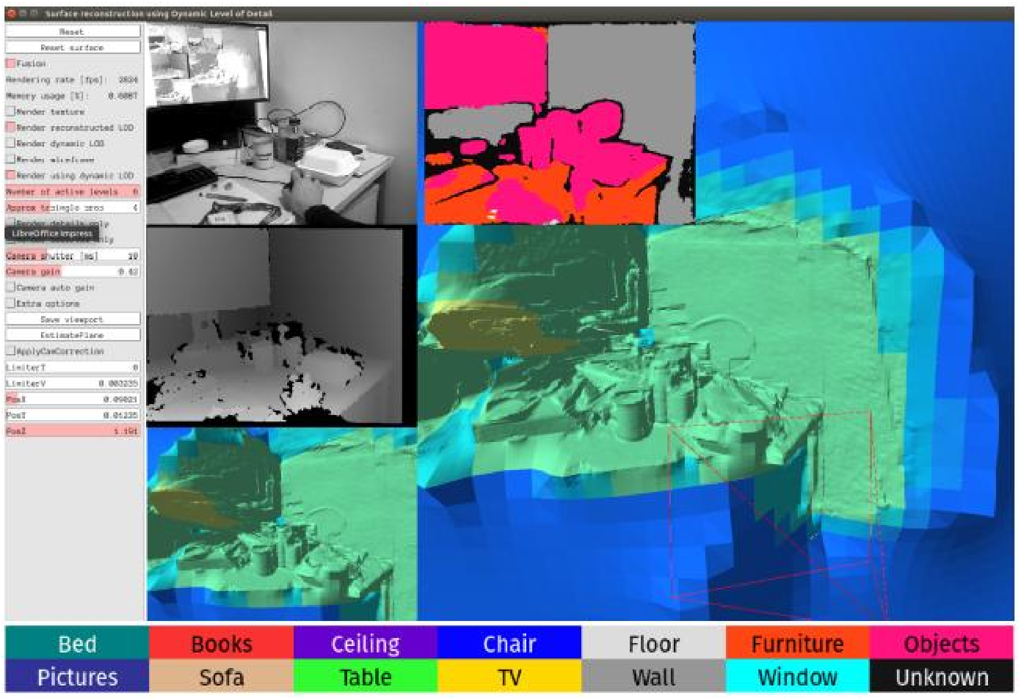
Figure 4.5: Diagram illustrating how the semantic label distribution of one vertex of the height map would be updated with a new measurement, in image space. The new semantic predictions of neighbouring pixel values are weighted according to bilinear interpolation and used to update the posterior semantic distribution of the projected vertex.

In principle, interpolation in image space is conceptually easier and probabilistically more sound, as it uses bilinear interpolation instead of a weighted addition of distributions using barycentric coefficients. Furthermore the interpolation would not be affected by the scale of the height map, whereas the interpolation in height map space could be unequally scaled given the non-uniform resolution of the triangular mesh around a vertex. However, to be consistent with the system's implementation of depth fusion and given that the first prototype implementation yielded good performance, we decided to interpolate in height map space.

4.5 Preliminary experiments

4.5.1 Integrating an off-the-shelf neural network with height map fusion

For an initial prototype, the implementation of U-Net [Ronneberger et al., 2015] used in SemanticFusion [McCormac et al., 2017b] was adapted and integrated into the height map fusion framework. The pipeline was tested by feeding the network the RGB images captured by the moving camera used for height map reconstruction and displaying the semantic segmentation of this frame in parallel to the system visualisation. The off-the-shelf semantic segmentation network was trained on a dataset of larger indoor scenes, where all smaller sized objects are represented by one class. Although the network produces reasonable segmentation for cluttered table-top scenes (see top center of Figure 4.6), for our table-top scenario, we would like to be able to differentiate between different object classes (e.g. book, pen, scissors).


Surface reconstruction using Dynamic Level of Detail

```
Reset
Reset surface
Fusion
Rendering rate [fps]:   2824
Memory usage [%]:     0.6087
Render texture
Render reconstructed LOD
Render dynamic LOD
Render wireframe
Render using dynamic LOD
Number of active levels     6
Approx triangle area        4
Camera shutter [ms]        10
Camera gain              0.42
Camera auto gain
Extra options
Save viewport
EstimatePlane
ApplyCamCorrection
LimiterT                    0
LimiterV             0.002235
PosX                  0.09821
PosY                  0.01235
PosZ                    1.191
```

Figure 4.6: Example of the semantic height map pipeline prototype running in real-time (screenshot).

GPU bottleneck for real-time monocular semantic fusion One hardware limitation encountered during this experiment was caused by GPU memory: both the depth reconstruction part of height map fusion and the network forward pass require a significant amount of memory. Thus, the semantic prediction cannot be run real-time if height map fusion is using a monocular camera for reconstruction with its expensive depth estimation algorithm. We therefore use an RGB-D camera (The Microsoft Kinect camera).

4.5.2 Implementing and testing semantic height map fusion

As a next step, we implemented and tested our semantically augmented height map fusion system using the same off-the-shelf semantic segmentation network. Semantic interpolation in height map space was implemented according to the formulation in section 4.4, running in parallel to height fusion, on the GPU (see Figure 4.7). An example of the results of fused semantics into a height map reconstruction can be seen in Figure 4.8. This initial prototype verifies the functionality of the implementation, but given the poor prediction quality by the network, it is difficult to properly evaluate the performance of the fusion algorithm. Therefore, as a next step the off-the-shelf network was replaced with a network trained on a more appropriate dataset.

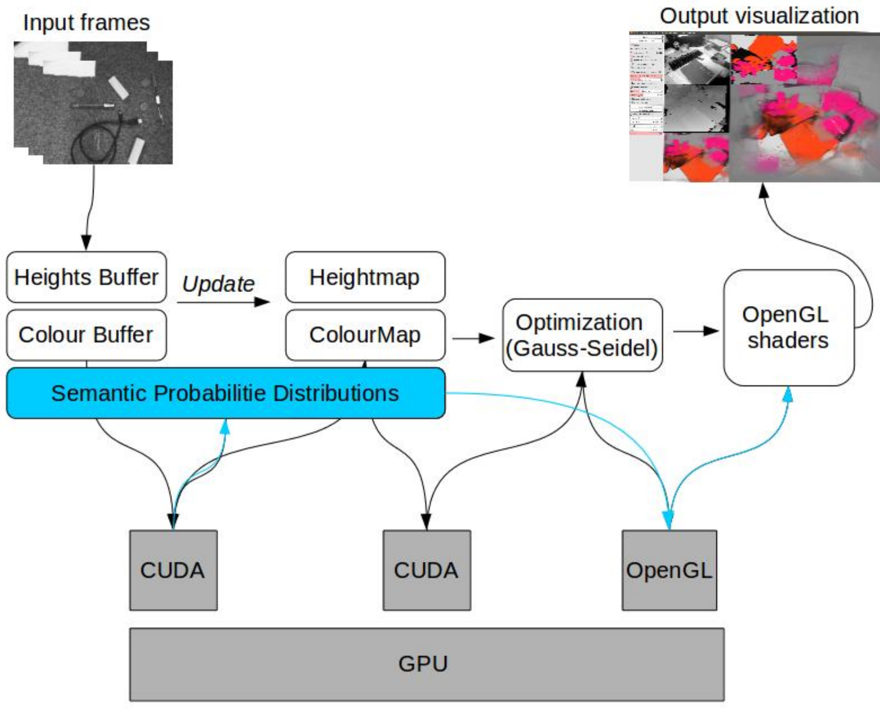
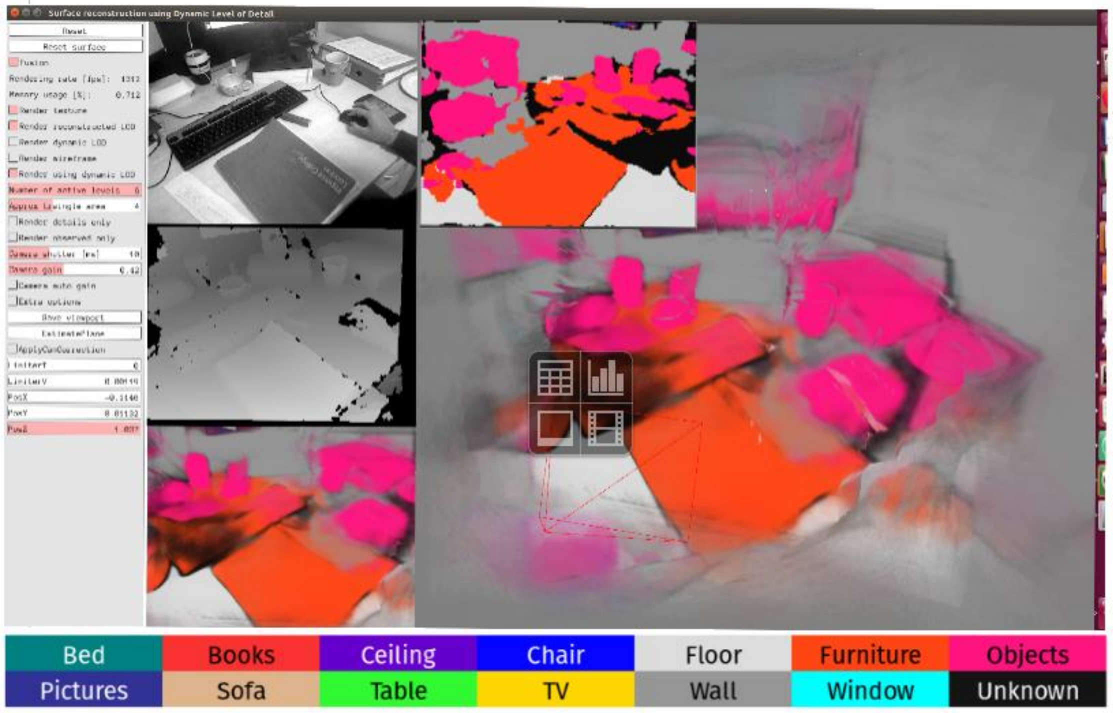


Figure 4.7: Augmenting height map fusion with semantic bayesian fusion. Semantic class probabilities are saved and processed in parallel to height and colour.

4.5.3 Creating a real-world dataset of small objects

In a first instance, we decided to use an off-the shelf network and fine-tune it on a real-world dataset of scenes with small objects. To avoid the impossible task to label over a thousand images in order to train a network for semantic segmentation on small objects, the reconstructed height map was leveraged to generate training data. First, an extension to height map fusion was implemented to allow for saving and reloading of both the fused height map and the fused colour map.



Rendering rate [fps]: 1312
Memory usage [%]: 0.712
Number of active levels 6
Approx. triangle area 4
Camera shutter [ms] 10
Camera gain 0.42
InitArf 0
LimitArV 0.00449
PosX -0.1140
PosY 0.01132
PosZ 1.037

Figure 4.8: Example of a first implementation of the fusion algorithm running real-time within height map fusion (screenshot).

Additionally, the framework was modified to allow for two run-time modes:

- Height map reconstruction (with colour fusion) and saving.
- Loading of the annotated colour map and generation of synthetic views with semantic labels.

Note that for memory efficiency, only the intensity values of the fused colour map are saved. After generating and saving a height map and its respective intensity map, a large number of camera poses and corresponding keyframes are collected. All values are saved and the colour map is converted from binary data into an image. This image is subsequently coloured for a selected number of semantic labels using a simple drawing software. To generate a 2.5D labelled height map, the annotated image is loaded back into the Height Map Fusion system together with its respective height map. An example can be seen in Figure 4.9.

From this representation and using the saved camera poses, a large number of 2D projections can be generated, each containing the semantic labels added to the 3D reconstruction (see Figure 4.10 for two training data examples). This, together with the keyframes and saved camera poses constitutes a dataset for training a semantic segmentation network. An illustration of the pipeline can be seen in Figure 4.11. In total, 12 height maps were manually annotated from which a total of 1500 projected training images were generated.

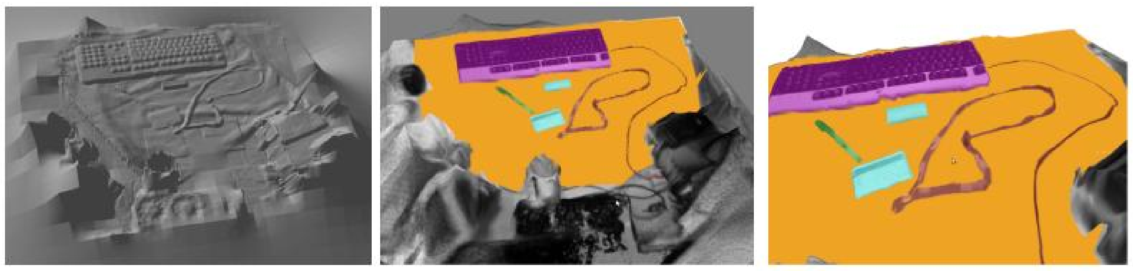


Figure 4.9: **Left**: reconstructed height map of a table top scene. **Centre**: the corresponding intensity image, with label annotations. **Right**: A screenshot of the 2.5D Semantic ground truth height map after loading the annotated height map back into height map fusion.

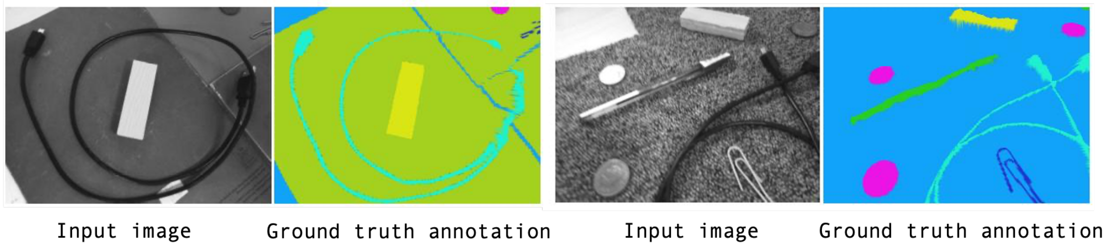
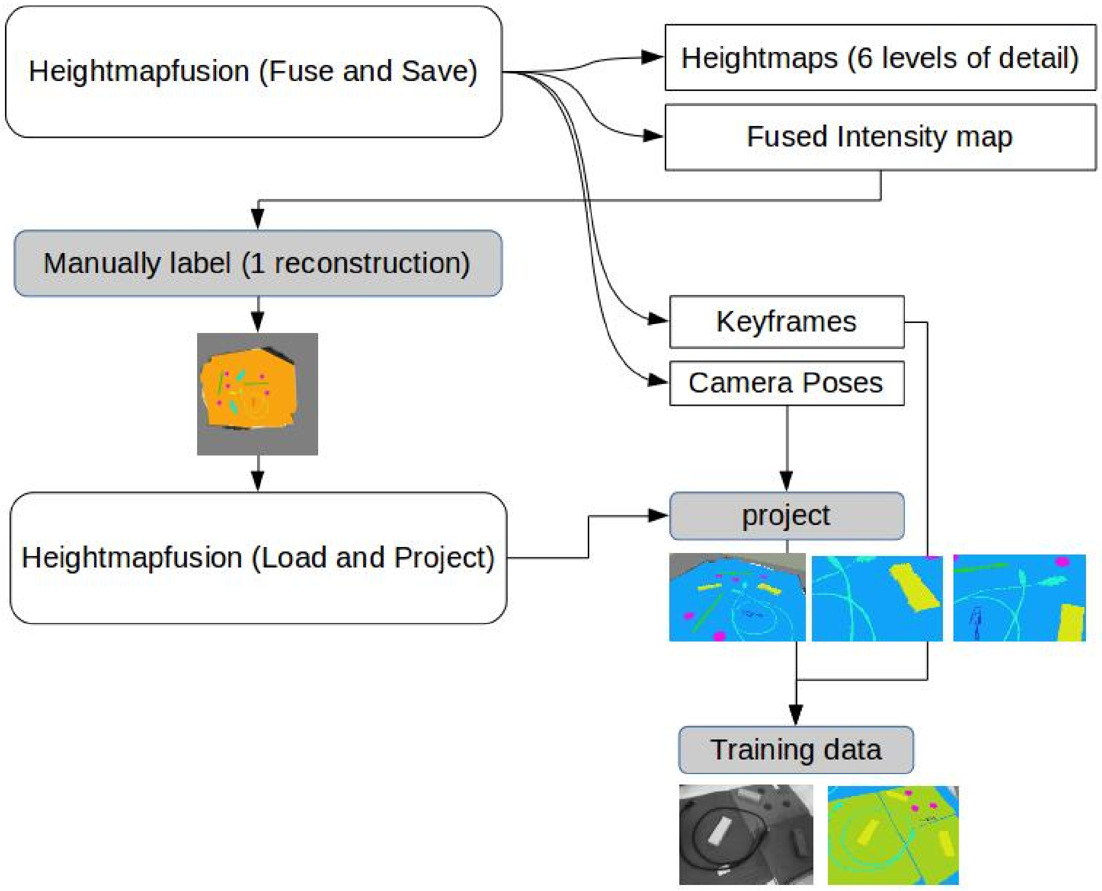


Figure 4.10: Two examples of projected intensity and semantic labels from a manually annotated height map.


Heightmapfusion (Fuse and Save)
Heightmaps (6 levels of detail)
Fused Intensity map
Manually label (1 reconstruction)
Keyframes
Camera Poses
project
Heightmapfusion (Load and Project)
Training data
Figure 4.11: The dataset generation pipeline to produce a set of labelled projections from a manually annotated height map reconstruction.

As a consequence of the colour-interpolation performed by OpenGL to generate the 2.5D representation of the coloured intensity map in 2D, object boundaries were found to contain interpolated colour gradients and could therefore not be easily assigned to one class or another when converting the colours into training labels. These areas were therefore chosen to be set to contain ignore labels excluded during training (see Figure 4.12).

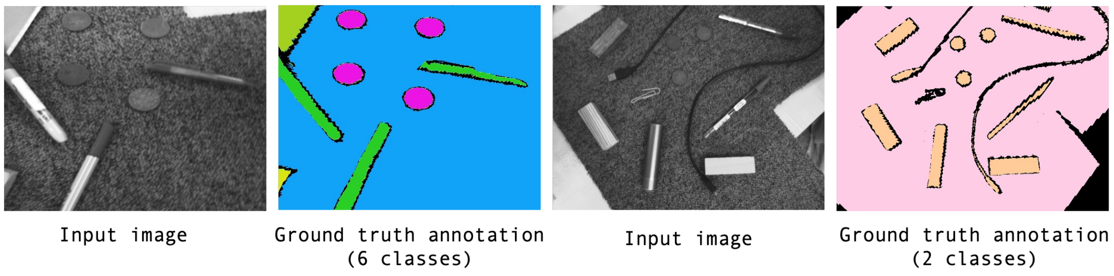

Input image
Ground truth annotation (6 classes)
Input image
Ground truth annotation (2 classes)

Figure 4.12: Examples of training data; ground truth annotation includes ignore labels in object border regions. **Left**: 6 classes. **Right**: 2 classes (foreground and background).

4.5.4 Training on the augmented real dataset

Two off-the-shelf networks were fine-tuned to the generated dataset. First, an implementation of *RefineNet* [Lin et al., 2017] (among the state of the art in image segmentation at the time of experiments) with pretrained weights on the ImageNet dataset [Russakovsky et al., 2015] was trained using the Tensorflow python API. The network was trained on two dataset versions, one with 2 classes only, foreground and background, and one with 6 classes (*background*, *coin*, *cable*, *pen*, *brick*, *paperclip*). For training, an image size of 640×480 was used and a batch size of 2. The learning rate was set to $1e^{-3}$ and we used the Adam optimiser [Kingma and Ba, 2015a]. Given the ignore labels in the training data, a binary label mask was applied to the loss function, weighting every pixel loss at by 0 if its ground truth is set to the ignore class.

As can be seen from the results on the dataset of 2 classes, the network learns to predict foreground and background (see Figure 4.13, left), but does not generalise well for unseen and different looking surfaces (Figure 4.13, right). Similarly, the network did not generalise well on the dataset of 6 classes (Figure 4.15, left). It was therefore decided to add more variety to the data in order to avoid overfitting. To this end, a number of pre-generated textures with randomized colour overlays were added to the images (see Figure 4.14). Despite the additional data augmentation, segmentation results did not improve and to rule out the large network size of RefineNet as a cause of overfitting, we trained on a smaller network, implemented after the U-Net architecture [Ronneberger et al., 2015]. Despite the reduced network size, results did not improve significantly (see Figure 4.15, right). It was concluded that despite producing a large number of labelled projections from

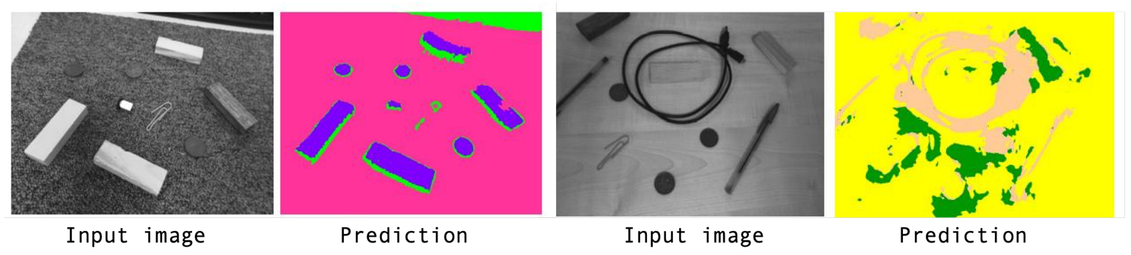


Figure 4.13: RefineNet results for 2 class prediction. **Left**: Input scene with the same table surface as used in training scenes. **Right**: Input image with a different table surface

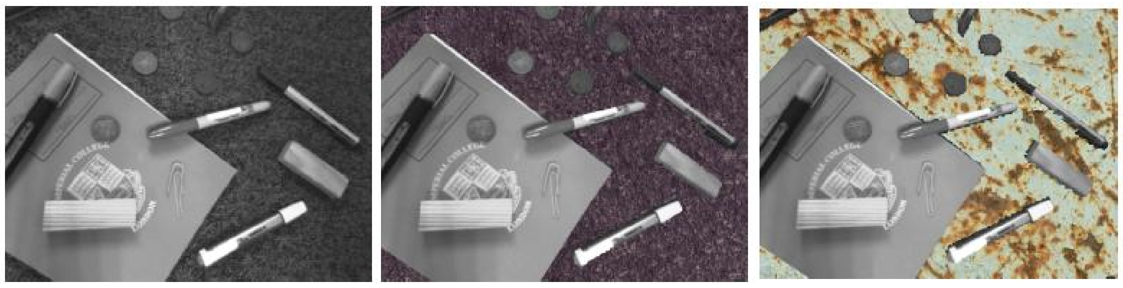


Figure 4.14: Examples of varying backgrounds with random textures and colour overays in training images for the purpose of data augmentation

one hand-annotated fused reconstruction, the dataset variety remained limited since a majority of datapoints are produced from the same set up, containing the same objects with a fixed arrangement. For better results, more variety in the scene has to be introduced, which is best achieved through synthetic data.

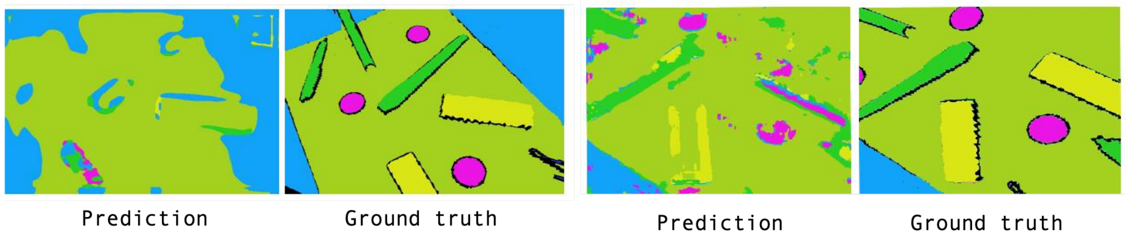


Figure 4.15: Qualitative results for RefineNet (left) and U-Net (right) on a test dataset example

4.5.5 Discussion and conclusion of the preliminary experiments

From the preceeding experiments it could be concluded that 1) a neural network for semantic segmentation can be integrated and run in parallel with the height map fusion system proposed by [Zienkiewicz et al., 2016], 2) the semantic fusion algorithm proposed in section 4.4 can integrate semantic labels into the height map representation from multiple segmented viewpoints. Our experiments on fine-tuning off-the-shelf networks on a hand-labelled real-world dataset demonstrated that despite our data augmentation methods, the generated data lacks variety and does not enable good generalisation. In conclusion, it was decided to generate fully synthetic data to train the semantic segmentation network. The dataset generation method will be introduced in the following section.

4.6 Tabletop scene data generation

To generate synthetic scenes of small objects, both 3D object models and a scene generation and rendering environment are needed.

Models Several 3D object model datasets exist, but the largest database for 3D objects at the time of experiments was ShapeNet [Chang et al., 2015] with over 3 million models and 4000 categories. Among those models, several are suitable for height map representation, including *computer keyboard, keypad*, *remote control, remote* and *airplane, aeroplane, plane*, as they typically have little overhang. For each category, 66 instances are selected, split into training, validation and test data with fractions $75\%, 10\%$ and 15%, respectively.

Scenes Synthetic scenes are generated by randomly placing instances from each object category, sampling from a uniform distribution for object ID, (x,y) position and orientation. To load models and generate scenes, the python library *Trimesh* [Dawson-Haggerty, 2019] is used. An example of a generated scene can be seen in Figure 4.18, left.

4.6.1 Dataset generation

For the comparison, synthetic data has to be generated for both the view-based and the map-based approach.

For the view-based approach, the training data are pairs of input RGB and depth with the corresponding pixel-wise semantic annotation. To render RGB images, we chose SceneNet RGB-D [McCormac et al., 2017c] for its photorealistic rendering. After a scene is generated, it is loaded into SceneNet RGB-D to render RGB, depth and semantic label maps. To generate the data, scene views are extracted at random camera locations and at a resolution of 240×320 (see Figure 4.16 for a dataset example).

For the map-based approach, training data consists of the full height map reconstruction and its semantic annotation. To generate the reconstruction, we use height map fusion with ground truth camera poses to reconstruct height maps of dimension 1025×1025 of the training scenes. To obtain semantic labels, the same scene is loaded into SceneNet RGB-D, from which semantic label

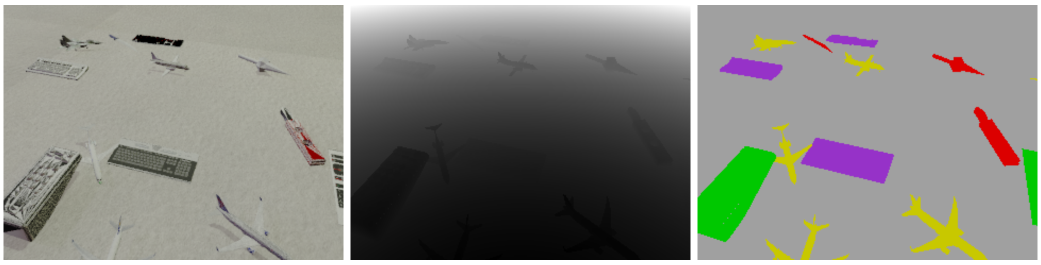


Figure 4.16: Dataset example from the view-based dataset. **Left to right**: RGB, depth and pixel-wise semantic annotation.

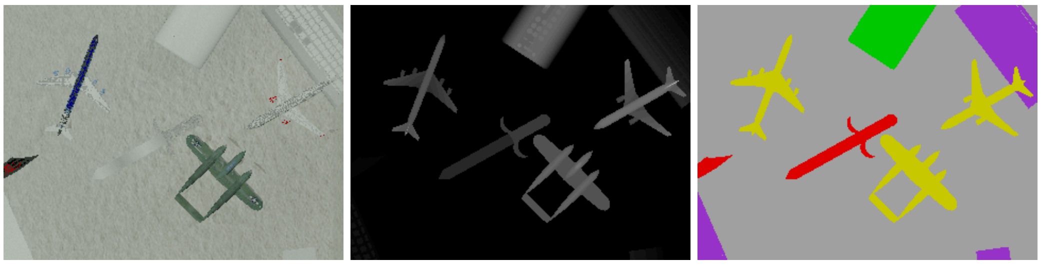


Figure 4.17: Dataset example from the map-based dataset. **Left to right**: RGB, height (inverse depth) and pixel-wise semantic annotation. The presented images are cropped sub-samples from a larger height map reconstruction.

projections can be rendered. To adapt its rendering engine to render height maps, an orthogonal camera is simulated using the OptiX engine. To evade the memory bottleneck of labelling the entire scene in one forward pass, map crops of resolution (240×320) are taken at random locations (see Figure 4.17 for a dataset example for the map-based approach). Note that while we vary the camera height between $0.18m$ and $0.4m$ and extract views with large angle variance $(0° - 40°)$, the rendered height maps exhibit canonical scale and orientation.

Overall, 647 scenes are generated, of which 500 are training, 120 validation and 27 are test scenes. From each scene, both views and height map crops are extracted to train the view-based and map-based networks respectively. An illustration of the synthetic dataset pipeline is shown in Figure 4.18.

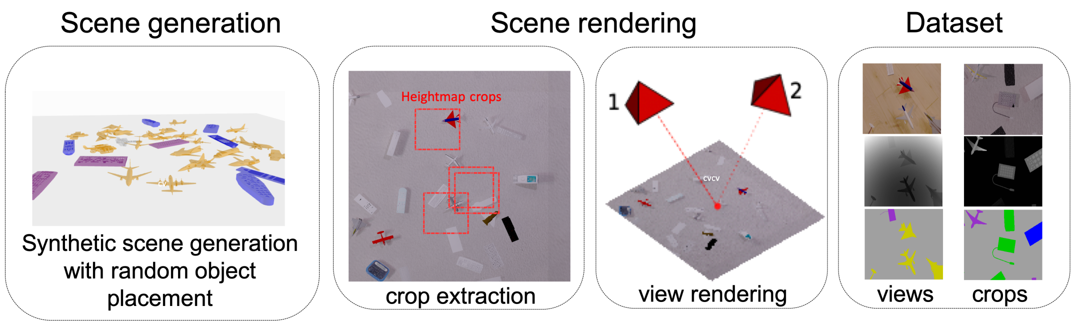


Figure 4.18: Dataset generation. A random synthetic scene (**left**) from which a height map, height map-crops and views are extracted (**centre**), yielding our datasets of views (RGB, depth, semantic segmentation) and height map crops (RGB, height, semantic segmentation) (**right**).

4.7 View-based and map-based labelling

With the access to a large number of synthetic table top scenes with small objects, we were able to train a semantic labelling network for the comparison. To ensure fairness, the same network architecture (introduced in section 4.7.1) is used for both approaches. This is possible, since the 3D reconstruction is a height map – the height values can be fed as a *depth* channel to an RGB-D network. While the network for the view-based method is trained on random RGB-D views, the labelling network for the map-based method is trained on crops taken from the height maps reconstructed with the height map fusion system. For the comparison, the trained networks are deployed in their respective systems (view-based and map-based) to add semantic labels to the scene reconstruction. The following sections describe each step of the methodology in more detail.

4.7.1 Network architecture and training

We use a fully convolutional network based on the *FuseNet* architecture [Hazirbas et al., 2016], which uses two parallel encoder branches (one for RGB, one for depth), to predict semantic labels for every pixel (see Figure 4.19). Similarly to the *FuseNet* implementation, the features extracted from the depth encoder are added to their RGB counterpart. Note that we do not experiment with RGB input only, since in the setting of the comparison of both methods, each network requires a height or depth component in the input. We experiment with different network architectures to obtain the best performing network with a minimum number of parameters. The best performing model (8 convolutional layers and skip layers at every downsampling step) was trained with a batch size of 8, a drop-out rate of 0.1 and a learning rate of $1e^{-3}$ with exponential decay of base 0.96 every $1e^{5}$ steps. All our models were trained using the Adam optimizer [Kingma and Ba, 2015a] and developed in Tensorflow. Using the same architecture, two networks were trained, one for the view-based method using the dataset of random views of synthetic scenes and one for the map-based method using the dataset of random crops generated from a set of height map reconstructions and semantic annotations of simulated scenes (see Section 4.6.1). Both networks were trained until convergence which we observed after 250 epochs. Note that we did not average performance over multiple random seeds in these trainings, since the final goal was not an evaluation of the generalisation on the test set, but to use the best network for each task in our comparative study of view-based and map-based semantic labelling. For both networks, the checkpoint with the best prediction performance on the validation set was selected (see Table 4.1 for quantitative

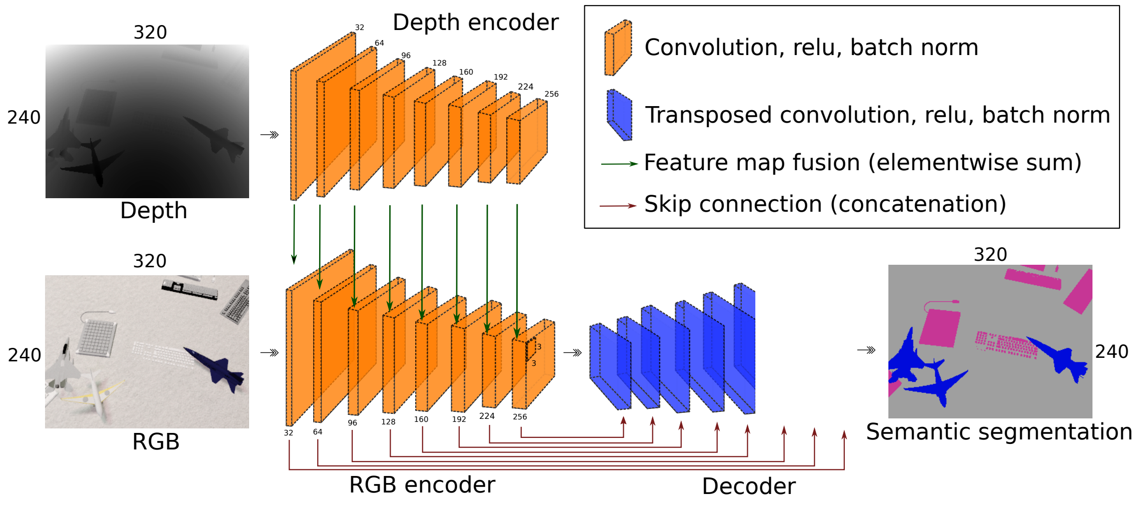

320
240
Depth
Depth encoder
32
64
96
128
160
192
224
256
Convolution, relu, batch norm
Transposed convolution, relu, batch norm
Feature map fusion (elementwise sum)
Skip connection (concatenation)
320
240
RGB
3
3
RGB encoder
Decoder
320
240
Semantic segmentation

Figure 4.19: Illustration of the fully convolutional architecture of the networks we use for view-based and map-based labelling. Depth (view-based) or height (map-based) and RGB encodings are fused at every encoding layer. Both encoders consist of 8 convolutional modules. Deconvolutional modules consist of one upsampling and two convolutional layers.

prediction results of each model on our test set).

4.7.2 View-based labelling

The view-based method is implemented based on the semantic height map fusion system presented in Section 4.4. As a labelling network, the semantic segmentation network trained on table-top scene views is used. The method takes in a stream of RGB-D images of a scene which are fed to the height map fusion system for height map reconstruction and in parallel, sent through the CNN, producing pixel-wise semantic segmentation labels for every frame. The frame-wise semantic labels are then fused into the height map following the bayesian fusion algorithm presented in 4.4.1.

4.7.3 Map-based labelling

The map-based labelling method is implemented as a one-off segmentation of the entire height map reconstruction produced by height map fusion. In principle, labelling the scene directly could be implemented via a single pass through a very large CNN, but here, we propose to sequentially crop and segment parts of the map using a sliding window approach. Our choice makes our method scalable, as the sliding window can be applied to an arbitrarily sized height map without memory restrictions. While a naive sliding window approach of tiling the map and processing each sub-height-map would result in a loss of context in the border regions of each crop, we ensure correct

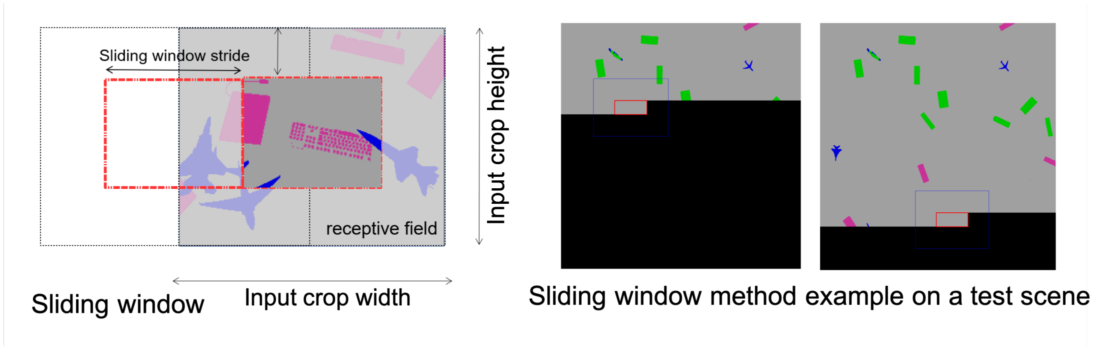


Figure 4.20: **Left**: the map-based labelling uses a sliding window approach, with the offset determined according to the receptive field of the network. **Right**: example of the sliding window method on a test scene.

segmentation by choosing the sliding window offset o based on the theoretical receptive field r of the network:

$$o = d - 2r \,, \tag{4.11}$$

where d is the dimension of the network input in the sliding direction (width or height). We use the following equations to compute the theoretical receptive field recursively through all layers:

$$\begin{aligned} s &= s_{l-1} s_l , & (4.12) \\ r_l &= r_{l-1} + (k-1)s \,, & (4.13) \end{aligned}$$

with s as the stride of the network convolutions, l as the layer index, k as the kernel size and r_l as the receptive field for features at layer index l. Note that the sliding window offset is conservatively chosen and could practically be increased by considering the network's effective receptive field [Luo et al., 2016]. With this method (see Figure 4.20), we ensure the same context for every pixel as would be obtained during a single forward pass of the entire map through our network, while avoiding GPU memory limitations.

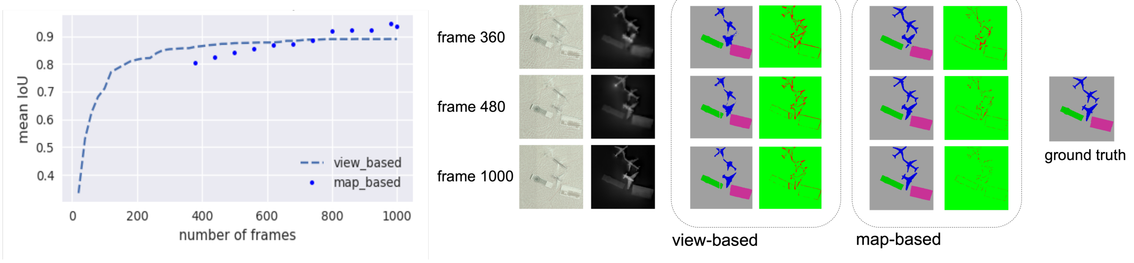


Figure 4.21: **Left:** semantic labelling quality based on map coverage (number of frames seen) for the view-based and the map-based method. **Centre:** A sub-sample of the map showing RGB and height reconstruction at different coverage. **Right:** corresponding semantic segmentation and error maps for the view-based and map-based methods.

4.8 Evaluation and comparison

Our final goal is a principled comparison of both methods in terms of labelling accuracy and efficiency. To this end, we evaluate and compare our methods on our test set of simulated scenes, which allow for a set of controlled experiments. We leave the evaluation on real data for future work. In the following we present our experiments and results.

4.8.1 Evaluation procedure

We generate test sequences from our test scenes which have camera locations relative to the scene, randomly sampled from a range which stochastically achieves full scene coverage. For each test sequence, we deploy the view-based network during the reconstruction of a synthetic scene and use our bayesian fusion update scheme to obtain a semantically labelled height map. We save the reconstruction at regular intervals where it is labelled by the map-based network. The semantic scene segmentation obtained by each approach is compared against the scene's ground truth using the mean Intersection over Union (IoU) over all classes. The alternative per-pixel accuracy measure is not very useful in our scenario due to the dominant number of pixels in the background class.

4.8.2 Training results

The best models achieved a mean IoU of 0.95 and 0.93 for the view-based and map-based tasks respectively (see Table 4.1). We suggest that this small discrepancy in accuracy, occurring despite identical architecture and training sample number, could be due to the different characteristics of the data seen by the networks, arising from the nature of their tasks and the distributions of their views. Compared to the view-based task, the map-based task is easier to learn since all map crops are taken from a canonical top-down orientation of the camera. The lower variability can possibly also lead to stronger overfitting on the training data and could explain the lower performance of our map-based method on the test data. Qualitative results for both networks can be seen in Figure 4.22.

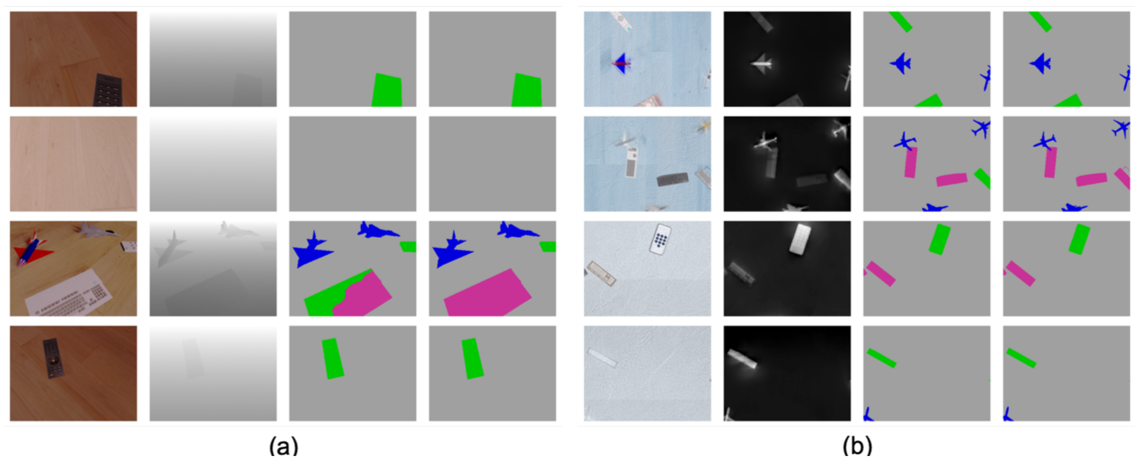


Figure 4.22: qualitative results from the view-based network (**a**) and the map-based network (**b**). *Left to right*: Input RGB, input depth (view-based) or input height (map-based), network output, and ground truth. The image dimensions are 320×240.

	mIoU	surface	remote control	keyboard	model plane
View-based	0.95	0.99	0.89	0.74	0.51
Map-based	0.93	0.99	0.68	0.91	0.78

Table 4.1: Performance of our view-based and map-based networks on the test dataset of single views and map crops, respectively. Evaluated using mean Intersection over Union (mIoU).

4.8.3 Evaluation of the view-based method

We evaluate our view-based method on our test scenes by generating a semantic height map using our semantic fusion algorithm as described in 4.8. We evaluate at every 100 frames, to track the increasing semantic accuracy of the reconstructed scene. On average, our view-based method reaches a value of 0.889 mean IoU after 1000 frames, with 95% confidence bounds of $(0.86, 0.90)$. Figure 4.23 (left) displays our results for all test scenes, together with the average performance. As illustrated by the semantic reconstruction example at different coverage levels (number of seen frames) in Figure 4.21, the rapid early improvement in label accuracy is obtained from increasing coverage of the map, though the IoU continues to improve slowly due to fusion once full coverage has been achieved at around 400 frames. This is not surprising given the high labelling performance of the network on single images. With a more poorly performing network we would expect an even higher increase in accuracy from incremental fusion. We tested the influence of the alpha parameter in Equation 4.8 on our semantic reconstruction. It did not affect the reconstruction strongly in this setting and we chose a value of 1.0 for our experiments. We further performed an analysis of computational time for our view-based method, presented in Table 4.2.

Module	Run time
Data loading	6.13 ms
Reconstruction	1.78 ms
Semantic segmentation (1 forward pass)	77.00 ms
Semantic fusion	31.05 ms

Table 4.2: Computational analysis (per frame average) of our view based method for 27 synthetic test scenes reconstructed from 1000 frames.

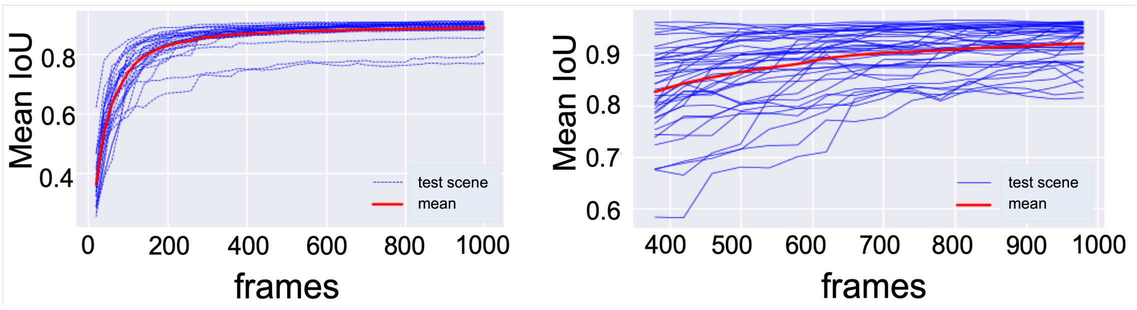


Figure 4.23: Incremental fusion results (**left**) and map-based results (**right**). For both approaches the semantic labelling accuracy is measured as the mean IoU over all classes for 27 scenes. The red line shows the average over all test scenes which reaches a maximum of $\mathbf{0.889 \pm 0.0020}$ and $\mathbf{0.922 \pm 0.0014}$ at 1000 seen frames for the view-based and the map-based, respectively. For our given sample size, the 95% confidence bounds are $(0.86, 0.90)$ and $(0.91, 0.93)$ for the view-based and map-based methods respectively.

4.8.4 Evaluation of the map-based method

The map-based method is evaluated using the same test sequences. The scene geometry is reconstructed using variable number of frames (up to 1000) and the reconstructed height maps are segmented using the sliding window method with a shift set by the theoretical receptive field of $(91, 91)$ of our network. Unlike the view-based method, we start the map-based evaluation once full coverage of the scene has been reached, to avoid segmenting incomplete image patches which lie outside of the network's learned distribution. Figure 4.23 (right) shows our results over all scenes. Our map-based method achieves an average mean IoU of 0.922 with 95% confidence bounds of $(0.91, 0.93)$. We measure the average time to segment one reconstructed map on a single GPU as $8.3s$.

4.8.5 Both approaches in comparison

To compare both methods, the mean IoU achieved on test scenes during reconstruction is evaluated, segmenting at every 20 and 40 frames for the view-based and for the map-based method respectively. This demonstrates each method's improvement with respect to the reconstruction state. Both methods are also compared with regards to processing time to evaluate their effi-

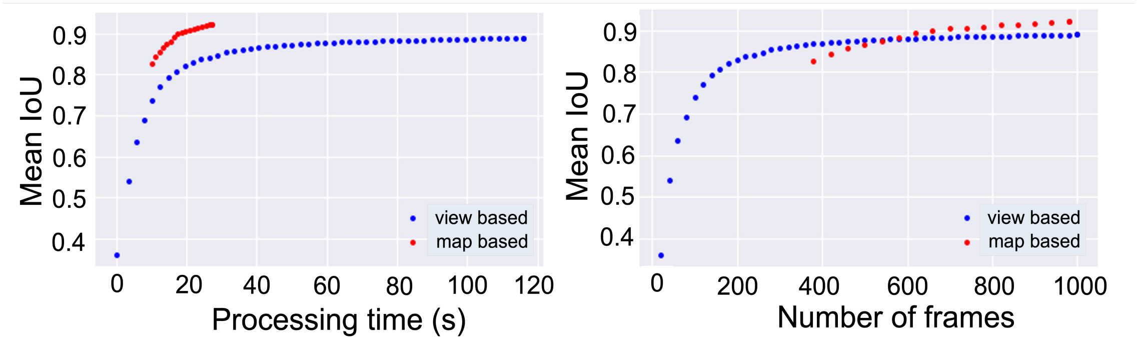


Figure 4.24: Semantic labelling accuracy achieved by the view-based method and the map-based method in comparison. **Left**: The average of the mean IoU over all classes for 27 test scenes, plotted against the number of frames used in the reconstruction. **Right**: The same results plotted against processing time. For the view-based method, we measure the time for reconstruction, the network forward pass at every keyframe and semantic fusion. For the map-based method, we combine the processing time for reconstruction without semantic fusion with the processing time required to segment the map.

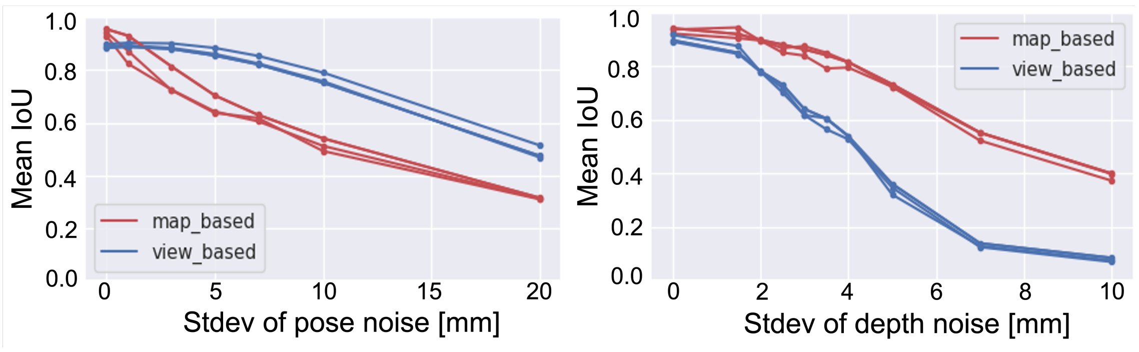


Figure 4.25: Semantic labelling accuracy over 3 randomly selected test scenes with increasing noise (**left**: pose noise, **right**: depth noise) We apply normally distributed pose and (per pixel) depth disturbances.

ciency. The results are shown in Figure 4.24. For the view-based method, the computation time of the reconstruction, frame-based semantic segmentation and semantic fusion is measured; for the map-based approach, we measure the computation time for the reconstruction and the one-off scene labelling. Note that the overall processing time of our map-based method could be reduced further, if we had only segmented the map once after reconstructing the full scene.

Our results show that on average, with overall much less computation time, the map-based method achieves a segmentation accuracy superior to the view-based method. However, after full coverage has been reached, we observe a region in which the view-based method achieves higher labelling accuracy compared to the map-based method (see Figure 4.24 on the right). This demonstrates that for the map-based method to work well, a certain level of reconstruction accuracy has to be reached. We further observe that the map based method performs better at the contours of objects than the view-based method. This is visualised in the error maps of the reconstruction example in Figure 4.21).

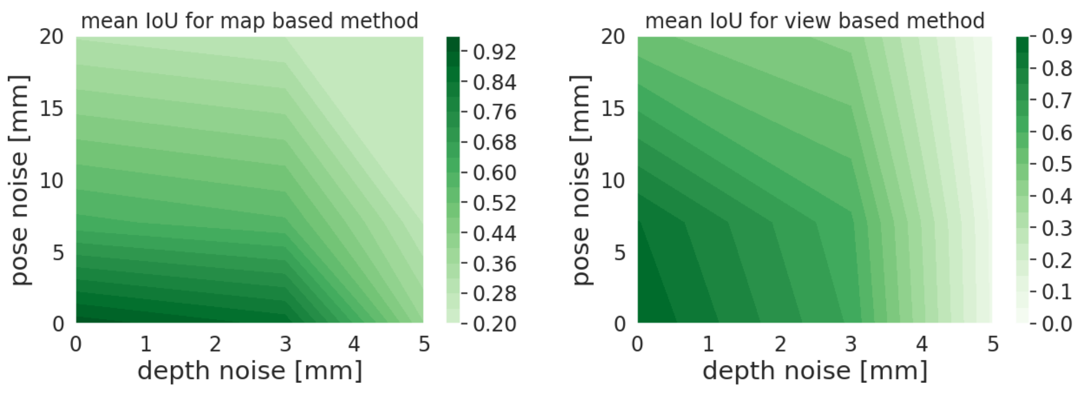


Figure 4.26: Map-based (**left**) and view-based (**right**) mean IoU as a function of pose and depth noise. Darker regions indicate higher labelling performance for each method.

4.8.6 Comparison with reduced map reconstruction quality

We experiment with degrading the map quality to evaluate the decrease in labelling quality with respect to noise, on the compared methods. In a first study, we apply normally distributed pose disturbances during reconstruction. Our results (Figure 4.25) show that the map-based method is much less robust to pose noise than the view-based method, which can be attributed to the bayesian filtering of multiple views. We then apply depth noise drawn from a normal distribution, which has a stronger negative effect on the view-based method, most likely because the latter now has to deal with reduced quality in 2D labels as well as projection errors. Plotting both methods against pose and depth noise (Figure 4.26) shows that while the view-based method is robust to pose noise, it quickly degrades to < 0.2 mean IoU with increasing depth noise. On the other hand, the map-based method is sensitive to both noise types, but to a lesser degree, degrading to only 0.2 mean IoU in the tested noise range.

4.9 Discussion

The obtained results show that for a setting which assumes perfect poses and depth, the map-based approach achieves higher labelling accuracy, and can be achieved with less computation. We argue that although the view-based network achieves a high labelling accuracy on individual frames, its deployment during the early reconstruction phase results in more errors, especially in border regions of objects, from which it cannot always recover easily. The overhead of repeated forward passes and multi-view fusion is a further disadvantage of this approach. Our experiments on pose and depth noise show that for the studied noise range, the map-based approach, although more equally affected by both types of noise, stays overall more robust than the view-based method

(Figure 4.26). We argue that its advantage results from operating on fused data. The view-based method on the other hand is less affected by pose noise, due to its in-view labelling and the benefits of bayesian label fusion. However, it strongly degrades in the presence of depth noise. We argue that this is caused by the fact that it has to operate directly on the noisy depth data, resulting in not only projection errors, but also 2D segmentation errors. Finally, since the map-based method requires a certain state of completion in reconstruction, it cannot yield ad-hoc labels in early stages of the reconstruction as would be possible with the view-based method. In summary, the advantages of one or the other method depend on the stage at which semantic annotations are required, the accuracy of camera tracking and the quality of the depth camera, and the accuracy of the labelling network itself.

Limitations and future work However, the presented results should be carefully considered within the context of our experiments. Firstly, the 2.5D geometry of the height map alleviates the map-based method from the memory limitations of 3D segmentation tasks. While using a height map allows for a principled comparison in terms of labelling accuracy and performance, real-world settings will often use a 3D representation and its respective network architecture for the map-based approach. When deciding on the labelling approach based on the specific system requirements, the additional overhead of using a full 3D labelling network will have to be taken into account. Secondly, the appearance features of our selection of scattered objects with little overhang are well visible from a top-down perspective, while for other objects (e.g. cups, bowls, chairs) with more ambiguous features it would be more difficult to train a map-based network to achieve good segmentation quality. Thirdly, in settings with less well performing networks, one-off labelling of the map segments would likely yield more errors which would in turn be better recovered with the view-based method due to multi-view bayesian fusion of labels. While we don't cover the cases of differently well performing networks in this study, we would like to leave it for future work. We see a further point of continuation in combining both view-based and map-based methods, leveraging the respective advantages in the correct setting. For instance, a real-time SLAM system would benefit from incremental fusion in the initial mapping phase and a regular map-based label refinement step in well-reconstructed regions of the map.

4.10 Conclusion

This chapter presented a study on how to best add semantic labels to a 3D reconstruction built by a real-time SLAM system, which has access to an arbitrary number of views — i.e. can achieve good scene coverage during reconstruction. In particular, two commonly used approaches based on deep learning are compared: view-based semantic labelling, whereby 2D semantic labels from keyframes are back-projected and fused into the reconstruction and map-based semantic labelling, whereby the reconstruction is directly labelled by a scene-network. The experimental setting focuses on table-top scenes which are 1) less explored than general large-scale indoor and outdoor scenes and 2) facilitate a principled comparison.

Our comparison shows that in the absence of noisy data, the map-based approach shows higher labelling accuracy and object-border details. In the presence of pose noise, only affecting the reconstruction, the view-based method shows a significant advantage, but it deteriorates more strongly in the presence of depth noise. In terms of computational cost, the map-based approach is in principle more efficient, given that every map element is processed only once, compared to the frame-wise segmentation and fusion required by the view-based method. In a three dimensional setting, this advantage will likely be reduced due to expensive 3D data processing such as volumetric labelling, but we leave this analysis for future work, as it requires the comparison between different representations of 3D data and their different methods of 3D labelling.

CHAPTER 5

Reasoning about shape and instance of scenes with limited viewpoints

Contents

5.1 Introduction

The last chapter discussed the task of semantically annotating 3D scenes and provided a comparative analysis of frame-wise incremental semantic segmentation of projected labels and direct annotation of the reconstructed scene. The experiments were conducted under the assumption that an unlimited amount of viewpoints is available, and that a scene can be reconstructed to a high degree of precision. In many settings however, those assumptions don't hold. Large scenes can be expensive to scan and in cluttered regions it may not be possible to reach all the viewpoints required to reconstruct a full model. In such scenarios it would be desirable to obtain a 3D model of a scene from only one or few views. This chapter is concerned with reconstructing cluttered 3D multi-object scenes with one or few views by reasoning about missing data. While the last chapter focused on semantic annotations, for this research problem we aim to obtain instance segmentation of the objects in the scene.

5.1.1 Reasoning in the presence of missing data and uncertainty

Humans are able to make estimates and take decisions without knowing every detail of the state of the world around them. For instance, we can cross the street without knowing the location of every car in the neighbourhood — we simply estimate that since we currently can't see any car approaching, it will be unlikely a car will reach us by the time we make it to the other side of the road. We deduce an estimate from contextual information, as well as our experience of previous street crossings. In fact, as humans we reason about missing data every time we open our eyes to scan our surroundings — we build a 3D map using stereo-vision, combining the two images created on the retinas at the back of our eyes. Apart from reconstructing a 3D scene by leveraging stereo information, we use our intelligence to reason about additional features of our surroundings, for instance, the shape and composition of 3D space currently invisible to us. We know that, depending on the context, larger objects can occlude smaller, partly occluded or completely hidden objects. Smaller objects occlude parts of larger objects. The likelihood of what shape space takes in invisible areas is constrained through our knowledge of physics — where and how objects can be physically placed and that they cannot intersect. In fact, there is a spot in the centre of our vision field that we are completely blind to, but which we don't notice since our brain fills it in using the context.

To be robust and versatile, autonomous systems will have to be able to reason and make decisions in the presence of unknowns and uncertainties in a similar fashion. Among areas such path planning and decision making, reasoning under partial information also applies to scene understanding, whereby an autonomous system needs to estimate the shape, type and the number of objects in its surroundings. Obtaining a full reconstruction of a systems' surroundings with useful annotation such as semantic and instance labels requires comprehensive and time-consuming scanning. On the other hand, autonomous systems which can operate in novel, partially observed territories and incrementally grow and leverage knowledge about their surroundings, will be more readily deployable. To achieve this, these systems will have to be able to reason about the state of the world around them, augmenting scene understanding of the directly observable with reasoning about the unobservable. Such reasoning includes estimating occupied space, object presence as well as relative distances and stability. Since autonomous systems mainly operate with cameras or laser sensors, the additional information about unobserved space has to be inferred from a laser scan or from image data.

5.1.2 Reconstructing 3D space from images

While humans are intuitively able to interpret partially observed scenes using geometric reasoning and prior experience, estimating 3D shape from RGB or depth images is challenging in computer vision due to ambiguity — many 3D shapes can explain a 2D observation. The classical approach to generate 3D shapes from depth images involves taking images from all sides of an object and fusing the re-projected points into a common 3D representation such as a truncated signed distance function (TSDF) [Newcombe et al., 2011a]. However, apart from the exhaustive nature of the task, it is often impossible to reach all required viewpoints to generate a watertight surface reconstruction [Dai et al., 2018]. This drawback has led researchers to explore learning based approaches to reconstruct 3D shapes, such as learning to complete partial reconstructions [Dai et al., 2018, Dai and Nießner, 2019, Stutz and Geiger, 2018] and predicting scenes [Song et al., 2017] or objects [Yang et al., 2017, Yang et al., 2019] from a single depth image. In parallel, researchers explore 3D shape prediction for scenes [Nicastro et al., 2019, Firman et al., 2016, Shin et al., 2019] and objects [Choy et al., 2016, Xie et al., 2019, Yao et al., 2020] from single or multi-view RGB data. While most of these approaches work with pointclouds, voxel occupancy grids or TSDF representations, some have explored alternative representations such as parametric surface elements [Groueix et al., 2018], 2D sketches [Wu et al., 2017], graph neural networks [Dai et al., 2017b], and the increasingly popular implicit neural surface representations [Park et al., 2019a, Mildenhall et al., 2020, Chabra et al., 2020].

5.1.3 3D instance segmentation and object decomposition

2D instance segmentation has progressed significantly with state-of-the-art methods such as Mask R-CNN [He et al., 2017] and DETR [Carion et al., 2020], but segmenting objects in 3D remains challenging. Hou *et al.* [Hou et al., 2019] extend the idea of region proposals to 3D, combining backprojected 2D features with 3D partial scans. Another method for segmenting 3D pointclouds into objects is center voting [Qi et al., 2019b, Qi et al., 2020, Han et al., 2020]. The approach of Xie *et al.* [Xie et al., 2020] uses Hough centre voting to segment a cluttered table top into instances. However, their final output is a 2D instance segmentation whereas we aim at 3D output. Related to instance segmentation, object decomposition focuses on breaking objects down into their primitive components. Most work on decomposing compound objects into parts has focused on rigid objects such as furniture with significant structure and symmetry. Decomposing such objects into parts can help to leverage symmetry for shape completion [Sung et al., 2015] or discover structure in unseen data [Tulsiani et al., 2017]. Most approaches decompose into geometric primitives [Deng et al., 2020a, Tulsiani et al., 2017] using a library of shapes. Recently, Paschalidou *et al.*proposed soft, genus-zero shapes, learnt by an invertible neural network which learns homeomorphic mappings between a sphere and the input shape. [Paschalidou et al., 2021a]. Others use hierarchical representations and graph neural networks [Mo et al., 2019]. Paschalidou *et al.* [Paschalidou et al., 2019] showed that using superquadrics over shape primitives improves reconstruction quality, since their continuous parametrisation allows for more shape variety. Apart from providing structural information, part-decomposition has also been shown to improve reconstruction overall [Paschalidou et al., 2020].

5.1.4 Motivation

In most real-time applications it is desirable to obtain a quick estimate of the surroundings, for navigation and planning, but also to obtain a more complete understanding of the observed scene. This requires reasoning about 3D from a single view without comprehensive scanning and multi-view fusion. While estimating 3D shape is crucial for navigation and planning, for a more versatile robotic system that can interact with its environment, a notion of objects through object detection or instance segmentation is needed. Single-view 3D shape estimation for single and multiple objects has been addressed in previous works [Yang et al., 2017, Yang et al., 2019, Nicastro et al., 2019, Firman et al., 2016, Shin et al., 2019] and single-view object detection and instance segmentation for cluttered scenes has been solved in 2D using multiple approaches [Carion et al., 2020, Xie et al., 2020]. However, solving both tasks jointly for composite scenes in 3D (such as object stacks) has not been solved at the time of this project. It is a particularly challenging task: occluded areas are not only ambiguous in shape but also in instance segmentation; multiple decompositions could be valid. The task is particularly relevant for scenes containing small objects in close proximity generating occlusions which make object detection and segmentation difficult. Indoor scenes and in particular table-top scenes are common settings of such clutter (see Figure 5.1).

To address this task, we make the following observations and hypothesis: single-view object completion is challenging and has previously been solved leveraging priors and symmetry assumptions about the object shape [Yao et al., 2020, Wu et al., 2018b]. Reconstructing and segmenting multi-object scenes using a simple supervised training scheme will likely fail especially in occluded regions, due to the complexity and ambiguity of the solution. While stacks of unordered clutter don't have inherent symmetry, physics constrains the space of valid compositions of cluttered scenes. Can this constraint be learnt as a prior and leveraged to improve reconstruction and instance segmentation in occluded regions? The subsequent experiments explore this hypothesis, show that learning such a prior from data using a generative model indeed improves reconstruction compared to baselines and provide a solution to reconstruct and provide instance segmentation for a multi-object cluttered table-top setting, which we test on real data.

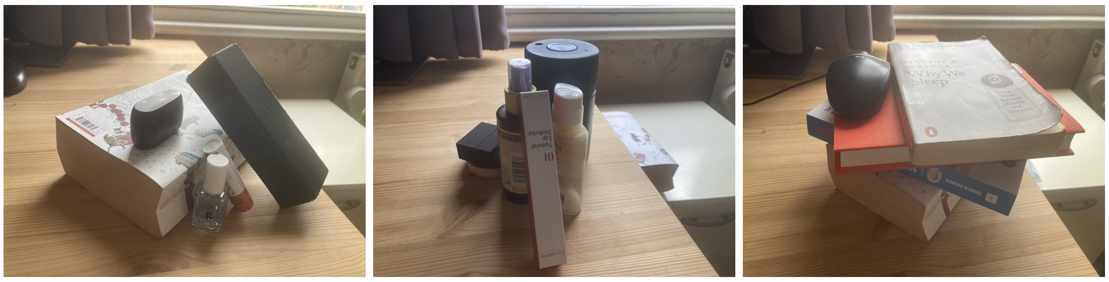


Figure 5.1: Problem setting: unordered stacks of small to medium objects in close proximity.

Task definition

Given a single depth image of objects stacked on a tabletop, our goal is to estimate the complete 3D shape of the group and segment the objects into instances. The developed method should be able to generate the full reconstruction and instance segmentation, in particular, be able to plausibly estimate shape and instances in occluded regions. To be able to work in a realistic setting, the method has to be viewpoint agnostic and work for a variable number of objects. Our goal is to work in settings where the object class is unknown and therefore the instance segmentation has to be class-agnostic.

5.2 Single-view reconstruction of multi-object cluttered scenes

5.2.1 Dataset and experimental setup

For the main experiments of this project, several design choices were made, which are described in this section. To start with, the object representation and the synthetic scene setup are described, which are used to generate the dataset. Then, the rendering method is explained, with which synthetic depth images are obtained from a given scene. Next, the architecture is introduced and the training setup is explained. Finally, the pre- and postprocessing methods are described with which the data is processed before training and after prediction.

Object representation

Although methods which work with specific CAD-model fitting are widely used and researched [Shotton et al., 2013] [Brachmann et al., 2014] [Wada et al., 2020], ultimately, relying on CAD models for object representation is a limitation, given that building an up-to-date database of all objects in the world is not feasible. As an alternative, approaches leveraging geometric primitives have received increasing attention by the deep learning community, especially in the context of shape decomposition. [Niu et al., 2018], [Tulsiani et al., 2017] and [Zou et al., 2017] decompose 3D shapes into a collection of cuboids. Improving on the the limited precision of cuboids, [Paschalidou et al., 2019] decompose shapes into Superquadrics and address the representational limitation of primitives. The trade off between reconstruction accuracy and the number of reconstructing primitives is further addressed by [Paschalidou et al., 2021b]. Other works have explored neural shape parsers [Sharma et al., 2018] and shape programs [Tian et al., 2019] to learn programmatic ways to decompose shapes into underlying primitives. Beyond single shape decomposition, primitives are also good models for data abstraction [Sommer et al., 2020] — representing scenes as a collection of primitives instead of raw pointcloud data brings advantages for space, optimisation and higher level reasoning, as well as structure understanding and shape editing [Li et al., 2019].

Superquadrics for shape approximation Superquadrics offer a general shape description, extending quadrics to multiple exponents. They define a family of shapes which includes *Superellipsoids*, *Superhyperbolloids* and *Supertoroids* (see Figure 5.2) and are defined by the implicit equation:

$$f(x,y,z) = \left|\frac{x}{a_1}\right|^r + \left|\frac{y}{a_2}\right|^s + \left|\frac{z}{a_3}\right|^t = 1. \tag{5.1}$$

The term Superquadrics is commonly used to refer to Superellipsoids in the computer vision community. To be precise, we will refer to Superellipsoids in this section when introducing equations, but will refer to them as Superquadrics in subsequent sections for simplicity and in accordance with the literature.

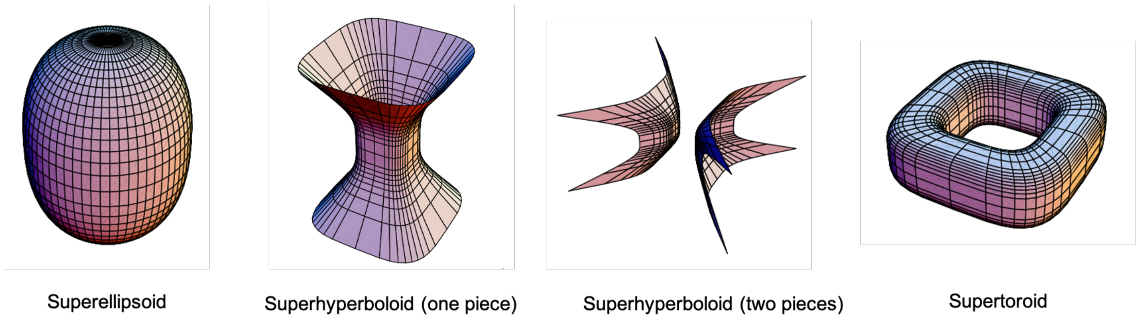


Figure 5.2: The Superquadric family

Superellipsoids can be derived from their analogous two dimensional entities: superellipses. In their explicit form, superellipses are defined by three parameters, a, b, ε, which define their extent along the x and y axes and their shape, respectively:

$$s_{a,b,\alpha,\varepsilon} = \begin{pmatrix} x(a,b,\varepsilon;\alpha) \\ y(a,b,\varepsilon;\alpha) \end{pmatrix} = \begin{pmatrix} a\cos(\alpha)^{\varepsilon} \\ b\sin(\alpha)^{\varepsilon} \end{pmatrix} \quad \begin{cases} \varepsilon > 0 \\ -\pi < \alpha < \pi \end{cases} \tag{5.2}$$

Using 5.2, points defining a superellipse can be generated by sampling values of α. Figure 5.3 shows examples of sampling points from 4 superellipses with shape parameters $\varepsilon = 0.1, 0.5, 1$ and 2. Taking the spherical product of two 2D superellipses yields a superellipsoid S_p [Jakli et al., 2000]:

$$S_p(\mu,\omega) = \begin{pmatrix} x(p,;\mu,\omega) \\ y(p;\mu,\omega) \\ z(p;\mu,\omega) \end{pmatrix} = \begin{pmatrix} a_1\cos(\mu)^{\varepsilon_1}\cos(\omega)^{\varepsilon_2}, \\ a_2\cos(\mu)^{\varepsilon_1}\sin(\omega)^{\varepsilon_2} \\ a_3\sin(\mu)^{\varepsilon_1} \end{pmatrix} \tag{5.3}$$

where parameters a_1, a_2 and a_3 scale the primitive along the 3 coordinate axes and e_1 and e_2 determine the shape of the superellipsoid cross-section perpendicular and parallel to the x, y plane, respectively. The implicit equation can be derived from Equation 5.3, using the trigonometric equality $\cos^2(\alpha) + \sin^2(\alpha) = 1$:

$$((\frac{x}{a_1}^{\frac{2}{\varepsilon_2}}) + (\frac{y}{a_2}^{\frac{2}{\varepsilon_2}}))^{\frac{\varepsilon_2}{\varepsilon_1}} + (\frac{z}{a_3})^{\frac{2}{\varepsilon_1}} = 1. \tag{5.4}$$

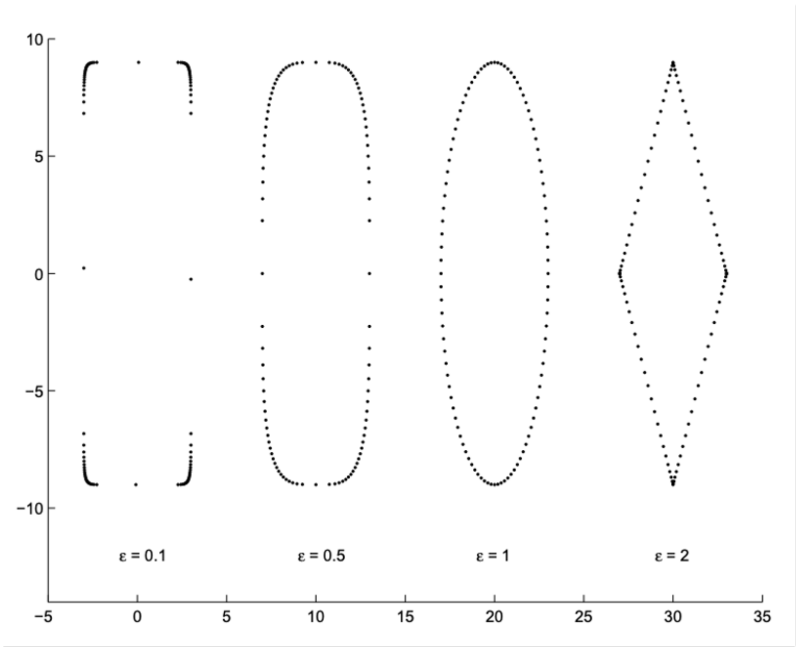


Figure 5.3: Superellipses [Katsoulas, 2004]

All points which correspond to Equation 5.4 lie by definition on the superquadric surface. Equation 5.4 is also called the *inside-outside function* since it determines whether a point (x,y,z) lies inside or outside of the surface. A correspondence between the general superquadric equation 5.1 and the implicit function for ellipsoids 5.4 can be found by setting:

$$r = s = \frac{2\varepsilon_1 + \varepsilon_2^2}{\varepsilon_1 \varepsilon_2} \qquad k = \frac{2}{\varepsilon_1} \tag{5.5}$$

For exponents r, s, $t >= 1$ equation 5.1 generates a continuous range of convex superellipsoids (see Figure 5.4 for some examples). Note that in addition to the shapes generated by keeping two exponents at the same value, we add one additional shape with $k \neq s \neq r$, which is technically not part of the superellipsoid family as it could not be represented by equation 5.4. Varying the scale parameters a_1, a_2, a_3 generates very flat, small, or bulky shapes, ideal for approximating many everyday household objects.

A dataset of superquadric object stacks

To train a learning-based method which can reconstruct multi-object scenes, a large dataset is required. Several datasets such as ShapeNet [Chang et al., 2015] or ModelNet [Wu et al., 2015] offer a large variety of different shapes widely used to train 3D reconstruction models. However, learning a general shape representation for multi-object scenes from categories with very specific

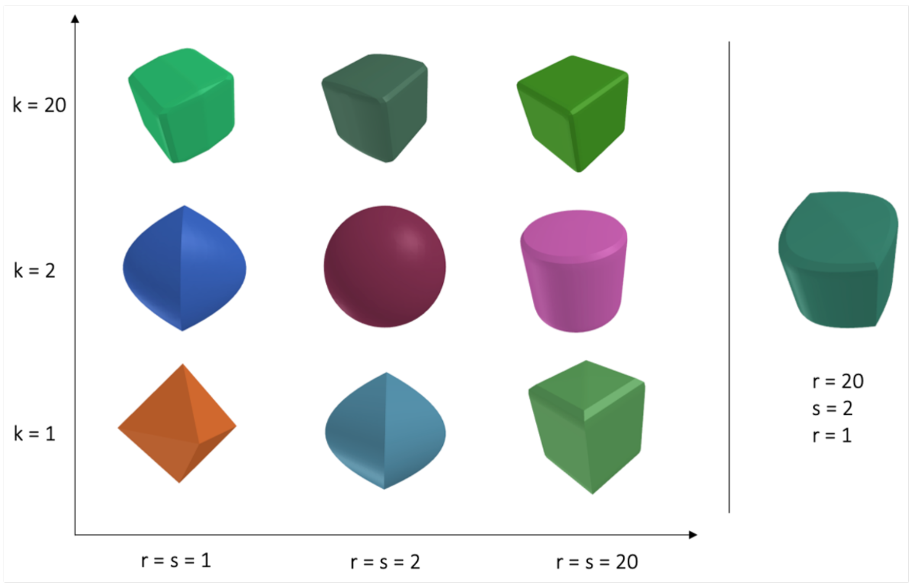


Figure 5.4: Examples from the superquadric shape library generated for the synthetic dataset.

shapes (chairs, planes, lamps) is very challenging, and as an initial step towards solving the task of multi-object reconstruction with instance segmentation, we decide to limit the scope to convex objects. To this end, superquadrics offer an attractive alternative for learning a more general shape representation for convex objects. Furthermore they offer an infinite variety of shapes given their continuous parametrisation, the latter also rendering them attractive in optimisation settings. To build a large dataset of composite scenes made of superquadrics, a large library of shapes has to be generated and combined into realistic stacks as will be described below.

Superquadric meshes To create superquadric meshes, equation 5.1 can be used to generate a voxelised signed distance function representation. The SDF can then be processed by Marching Cubes [Lorensen and Cline, 1987] to yield a mesh. Leveraging the continuous parametrisation of superquadrics, a large variety of shapes can be sampled. We generate 3500 individual shapes with exponents ranging between 2 to 100 and scales between 5 cm and 30 cm that yield our superquadric repertoire.

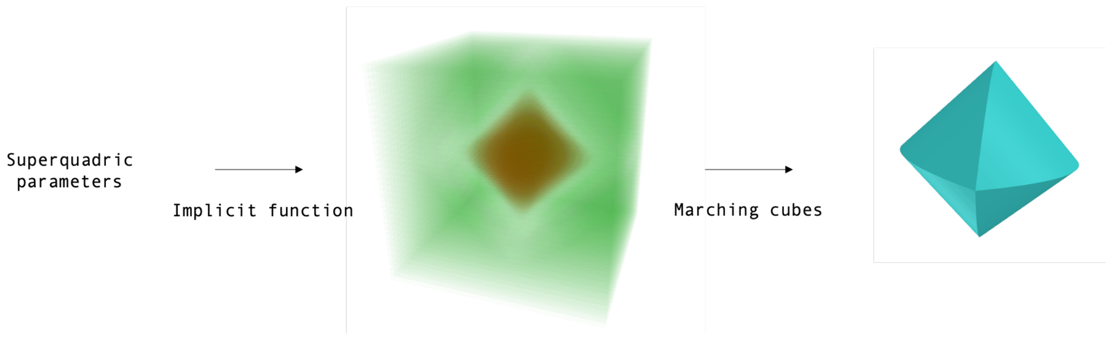


Figure 5.5: Generating a superquadric mesh

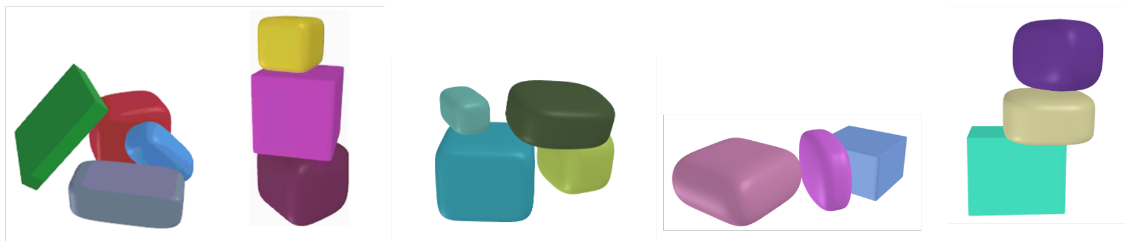


Figure 5.6: Examples from the superquadric (SQ) stacks dataset. All scenes are generated by random placement under physics simulation with *Pybullet*.

Superquadric stacks To generate realistic object stacks, a physics engine is required, which can simulate the dynamics between multiple objects. PyBullet [Coumans and Bai, 2019] offers a simple user interface to load object meshes into a simulation environment. We generate $10,000$ realistic object stacks by placing between 3 and 4 superquadrics randomly in each scene (see Figure 5.6 for dataset examples). For each object, the algorithm randomly samples a centre position within a specified scene volume as well as a random rotation, places the object into the scene and runs the physics simulator. If the chosen location and rotation does not result in any intersections with other objects or the floor and the object falls within the defined range after undergoing physics simulation, the object is kept inside the scene. The method is described in detail in algorithm 1.

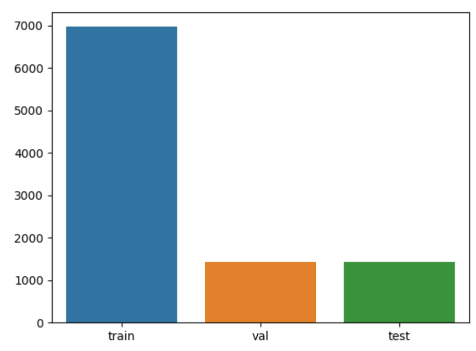
Algorithm 1: Algorithm to generate one Superquadric stack

```
1  $S$: 10 000                                          // Number of scenes
2  $N$: 4                            // Maximum number of objects in the scene
3  $T$: 10                           // Number of placement trials per object
4  $m$: 3                              // Minimum number of objects per scene
5  $SQ_{library}$: 3500                               // Library of superquadrics
6  $(X_{min}, Y_{min}, Z_{min})$: $-25cm$, $-25cm$, $0cm$           // Scene limits
7  $(X_{max}, Y_{max}, Z_{max})$ : $25cm$, $25cm$, $60cm$           // Scene limits
8  for $s$ in $S$ do
9      for $n$ in $N$ do
10         for $t$ in $T$ do
11             $sq \sim SQ_{library}$ $(x, y, z) \sim (X_{min} : X_{max}, Y_{min} : Y_{max}, Z_{min} : Z_{max})$
               $r_x, r_y, r_z \sim$ $(0 : 2\pi)$ if $sq \cap SQ_{scene} == \emptyset$ then
12                 run 1000 simulation steps
13                 if $X_{min} \leq x \leq X_{max}$ & $Y_{min} \leq y \leq Y_{max}$ &
                    $Z_{min} \leq z \leq Z_{max}$ then
14                     $SQ_{scene} = sq \cup SQ_{scene}$
15                     break
16     if $SQ_{scene} \geq m$ then
17         save scene
```

The dataset statistics The generated dataset consists of 9852 scenes which are split to 70%, 15% and 15% into training, validation and test sets. Every scene contains 3 or 4 objects, but scenes with 3 objects are more common (see Figure 5.7 left and centre).

Representing scene geometry

The superquadric parameterisation offers a very concise shape description, with which explicit geometry could be completely omitted. A shape would thereby be defined by its position (x, y, z), its orientation (w, i, j, k for quaternion representation) and its superquadric parameters $(\varepsilon_1, \varepsilon_2, a_1, a_2, a_3)$. However, despite being concise, representing shapes and composite scenes using this compressed representation has several drawbacks. First, predicting the orientation of objects is notoriously difficult and the symmetrical properties of superquadrics pose an additional



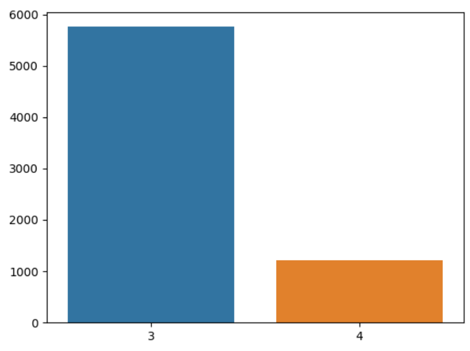


Figure 5.7: Dataset statistics. **Left**: train, val and test split in number of scenes. **Right**: number of scenes per number of objects (note: Only training data is shown since validation and test splits have the same proportion).

hurdle, as multiple object appearances correspond to the same orientation. Secondly, an already compressed shape representation of a number of floating point numbers is not very suitable for feature learning. It would require a well adapted architecture and training procedure. On the other hand, very expressive features can be learnt from 2D images and 3D explicit geometry using standard architectures such as CNNs.

To represent 3D geometry one has a few options, each with their advantages and disadvantages. Pointclouds are memory efficient but lack structure; meshes are dense, but difficult to process with CNNs. Voxels don't scale well, but are the most straightforward to process using 3D CNNs. When using voxel occupancy as a representation, resolution greatly affects the reconstruction due to precision. However, by using a (truncated) signed distance function, the shape can be sampled at infinite resolution and the voxelgrid can be processed using Marching Cubes to obtain smooth meshes.

Implementation details One option to generate a full TSDF scene from individual superquadrics is to overlay each superquadric TSDF grid and take the minimum value of each voxel. However, since each superquadric TSDF grid is in its canonical orientation and position, before creating the overlay, every superquadric grid would have to be rotated and placed within the final (larger) scene grid. This would require interpolation to transfer the TSDF values to the embedding scene grid. A simpler solution is to leverage the position and orientation of each mesh from the scene generated by the physics simulator together with the individual mesh models to create a joint scene mesh. This joint scene mesh can then be processed by an off-the-shelf python library *mesh-to-sdf* to create a scene SDF, which can hereafter be truncated to a TSDF.

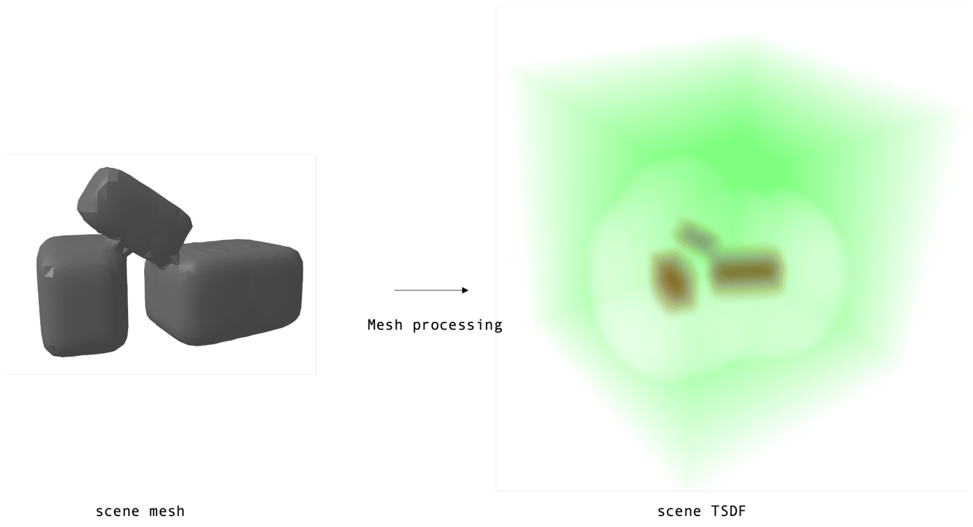


Figure 5.8: Generating a scene TSDF representation from the joint scene mesh using the mesh processing library mesh-to-sdf

Representing instances

Instance segmentation is the task of delineating each object of interest in a particular scene. Going further than semantic segmentation, it captures ”objectness” as well as class, understanding which parts of a scene belong to one individual object instance. Compared to object detection, which provides the centre coordinates and extents of a detected object in the scene — usually in the form of a bounding box — instance segmentation provides labels per scene element and therefore offers a more precise object representation. This also makes it a more complex task than object detection [Bolya et al., 2019].

At the time of writing, the go-to off-the-shelf model for instance segmentation and object detection is Mask RCNN [Kaiming et al., 2017], which extends the object detector Faster RCNN [Ren et al., 2015] with an instance mask prediction branch. Other state of the art instance detection methods include PANet [Liu et al., 2018], TensorMask [Chen et al., 2019] and YOLACT [Bolya et al., 2019]. Although these methods generate impressive results on very cluttered 2D images and can handle many overlapping instances, they are coupled with semantic segmentation — they can only predict instances which belong to a certain semantic class. Recently, some works have explored unknown object segmentation in 2D [Durner et al., 2021] [Xie et al., 2020], whereby [Xie et al., 2020] use a Hough Voting scheme in which each pixel votes for its object centre. Compared to approaches which produce a fixed number of predictions coupled with objects existence [Carion et al., 2020], voting-based methods do not rely on a pre-defined maximum number of instances in the scene.

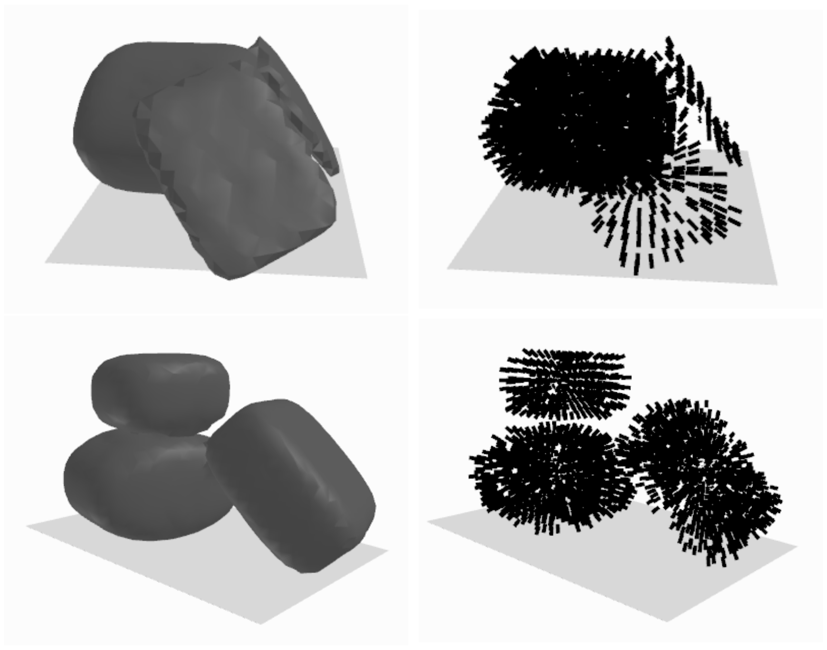


Figure 5.9: Representing instances as centre voting vector fields. **Left**: The scene mesh generated from the TSDF using Marching Cubes. **Right**: The centre voting fields for the respective scene meshes on the left.

Several approaches have extended instance segmentation to 3D scenes. [Hou et al., 2019] backproject 2D features from RGB-D scans into a 3D voxelgrid and generate per-voxel mask predictions. Similarly to [Yang et al., 2019], they rely on semantic classes for instance mask predictions and do not work with unknown objects. [Engelmann et al., 2020] adopt the object-centric approach by aggregative object centre votes within a pointcloud, but nevertheless couple their prediction with semantic classes.

For improved generality, being able to detect entities as objects without knowing their class is preferable. In this work, the aim is to provide instance segmentation which is class-agnostic and independent of the number of objects present in the scene.

Representing instances using a voting vector field Voting based algorithms are inherently independent of class and also don't limit the number of objects in a particular scene. A suitable representation for this goal is therefore a vector field of centre votes, whereby each voxel in our scene grid contains a unit vector, starting from the voxel centre and pointing in the direction of the object's centre. Voxels which don't belong to objects are defined with a vector of $(0,0,0)$. Choosing this representation for ”no object”, we have to assume that the centre of an object will never exactly coincide with the centre of a specific voxel. Using unit vectors alleviates the network from having to predict the object size in addition to its centre location and the vector direction is enough to extract the object centres during post processing. Based on this representation, we design the network output to be a grid of voxels which ”vote” for the location of the centre they belong to.

Class-agnostic instance segmentation with 3D Hough Voting

Extracting the object centres based on the network prediction can be achieved by leveraging Hough Voting and adapting it to a voxelgrid representation.

Hough Voting Initially introduced in 1959 [Hough, 1959] as the Hough Transform, Hough Voting is a feature detection algorithm, isolating particular shapes within an image. It translates the problem of pattern detection in point samples to detecting peaks in feature space. In its classical form it is applicable to simple shape features such as lines or curves. In the example of detecting lines, the Hough Transform transforms all point samples into the parameter space of lines, defined by a and b, given the line equation $y = ax + b$. In this parameter space, all points which lie on a line defined by specific values of a and b, will intersect and can be collected. The idea behind the algorithm is, that each measurement provides a contribution to a globally consistent solution. Hough Voting was later extended to its general form, allowing for arbitrary shapes [Ballard, 1981], in particular applying it to image patches as indicators for the existence of a complex object. Before the wide adaptation of deep learning models, voting was used by several methods for object detection [Leibe et al., 2007, Knopp et al., 2010, Velizhev et al., 2012, Lehmann et al., 2010] and its performance was improved for 3D shapes by [Woodford et al., 2013] by introducing the minimum-entropy and intrinsic Hough Transforms. With their method *Vote3D*, [Wang and Posner, 2015] demonstrated how voting is effectively equivalent to sparse convolutions with a linear classifier and use their method to efficiently apply a sliding window approach on a full 3D voxelgrid structure for object detection in point clouds. More recently, Hough Voting has been combined with deep learning methods for object detection in point clouds [Qi et al., 2019a]. Inspired by previous applications, we formulate instance segmentation as a Hough Voting task, described below.

Instance segmentation We assume a network prediction of a vector field in which each voxel ”votes” for the object it belongs to by predicting a 3D unit vector $\hat{\mathbf{c}}$ from its own centroid to the object’s centroid. (see Figure 5.10). At a 10-fold higher resolution than our voxel grid (640×640) , we count the number of rays that traverse each voxel by marching each ray through the grid. We use the object bounds obtained from the TSDF prediction to limit raycasting to the inside of objects. A higher resolution allows for more detailed prediction of centers which don’t coincide with voxel centers at the original resolution. Those voxels with a number of traversals larger than μ are selected as object center votes and passed to MeanShift to obtain the final center locations. To allocate voxels to their corresponding centers, we compute:

$$\min_{c_1,\ldots c_N} |\arccos(\hat{\mathbf{c}} \cdot \hat{\mathbf{c_n}})| + \gamma * \|\hat{\mathbf{c_n}}\| \tag{5.6}$$

whereby $\hat{\mathbf{c_n}}$ is the distance from the voxel's centre to cluster centre n. Setting hyperparameters $\mu = 10$ and $\gamma = 0.1$ gave the best performance. While we observe limited sensitivity to γ, very large (≥ 25) and small (≤ 4) values of μ increase under- and over-segmentation, respectively. Paralellized on the GPU our method takes on average $19ms$ for a scene resolution of 64^3.

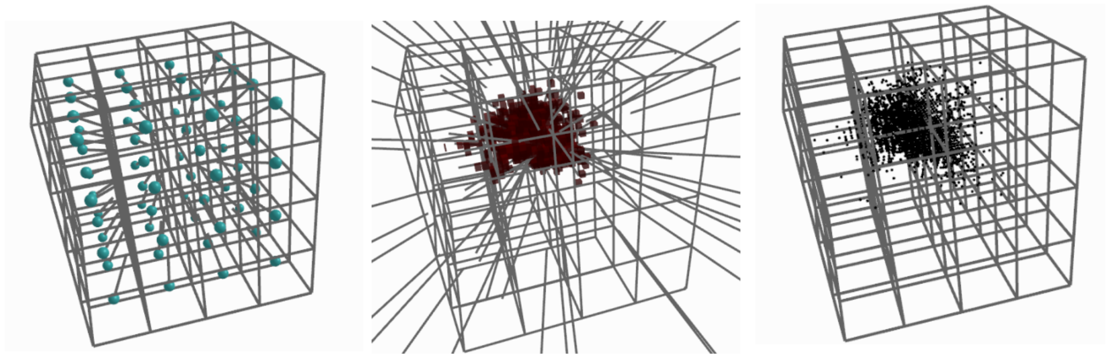


Figure 5.10: Our Hough Voting method illustrated for a single object center. **Left**: The network output — unit vectors pointing towards the center of the object. **Center**: Raycasting and extracting those voxels with a high number of traversal at a 10-fold voxelgrid resolution. **Right**: Voxels with a high ray-traversal rate are likely close to the object center and are selected as 'votes' for this object center. To obtain final object centers, selected votes are further processed using Mean shift clustering.

Scene setup

To capture enough detail, we choose to represent each scene with a voxelgrid of $1m$ at a resolution of 64^3 compared to many approaches that use a resolution of 32^3 (e.g. [Maturana and Scherer, 2015]). For training, a consistent scene setup has to be chosen. Placing the object collection in the centre of the voxelgrid is the most straightforward choice, but removes a constant from the dataset — the floor location — which could make the task easier for the network. We therefore choose to place all scenes at the x, y centre of the voxelgrid and at a fixed floor position of $z = 9.6cm$ (15 voxel rows). Given that our scene representation is a TSDF, placing the object stack at $z = 0$ would result in some partially defined surfaces, since a TSDF defines surfaces by the values surrounding it. The camera that renders synthetic depth images is placed relative to the scene centre at $x = 20.5cm, y = 20.5cm, z = 9.6cm)$. We visualise an example of our scene setup in Figure 5.11.

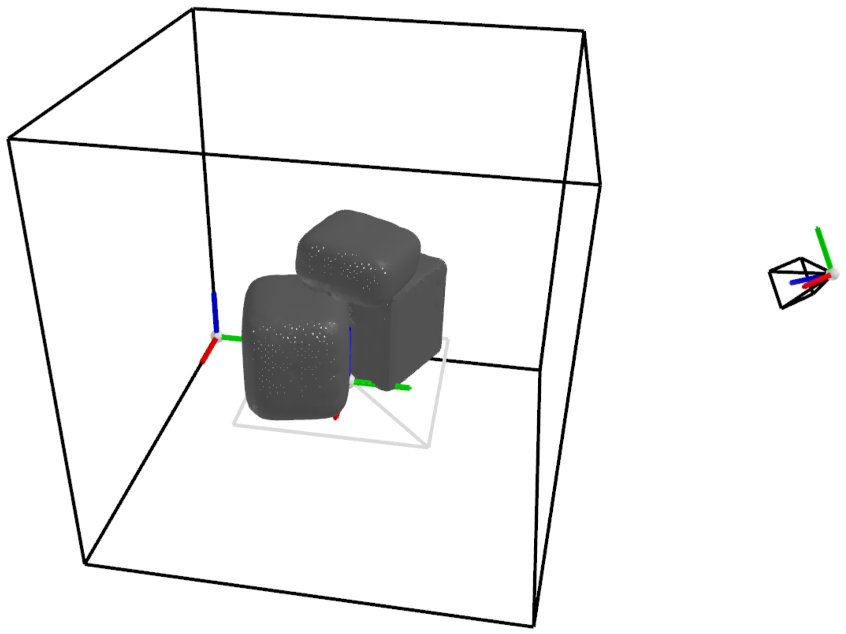


Figure 5.11: Our scene setup using a discrete TSDF representation of stacks of superquadric shapes.

Rendering depth using TSDF raytracing

In order to obtain 2D data from our 3D scene to train on, the scene has to be projected onto a 2D image plane through rendering. Rendering can be implemented using rasterization or raytracing (see Section 3.5). Although rasterization is faster, it is less precise compared to raytracing. Since our rendering method does not have to run in real-time, we choose raytracing and implement it on the GPU to maximise efficiency.

Raytracing implementation details Raytracing is inspired by how we perceive things in the real world — photons travel through space, interact with objects, get reflected and carry information about the material they interacted with, with them. In ray tracing, we follow imaginary rays sent out from the camera lens — one ray per image pixel — and traverse space until we encounter objects. In a TSDF voxel grid this means stepping along each ray in step sizes s and probing the content of the voxelgrid until hitting a surface, i.e. encountering a zero crossing. Implemented on the GPU, our algorithm can step in parallel for every ray, logging the distance travelled and reporting it back as a depth value, once a surface has been reached. For camera positions outside of the scene grid, the intersection with the closest scene border is found by line-plane intersection from which point onward we raytrace the voxel volume. Those rays that exit the scene grid without having intersected an object surface or the floor, are set to the default maximum depth value of 1000. To accelerate our algorithm we use *sphere tracing* [Hart, 1996], which steps along the ray

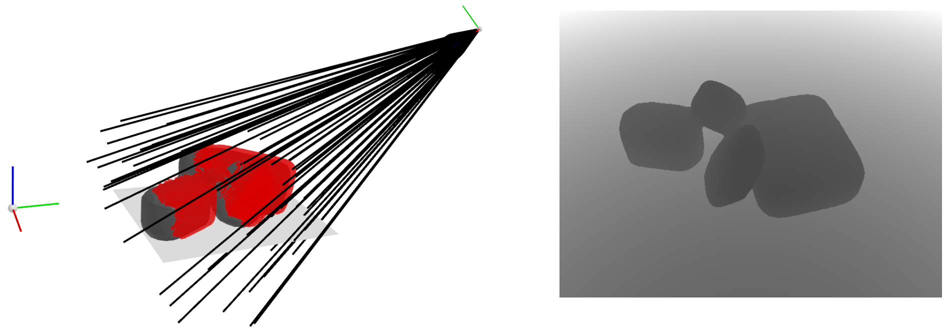


Figure 5.12: **Left**: illustration of the raytracer in 3D. Ray-surface intersections are highlighted in red and the number of rays is subsampled for visibility. **Right**: the rendered depth image.

direction using the current value of the TSDF.

The raytracing module is parallelised on the GPU using the cupy library, which allows for smooth integration of CUDA kernels with python code. With a kernel for every ray, we launch $w \times h$ kernels for every rendering call, according to the camera resolution. For a resolution of 640×480, our implementation renders a depth image in $0.2ms$.

We visualise the raytracing process and an example of a rendered depth image in Figure 5.12.

Training setup

To be useful in real-world settings, the method has to be able to function for any reasonable viewpoint and be independent of the relative position of the camera with respect to the object stack. To this end, the method has to be trained on random viewpoints.

Random viewpoint generation To achieve the highest variability in viewpoints and to avoid further increasing the size of the dataset, the viewpoints are generated online — during training. For every new viewpoint, a camera-centre position is sampled uniformly from a region around the scene centre x, y, z, whereby x and y are sampled from $-1m$ to $1m$ and the camera height z is sampled between $10cm$ and $1m$. To ensure the camera is not placed too close or inside an object, positions with a distance below $78cm$ from the scene centre are discarded. This results in less samples drawn from positions close to the scene centre and a distribution of poses as displayed in Figure 5.13.

Data augmentation To avoid any bias in appearance along the x or y axis, we augment our dataset with random rotations for each individual scene. To make the final method more robust to misalignment between the camera and the scene centre, we additionally add small scene perturbations

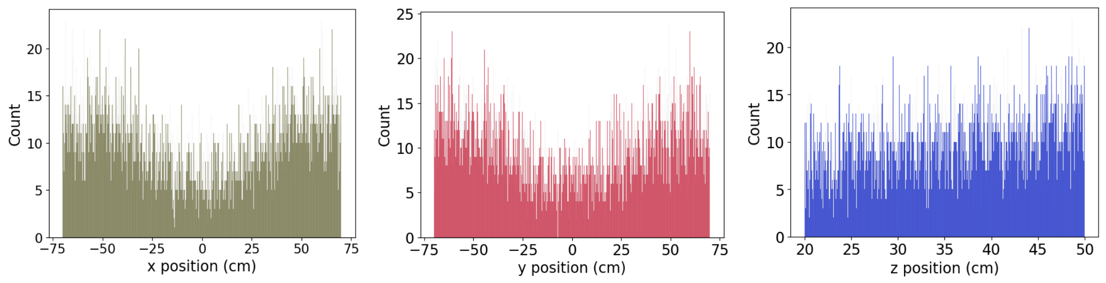


Figure 5.13: Distribution of poses after 10000 samples along the x, y and z axes. Samples are taken uniformly, however samples whose overall pose results in a distance below 50 from the scene centre at $(0,0,0)$ are discarded, leading to the dip around zero for x and y and the drop in lower positions for z.

of a few centimetres around the centre.

Data pre- and postprocessing

Network input Our task is to generate a 3D scene with shape and instance segmentation from a depth view, which is used to condition a 3D scene-prior VAE. Conditioning a 3D CNN on 2D depth image features would require the network to implicitly learn a reprojection task. To let it focus on the main task, we convert the input depth image into a partial TSDF by reprojecting depth into the scene. We use inverse raytracing for this and sample along each ray to fill the voxels in the camera view frustrum with the closest distance to a surface. The algorithm is parallelized on the GPU and steps along each ray asynchronously. For each ray, steps of 25 mm are taken until $1cm$ behind the surface, which is defined by the depth value at the current pixel. At each step, the current distance to the surface is propagated to the neighbouring voxels using trilinear interpolation: after obtaining the relative weights from trilinear interpolation between the current sample position d_s and all neighbouring voxel centres v_n for $n = 0, \dots, 4$, every voxel centre v_n is updated with the weighted distance from the surface

$$v_n = w_{tril_n} * d_s, \tag{5.7}$$

where $w_{trilinear_n}$ is the weight for v_n obtained from trilinear interpolation. Parallelized, our algorithm generates a partial TSDF in under 0.8 seconds and can be used in online training.

Shape refinement using superquadric fitting Our system outputs a TSDF and voxelised instance segmentation of the compound object; if needed, additional post-processing can be applied (e.g. primitive shape fitting, CAD-model fitting). To generate a compact representation, we propose the following refinement procedure, which extracts a set of superquadrics with their poses from the raw network output:

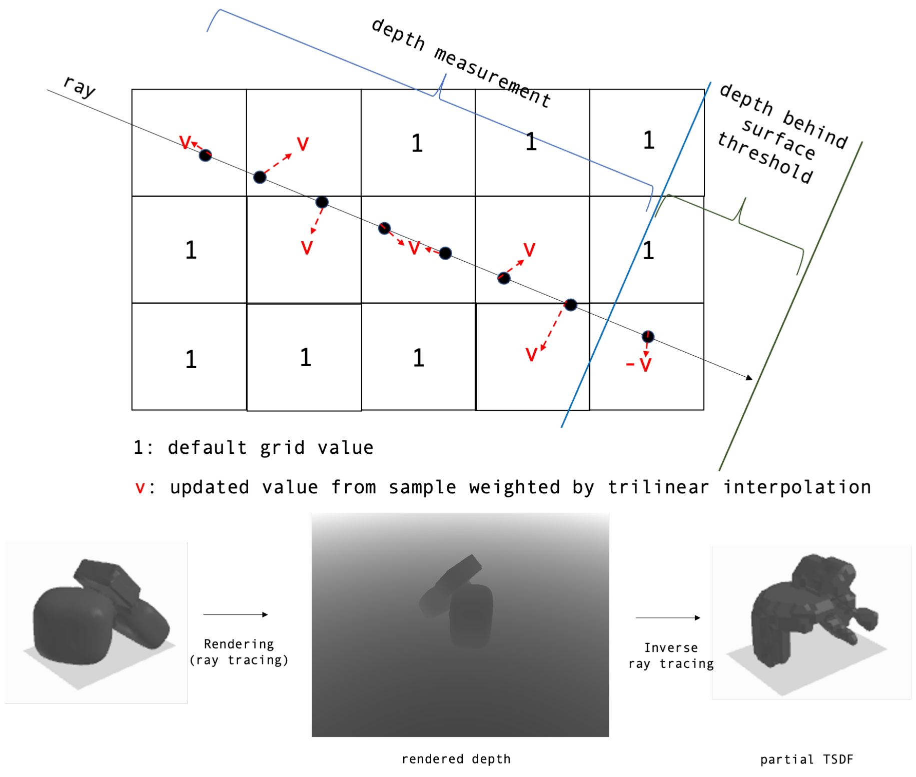


Figure 5.14: **Left**: illustration of the inverse ray tracing algorithm generating the partial TSDF. **Right**: example of a partial TSDF generated from a depth image.

First, we run marching cubes on the TSDF and sample from the resulting mesh to get a point cloud. We segment this point cloud according to the voxelised instance predictions. We then independently fit a superquadric to the N_i points belonging to the ith segment by minimising the "mean distortion" [Chevalier et al., 2003], $L_{\text{dist},i}$, given by:

$$L_{\text{dist},i} = \frac{1}{N_i} \sum_{k=1}^{N_i} \sqrt{a_{i1}\, a_{i2}\, a_{i3}} \cdot d_i(x_{i,k}, y_{i,k}, z_{i,k})^2, \tag{5.8}$$

where

$$d_i(x_{i,k}, y_{i,k}, z_{i,k}) = \|\overrightarrow{O_i P_k}\| \frac{f_i(x_{i,k}, y_{i,k}, z_{i,k}) - 1}{f_i(x_{i,k}, y_{i,k}, z_{i,k})}, \tag{5.9}$$

a_{i1}, a_{i2}, a_{i3} are the scale parameters and $f_i(x_{i,k}, y_{i,k}, z_{i,k})$ is the implicit equation from Eq. 5.1, $d_i(x_{i,k}, y_{i,k}, z_{i,k})$ is the approximate Euclidean distance to the surface [Bardinet et al., 1995], and $\|\overrightarrow{O_i P_k}\|$ is the distance from the point to the superquadric center.

We then optimise for the parameters of all Q superquadrics together while adding in additional cost terms to penalise collisions between superquadrics and intersection with the floor plane:

$$L = \sum_{i}^{Q} \left[\lambda_{\text{dist}} L_{\text{dist},i} + \lambda_{\text{coll}} \sum_{\substack{j=1 \\ i \neq j}}^{Q} L_{\text{coll},ij} + \lambda_{\text{floor}} L_{\text{floor},i} \right]. \tag{5.10}$$

The cost of sample points from superquadric j colliding with superquadric i, $L_{\text{coll},ij}$, is given by:

$$L_{\text{coll},ij} = \frac{1}{N_j} \sum_{k=1}^{N_j} \mathbb{1}_{d_i(x_{j,k},y_{j,k},z_{j,k})<0} \cdot d_i(x_{j,k},y_{j,k},z_{j,k})^2 \tag{5.11}$$

where $\mathbb{1}$ is the indicator function, and the cost of sample points from superquadric i colliding with the floor plane, $L_{\text{floor},i}$, is given by:

$$L_{\text{floor},i} = \mathbb{1}_{z_{i,k}<0} \cdot z_{i,k}^2. \tag{5.12}$$

Experimentally, we found $\lambda_{\text{dist}} = 100$, $\lambda_{\text{coll}} = 10$, and $\lambda_{\text{floor}} = 1$ to work well. For both stages, we minimised the cost function using the dogleg algorithm with rectangular trust regions, bounding the superquadric parameters to enforce convex shapes.

5.2.2 Baseline experiments: 2D $\Rightarrow$ 3D shape & instances

Before learning a realistic scene prior, we evaluate how well a simple feed-forward learning-based approach can predict the 3D shape and instance segmentation of a stack of synthetic superquadrics. At the time of this work, there is no directly comparable work on 3D instance segmentation for object groups and therefore the following two baselines are selected:

SSCNet Baseline The closest existing method is SSCNet [Song et al., 2017], which predicts a 3D occupancy grid with semantic labels for multi-object rooms from a single depth image. As this method does not predict instances, we use it as a baseline only for geometric reconstruction and adapt it as follows: (1) we change the last layer to predict a TSDF; (2) we remove downsampling (required only in large-scale scenes) from layers by adding padding as well as one upsampling layer. We refer to this modified version as *SSCNet***. A detailed overview of this adapted version of SSCNet can be seen in Figure 5.15.

Fully Convolutional (FC) baseline As a second baseline, we use a fully convolutional network predicting a TSDF and instance vector field from a single depth image. The input is encoded using a partial TSDF encoder branch that includes residual blocks [He et al., 2016] with Squeeze and Excitation (SE) [Hu et al., 2018], to be consistent with the main architecture (see Section 5.2.3). The decoder splits into two task specific decoder branches. This serves both as a baseline for instance segmentation and an ablation study, showing the advantage of a learned prior over direct prediction. We used no feature space compression, as we found it reduced convergence. A detailed illustration of the FC Baseline can be found in Figure 5.16

Note that as an additional baseline, a simple model-based fitting approach could be considered. However, although such an approach would be able to model object-specific occlusions, i.e. the objects which appear in the partial view will be completed using shape-symmetry, a model-based fitting approach is by design unable to model any fully hidden objects; it would therefore only provide a relevant baseline for scenarios where every objects is at least partially visible.

Loss function

When predicting a TSDF, areas close to the surface with lower values will incur a smaller penalty than more distant positions. Other methods [Xu et al., 2019b] have proposed masking the loss by applying a different weight near the surface. We observe that this approach adds a discontinuity to the loss as well as an additional hyperparameter δ for the mask area. Instead we propose a loss which adds the inverse of the TSDF itself as a weighting factor:

$$L_{\text{TSDF}} = \sum_{x=0}^{X} \sum_{y=0}^{Y} \sum_{z=0}^{Z} \frac{1}{|v_{x,y,z}| + \varepsilon} ||v_{x,y,z} - \hat{v}_{x,y,z}||_2, \tag{5.13}$$

where $v_{x,y,z}$ and $\hat{v}_{x,y,z}$ are the ground truth and predicted TSDF values at voxelgrid index (x,y,z), ε prevents division by 0 (set to $1e^{-9}$) and $||.||_2$ is the L_2-norm. We use the same loss for the partial TSDF prediction and refer to it as L_{p_TSDF}. For the instance vector voting task, we use the L_2-norm between predicted and ground truth center vectors $\hat{\bar{\mathbf{c}}}$ and $\bar{\mathbf{c}}$:

$$L_{\text{center\_votes}} \quad = \quad \sum_{x=0}^{X} \sum_{y=0}^{Y} \sum_{z=0}^{Z} ||\bar{\mathbf{c}}_{x,y,z} - \hat{\bar{\mathbf{c}}}_{x,y,z}||_2. \tag{5.14}$$

We found that this led to better convergence for the instance segmentation task than using cosine similarity.

Training details

We train both baseline models until convergence on the validation set, which we observed after 80 and 92 epochs respectively. We use the Adam optimiser [Kingma and Ba, 2015b] with a learning rate of 1^{-3}, decreasing every 8 epochs by a factor of 0.99.

Results

The results on the synthetic datasets show that both methods learn to reconstruct the visible region well from random viewpoints, but struggle to reconstruct occluded regions, as demonstrated in the qualitative results in Figures 5.17 and 5.18 and our quantitative results in Table 5.1. In the following sections, we propose our method, which generates realistic 3D shape and instances in visible as well as occluded regions.

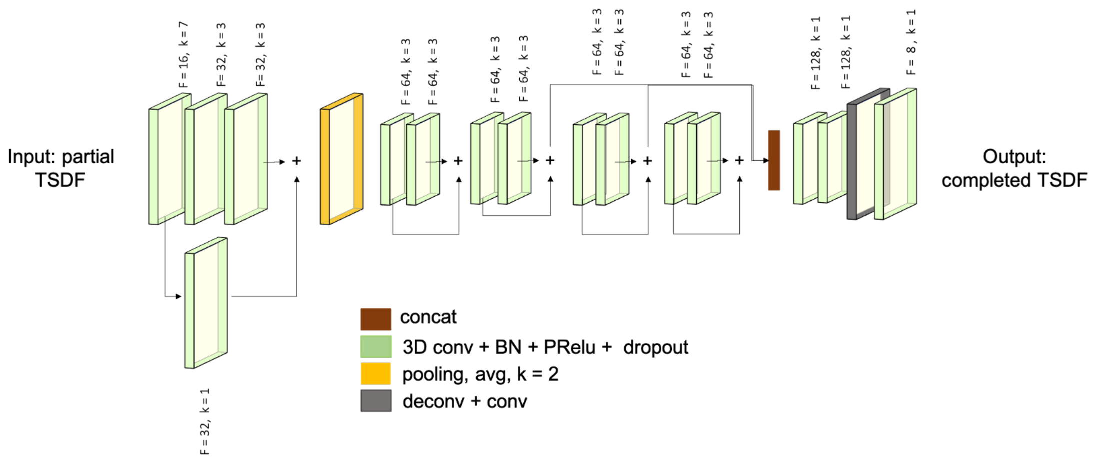


Figure 5.15: Baseline model based on the *SSCNet*** architecture adapted from the original model presented in [Song et al., 2017]. Our adaptation outputs a TSDF instead of an occupancy grid and removes downsampling steps only required for large-scale scenes.

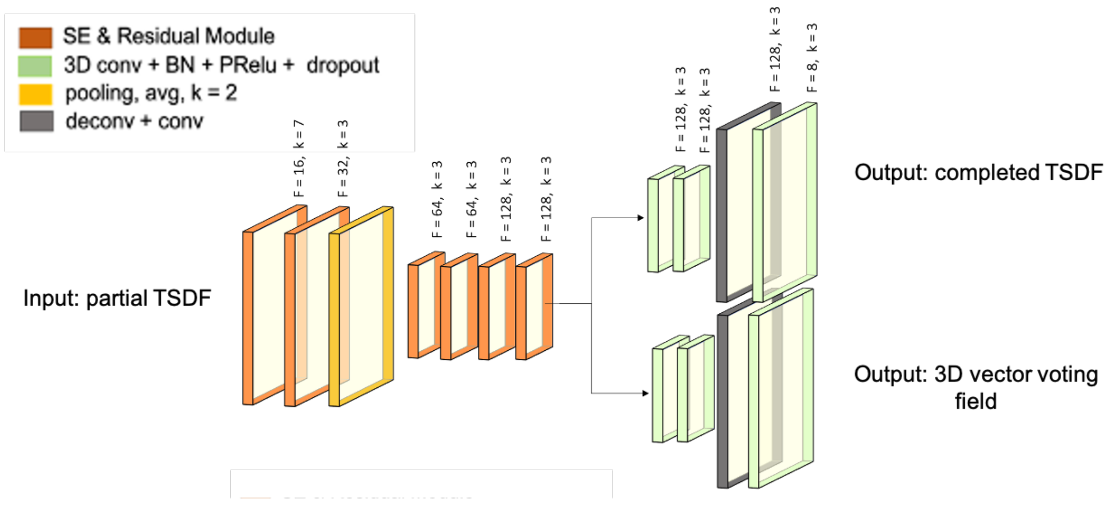


Figure 5.16: Fully convolutional (FC) baseline model architecture. Full TSDF and instance segmentation (modeled as a vector voting field) are predicted from a partial TSDF obtained from a single view of the scene. Please refer to Figure 5.19 for the SE Residual Module.

5.2.3 A conditional VAE for 3D shape and instance prediction

Learning a 3D scene prior for object stacks

As introduced at the beginning of this chapter, previous works have addressed reasoning for unobserved 3D space in two predominant ways: leveraging a-priori knowledge about object symmetries, or leveraging learnt shape priors. Unordered object stacks do not have inherent symmetry; however, we observe that physics constrains decomposition as well as shape in occluded regions and hypothesise that a latent space learnt from scenes built under physics simulation can serve as a prior to better predict shape and instances in occluded regions. We propose to learn this prior

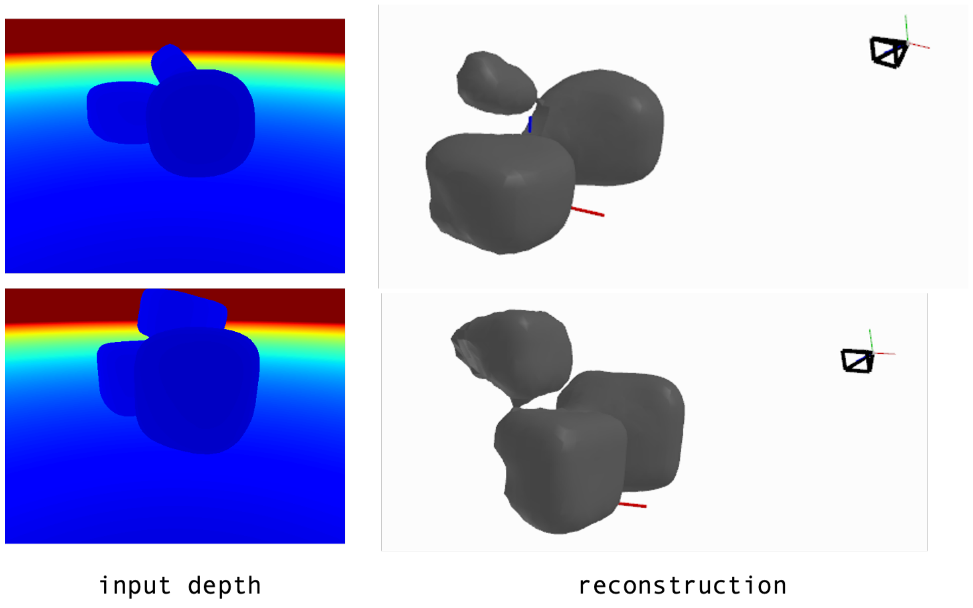


Figure 5.17: Qualitative reconstruction results of the *SSCNet** baseline. Quantitative results can be found in Table 5.1.

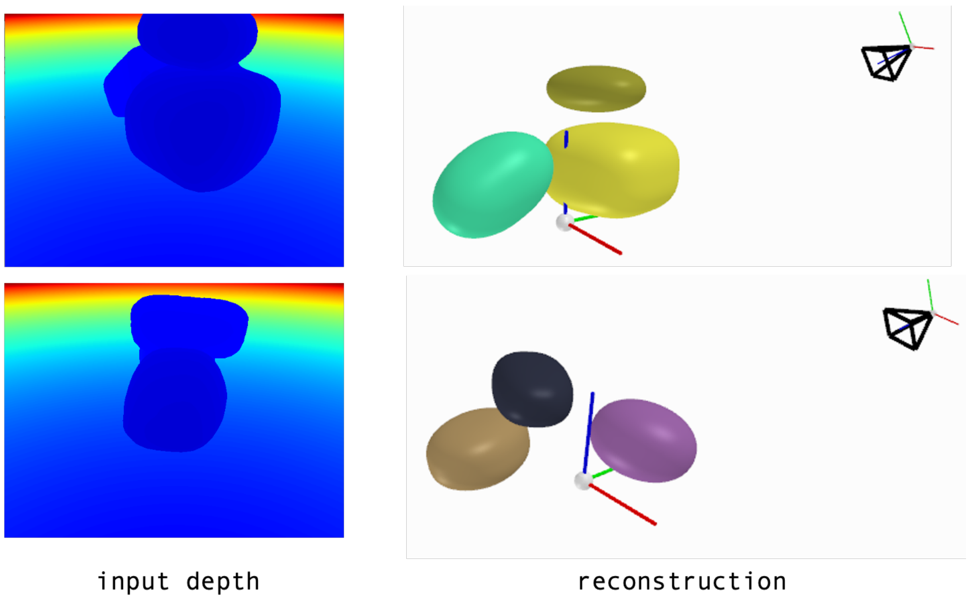


Figure 5.18: Qualitative reconstruction results of the fully convolutional baseline. Quantitative results can be found in Table 5.1.

distribution using a deep generative model. Within deep learning, generative models are common methods to approximate distributions from data and for our prediction task we select variational autoencoders (VAE), a generative method that leverages variational approximation of the target distribution. For the given task, the architecture of the VAE has to encode and decode two modalities: the TSDF and the instance vector field. To this end, a 3D CNN with two encoding and decoding branches is suggested, joined in a common bottleneck. We choose Residual blocks [He et al., 2016] with Squeeze and Excitation (SE) [Hu et al., 2018] over conventional convolutional layers in the joint encoder and decoder of the VAE to avoid gradient underflow and favour context integration. To learn a joint latent representation of the geometry and instance segmentation of a scene, both TSDF and instance segmentation are compressed into one latent code of size 96. The architecture is illustrated in Figure 5.19.

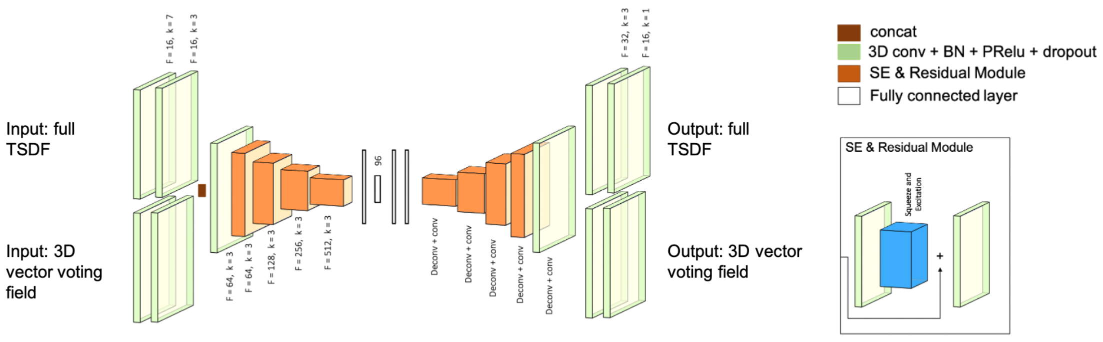


Figure 5.19: Shape and instance VAE architecture. A shape prior is learnt, by encoding geometry and instance segmentation (modeled as a voting field) into one joint latent code.

As a first experiment, we verified that the common bottleneck produces a joint and consistent latent representation of both shape and instance segmentation. To this end, we train our VAE architecture on the dataset and inspected the sample space. Formally, our aim is to train a generative model which learns the joint distribution over the data x and the latents z: $p(x,z) = p(x|z)p(z)$ [Kingma and Welling, 2019]. Within our Variational Autoencoder framework, $p(z)$ is a zero-mean multi-variate gaussian prior describing realistic pile configurations and the stochastic encoder $p(z|x)$ approximates the true posterior. $p(x|z)$ is a stochastic decoder which, given a sample from the learnt latent space, produces a realistic pile composition. In this initial experiment, we find that the learnt latent space produces realistic pile compositions when sampled and that interpolation between two random scenes shows realistic intermediate scenes as well as smooth transitioning, keeping shape and instance consistent.

To enable 3D shape and instance prediction from 2D depth at test time, depth information has to be integrated in the training process, This is achieved in form of a depth-conditioned VAE architecture, described in the following section.

A depth-conditioned VAE

Ultimately, the goal is to predict 3D shape and instances from a single depth image. In the initial experiments, we found that a lower dimensional latent space can be learnt, which jointly encodes 3D shape and instances of multiple objects in randomly composed stacks. To leverage this latent space as a prior in a *depth to 3D prediction* task, depth information has to be integrated into training. In accordance with the baselines, these views should be randomly sampled around the scene for view-independent performance of the model. A straightforward solution is to condition the scene-prior VAE on extracted depth features during training. This also allows the latents to focus on learning the occluded regions, while the reconstruction of the visible scene is guided by the depth information. The training task can now be reformulated as $p(x|\gamma,z) = p(z|x,\gamma)p(z)$ where $p(z|x,\gamma)$ now also depends on the depth encoding γ.

Network architecture details

The main VAE network architecture consists of the 3D scene prior VAE which learns a latent space for realistic pile configurations, and a conditioning network whose encoding layers feed their learned feature maps to the VAE, conditioning it on partial observations.

Conditioning autoencoder To learn descriptive depth features, the conditioning network is trained as an autoencoder, encoding and decoding the partial TSDF generated from the input depth view. Encoder and decoder each have 5 convolutional layers with 2 linear layers compressing the feature maps into a 1D bottleneck of 96. The encoder feature maps are used to condition the encoder and decoder of the shape and instance VAE.

Shape and instance VAE Our VAE's encoder maps input TSDF and instances to a common feature space using two convolutional layers for each modality. The resulting feature maps are concatenated and further compressed using 5 joint encoding layers. One linear layer maps our 3D feature space to our 1D latent code of size 96 while 2 linear layers map it back to 3D. Our VAE 3D decoder mirrors our encoder. We use batch normalisation (BN) and PRelu activations in all of our hidden layers. Our architecture can be seen in Figure 5.20.

Inference At inference time, the encoder of the shape and instance VAE is dropped and only the generative decoder is leveraged to probabilistically complete the occluded regions, based on the features obtained from an input depth image, by sampling from the learnt scene prior. Figure 5.21 illustrates the architecture at test time.

Loss function

For TSDF reconstruction and instance segmentation we use the loss functions L_{TSDF} and L_{center_votes} introduced for our baseline experiments in Section 5.2.2. To approximate our prior, we empirically found that training using the Maximum-Mean Discrepancy D_{MMD} [Zhao et al., 2017b] led to better convergence than using the standard KL-divergence [Kingma and Welling, 2014]. Our final loss is composed of 4 components:

$$L_{total} = \alpha L_{TSDF} + \beta L_{p\_TSDF} + \gamma L_{center\_votes} + \delta D_{MMD}. \tag{5.15}$$

We set $\alpha = \beta = \gamma = 1$ and δ to $1e^5$.

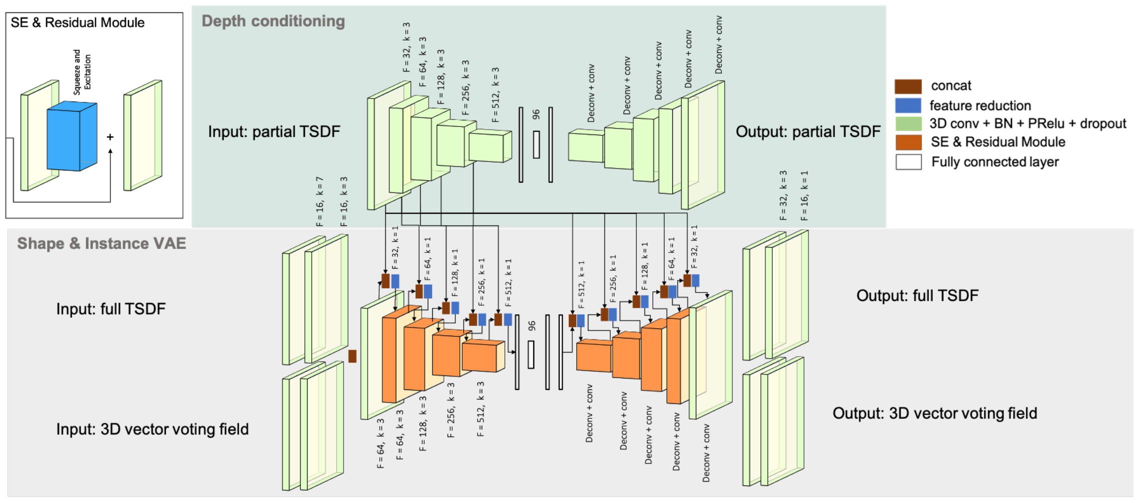


Figure 5.20: Depth-conditioned shape and instance VAE architecture. The Shape and Instance VAE is trained jointly with the depth feature encoding network. Input depth is transformed into a partial TSDF to simplify the training task.

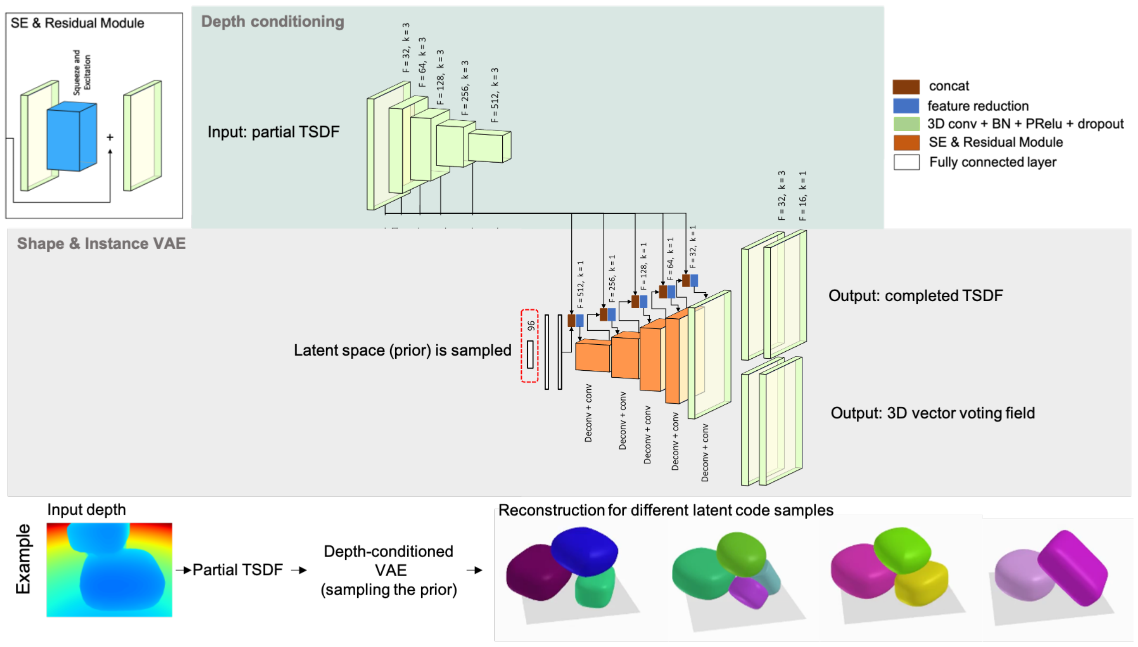


Figure 5.21: Depth-conditioned shape and instance VAE architecture at inference time. 3D shape and instance VAE encoder is dropped and samples are drawn from the learnt prior. Given a randomly drawn sample and conditioning features extracted from a single depth view of the scene, full scene reconstruction is generated. While the visible reconstruction remains constant, the occluded regions are probabilistically reconstructed based on different samples drawn from the prior.

Training details

We train both models (shape and instance VAE, depth-conditioning autoencoder) jointly, until convergence on the validation set, which we observed after 130 epochs. We use the Adam optimiser [Kingma and Ba, 2015b] with a learning rate of 1^{-3}, decreasing every 5 epochs by a factor of 0.99. By using the Maximum Mean Discrepancy instead of the KL distance, we do not need to follow the warm-up scheme suggested by [Higgins et al., 2017] and keep the weighting δ of D_{MMD} at $1e^5$ throughout the entire training. At training time, the model input consists of both a randomly sampled viewpoint (see Section 5.2.1) and the full 3D model of the scene (TSDF and 3D vector field).

5.2.4 Experiments and evaluation

In the following, we present the experiments with which the proposed method is tested and evaluated. The first experiments are performed on synthetic data to verify that our method works for unseen object stacks and outperforms the baselines. To ensure that the method can be employed for real scenarios, its performance is also tested on real data. Overall we aim for a method which 1) accurately reconstructs the visible 3D shape of a scene of multiple stacked objects, 2) generates an estimate of the occluded regions of the scene, consistent with the visible region, 3) segments the objects in the scene into instances, 4) generalises to real data. To this end, the following sections present our experiments for quantitatively evaluating reconstruction accuracy, instance segmentation (scene decomposition) as well as qualitative examples demonstrating realistic proposals for occluded regions and good performance on real data.

Test datasets

The proposed method is evaluated quantitatively and qualitatively on two synthetic datasets as well as a number of real data examples. For the evaluation on synthetic data, we use our own test dataset of superquadrics stacks (1419 scenes) as well as a dataset composed of YCB objects (256 scenes) to demonstrate generalisation to very different objects. The latter contains YCB objects with IDs: *2, 3, 4, 5, 7, 8, 9, 10, 36, 61* which can be approximated by a single superquadric. To generate object stacks, the same procedure is used as for the synthetic superquadric scenes (Section 5.2.1). Examples of the YCB test dataset are displayed in Figure 5.22.

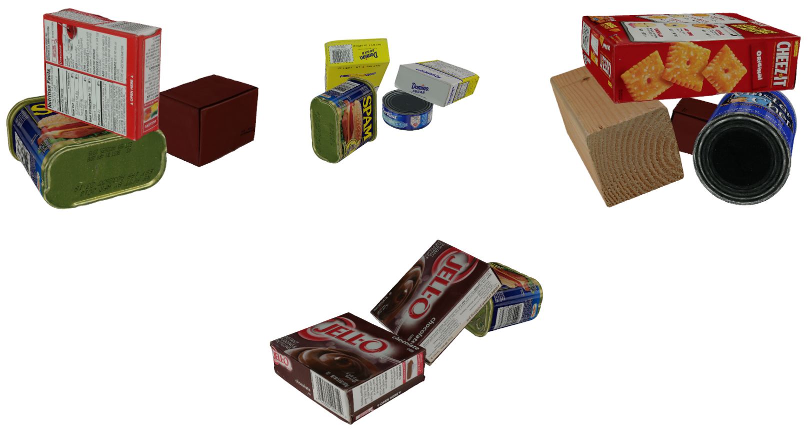


Figure 5.22: YCB dataset examples. All scenes are generated by random placement under physics simulation with *Pybullet*.

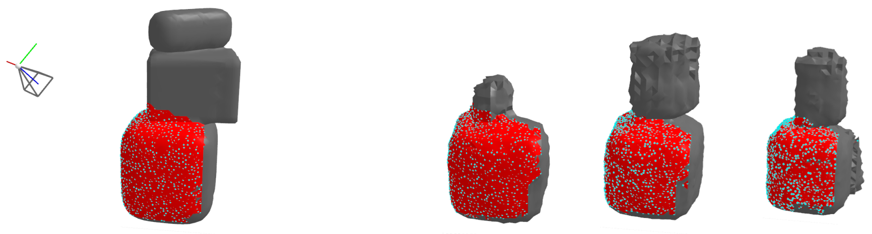


Figure 5.23: **Left**: Ground truth with the visible part of the mesh highlighted in red. **Right**: 3 random latent code samples generated by our network. Note how every sample has different reconstructions in the occluded areas, which are plausible but would yield high reconstruction error with respect to the ground truth TSDF.

Reconstruction accuracy

Evaluation procedure Comparing the full reconstruction to the ground truth scene will unfairly penalise the network in occluded regions for reconstructions which are plausible, but differ from the ground truth (see Figure 5.23). We therefore evaluate our method on the visible surface area and the full reconstruction separately. To show how performance differs depending on how much of the object stack is visible from a given viewpoint, we report results by surface visibility. Note that given a single view, the maximum surface visibility of a pile is around 50%, since the back will always be occluded. Since our method is viewpoint agnostic, we evaluate every test scene from 3 random viewpoints uniformly sampled from the same ranges used during training (see Section 17). We ensure the distance between camera and scene centres is at least $70cm$. For every viewpoint, we generate a 3D scene and compute the average reconstruction accuracy for 3 latent code samples to account for variability.

Visible surface evaluation To evaluate surface reconstruction accuracy, we extract a mesh from the generated and ground truth TSDFs using marching cubes. We obtain the visible surface by extracting all visible faces using ray-triangle intersection with the generated mesh. For every ray we find all faces it intersects and extract the closest one. We produce 1000 uniform samples from the extracted surface and evaluate the bidirectional chamfer distance between ground truth and predicted point sets, P_m and P_k:

$$CD = \frac{1}{N_m} \sum_{x_i=0}^{N_m} \min_{x_j \varepsilon P_k} (\mathbf{x_i} - \mathbf{x_j}) + \frac{1}{N_k} \sum_{x_i=0}^{N_k} \min_{x_i \varepsilon P_m} (\mathbf{x_j} - \mathbf{x_i}) \ , \tag{5.16}$$

where $\mathbf{x_i}$ and $\mathbf{x_j}$ are the 3D coordinates of sampled points from P_m and P_k respectively.

Full reconstruction evaluation We evaluate the full reconstruction in terms of surface reconstruction accuracy and predicted voxel occupancy. For the former we compute the chamfer distance between 1000 samples from the full predicted and ground truth mesh surfaces. For the latter we compute the binary cross entropy (BCE):

$$-\frac{1}{N} \sum_{\Omega} y_i \log(\frac{1}{S} \sum_{j=0}^{j=S} \hat{y}_j) + (1 - y_i) \log(1 - \frac{1}{S} \sum_{j=0}^{j=S} \hat{y}_j) \ , \tag{5.17}$$

where N is the voxelgrid resolution 64^3, S is the number of latent code samples taken per viewpoint, which we set to 3 and $\hat{y}$ and y are the predicted and ground truth occupancy value respectively, computed as follows:

$$Occupancy_{x,y,z} = \begin{cases} 1 & \text{if } TSDF_{x,y,z} > 0 \\ 0 & \text{if } TSDF_{x,y,z} <= 0 \end{cases} \tag{5.18}$$

Scene decomposition (instance segmentation)

Similarly to the reconstruction accuracy, it is impossible to make an exact quantitative evaluation of our method's instance segmentation since decomposition estimates in occluded regions can be plausible even when very different from the ground truth. In Figure 5.30, for example, our method hypothesises plausible hidden objects, although the ground truth does not contain one. We therefore decided to evaluate instance segmentation as a valid scene decomposition which generates stable piles in physics simulation.

Stability evaluation under physics simulation We evaluate stability by loading our generated 3D meshes into the PyBullet physics engine and simulating 10000 steps with a gravity setting of of $10\frac{m}{s^2}$ along $-z$, a friction coefficient of 1 and assuming uniform density. We compare our method against our FC baseline (which provides instance segmentation) in terms of object center displacement after simulation and report results in Table 5.1. Our method outperforms the fully convolutional baseline by over 50*cm* on both our test datasets indicating that the collections generated by our method are generally more stable and hence plausible. Note that in the case of rolling objects as well as intersecting objects (causing objects to be pushed apart due to contact force), average object displacement can become large.

Comparison to baselines We evaluate our baselines using the same procedure and display our quantitative results in Figures 5.24, 5.25 , 5.26 and Table 5.1. Note that we set S to 1 in 5.17 for our baselines and just use the prediction $\hat{y}$.

Evaluation of instance segmentation using a measure of stabiliy In addition to the main evaluation, a more intuitive evaluation of stability is proposed: in the case of bad predictions (unstable decompositions), rolling objects as well as intersecting objects, pushed apart by contact force can cause large object centre displacements. To dampen the effect of outliers, the percentage of stable piles according to a stability threshold, is estimated, determined through observation. To allow some shuffling of objects, but exclude falling objects, the following threshold is set: a stack is defined to be stable if none of its composing objects' centres move by more than 20cm and all objects' orientation change stays within 30°. The percentage of stable stacks by surface visibility on test data is displayed in Figure 5.27.

Qualitative results on synthetic data The qualitative results on synthetic displayed in Figures 5.28 demonstrate that our method generates realistic shape and instance segmentation, consistent with the visible regions, compared to the baselines which produce unrealistic reconstructions — they often miss objects or produce non-supported, flying objects.

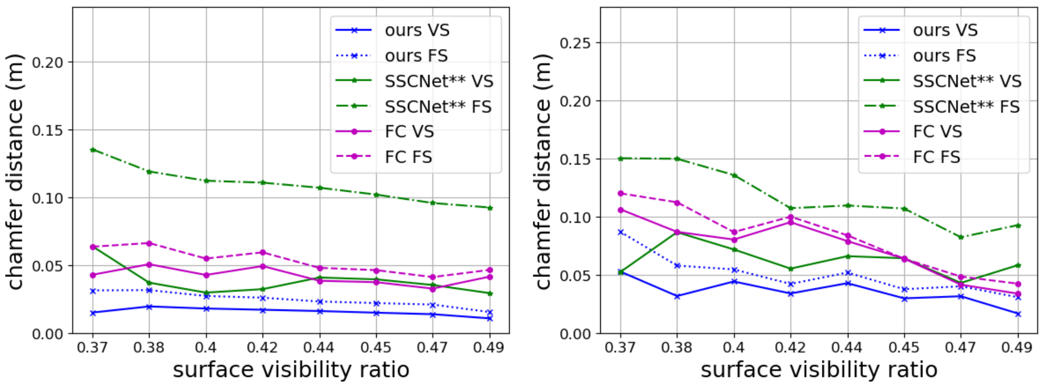


Figure 5.24: Chamfer distance for 1000 samples on our test dataset of superquadric shapes (left) and our test dataset of object stacks composed of YCB objects (right). We plot results for different surface mesh visibility ratios. **VS**: Visible surface. **FS**: Full surface

	ours		**FC**		**SSCNet****	
avg. Chamfer Distance (m) ↓						
	(μ)	(σ)	(μ)	(σ)	(μ)	(σ)
visible surface (SQ dataset)	**0.016**	0.0002	0.044	0.002	0.036	0.002
full surface (SQ dataset)	**0.028**	0.0003	0.062	0.107	0.03	0.002
visible surface (YCB dataset)	**0.036**	0.002	0.076	0.006	0.065	0.008
full surface (YCB dataset)	**0.062**	0.007	0.098	0.011	0.116	0.009
BCE of the expected occupancy ↓						
SQ dataset	**0.089**	0.003	0.175	0.005	0.247	0.008
YCB dataset	**0.112**	0.006	0.236	0.029	0.242	0.015
avg. object centre displacement (m) ↓						
SQ dataset	**0.593**	1.28	1.112	5.441	–	–
YCB dataset	**1.216**	4.734	1.756	6.494	–	–

Table 5.1: Comparing our C-VAE to the two baselines in terms of chamfer distance, BCE of expected occupancy and stability under physics simulation. We report the average results for 3 viewpoints per scene for our SQ test dataset and our YCB object test dataset.

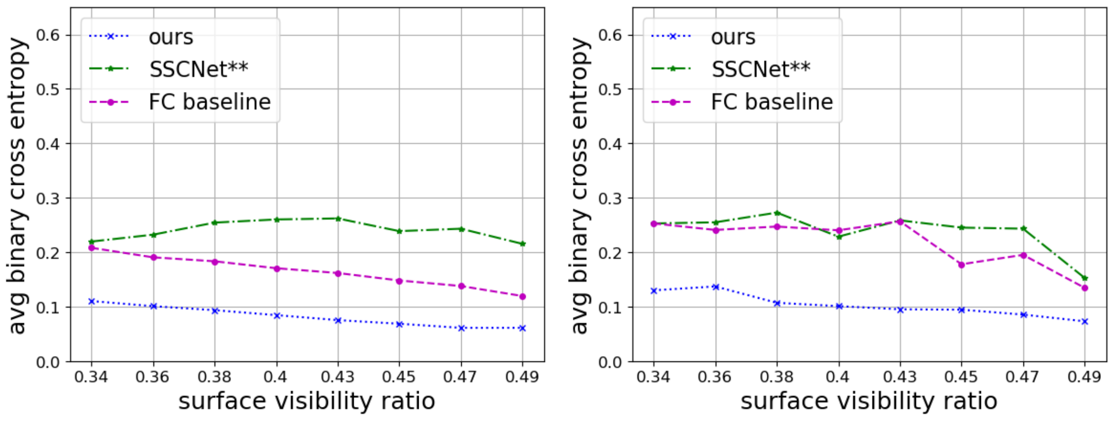


Figure 5.25: We compare our conditional VAE against *SSCNet*** and our fully convolutional (FC) baseline for predicted (expected) voxel occupancy. **Left**: test dataset of superquadric shapes **Right**: our YCB object test dataset.

Real data experiments

To qualitatively evaluate the proposed method on real data, we collect RGB-D images of collections of small to medium sized household objects (leveraging ORB-SLAM [Mur-Artal and Tardós, 2017] to obtain camera poses) and post-process them by 1) segmenting the floor plane using RANSAC [Rusu and Cousins, 2011] and 2) rectifying the roll and pitch of the camera pose by aligning the floor plane normal with the vertical axis. Examples can be found in our qualitative results (e.g., Figures 5.31 and 5.29).

Our results (Figure 5.31, 5.29 and 5.30) show that our method generates correct 3D reconstruction, realistic occlusion proposals and instance segmentation for a number of real world examples. Our example in Figure 5.30 shows that our method realistically fills in the occluded area below the box

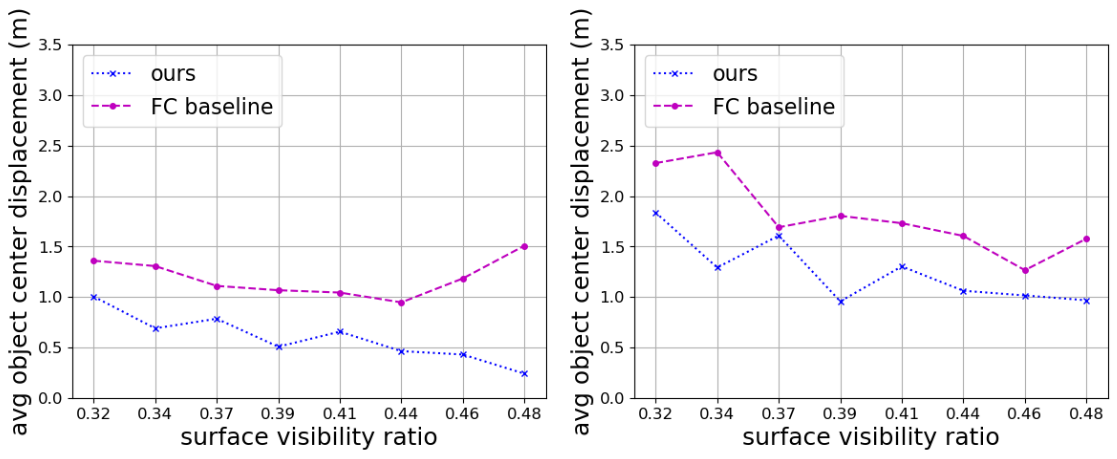


Figure 5.26: We compare SIMstack and our fully convolutional (FC) baseline in terms of stability under physics simulation (10000 steps) by computing the average object displacement per object stack (m). **Left**: test dataset of SuperQuadric shapes **Right**: our YCB object test dataset.

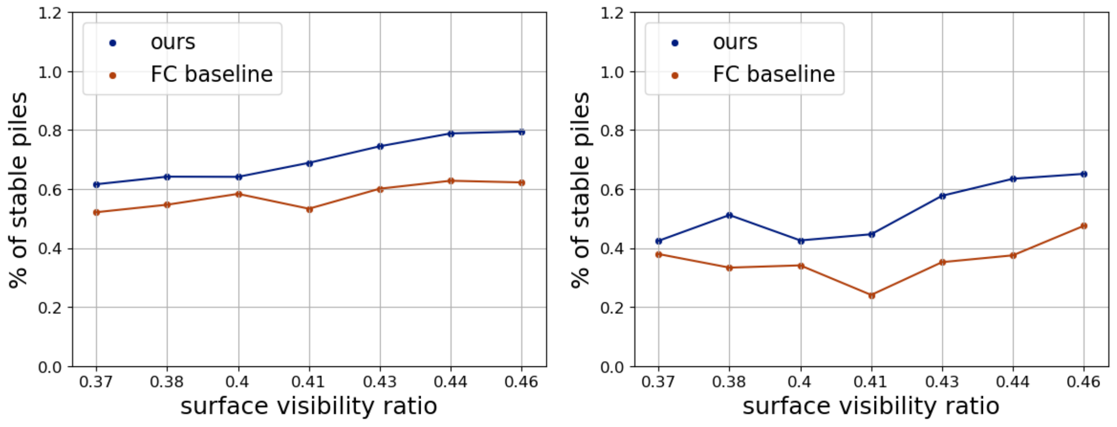


Figure 5.27: Percentage of stable piles generated according to our stability threshold (all objects center displacement stays within 20cm and all objects orientation change stays within 30°). **Left**: test dataset of SuperQuadric shapes **Right**: our YCB object test dataset.

by extending its shape to the floor. When sampling from the latent space, it is able to hypothesise different occluded objects. In particular, our example in Figure 5.29 (3rd row) shows that our network is capable to hypothetise a supporting object which is completely occluded, but required to support a leaning box. We also evaluate our method on YCB video sequences (Figure 5.29).

5.2.5 Discussion

Our method outperforms both baselines for full reconstruction and visible surface reconstruction, showing its overall advantage for reconstructing and decomposing a 3D scene. Note that the gap between visible and full surface reconstruction is largest for *SSCNet***, but more comparable between the FC baseline and our method. As expected for all methods, with lower visibility, full surface reconstruction accuracy drops faster than visible surface accuracy. The drop is however

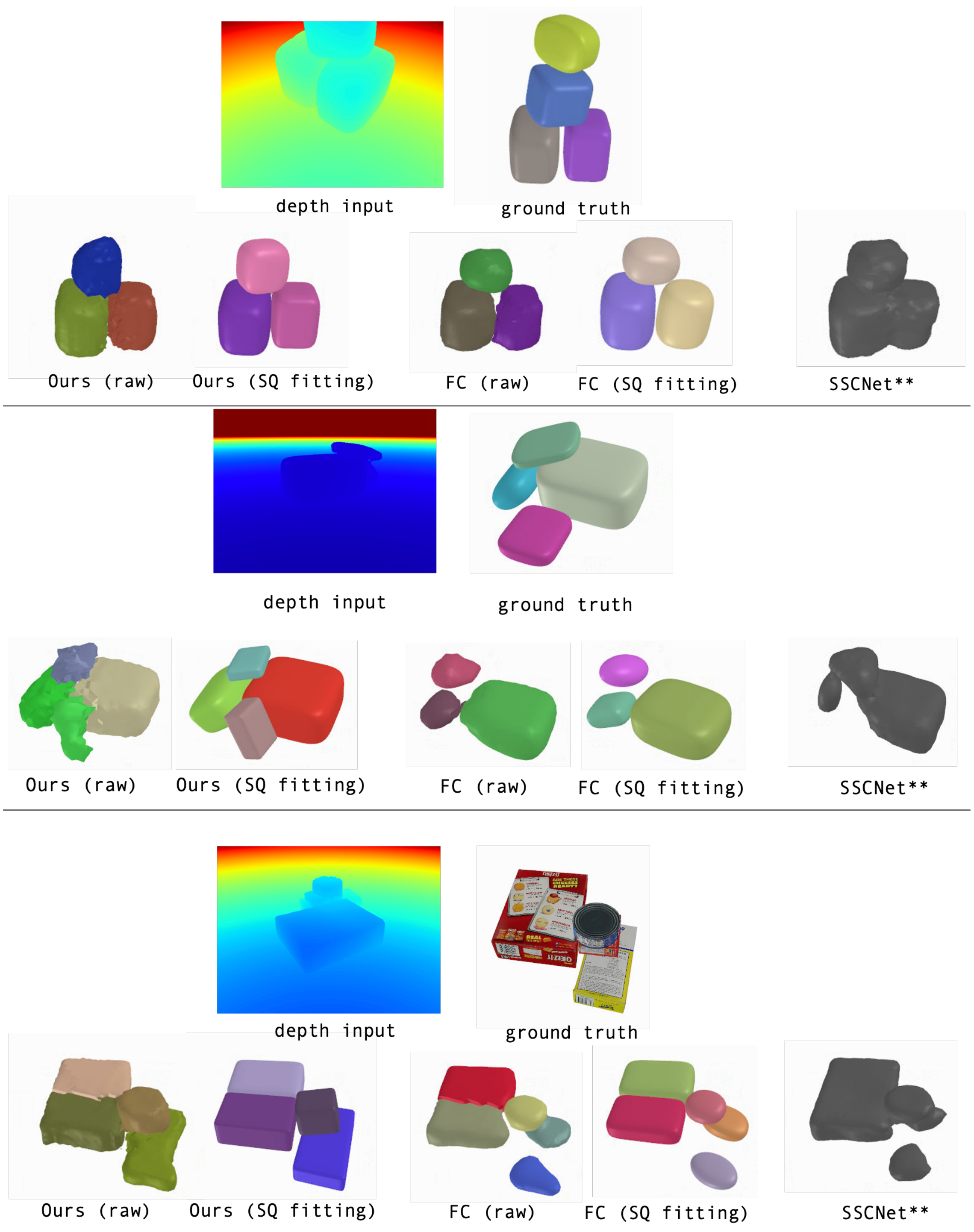


Figure 5.28: Qualitative comparison of our method against our fully convolutional baseline (FC) and our SSCNet baseline *SSCNet*** on examples from our SQ test dataset and our YCB test dataset.

most pronounced for *SSCNet***. These observations suggest that while using a prior improves overall performance, a large improvement in predicting occluded space is achieved by jointly predicting shape and instances. Perhaps, this is not surprising since our own reasoning about how space is shaped and how shapes can be completed in occluded regions is highly correlated with

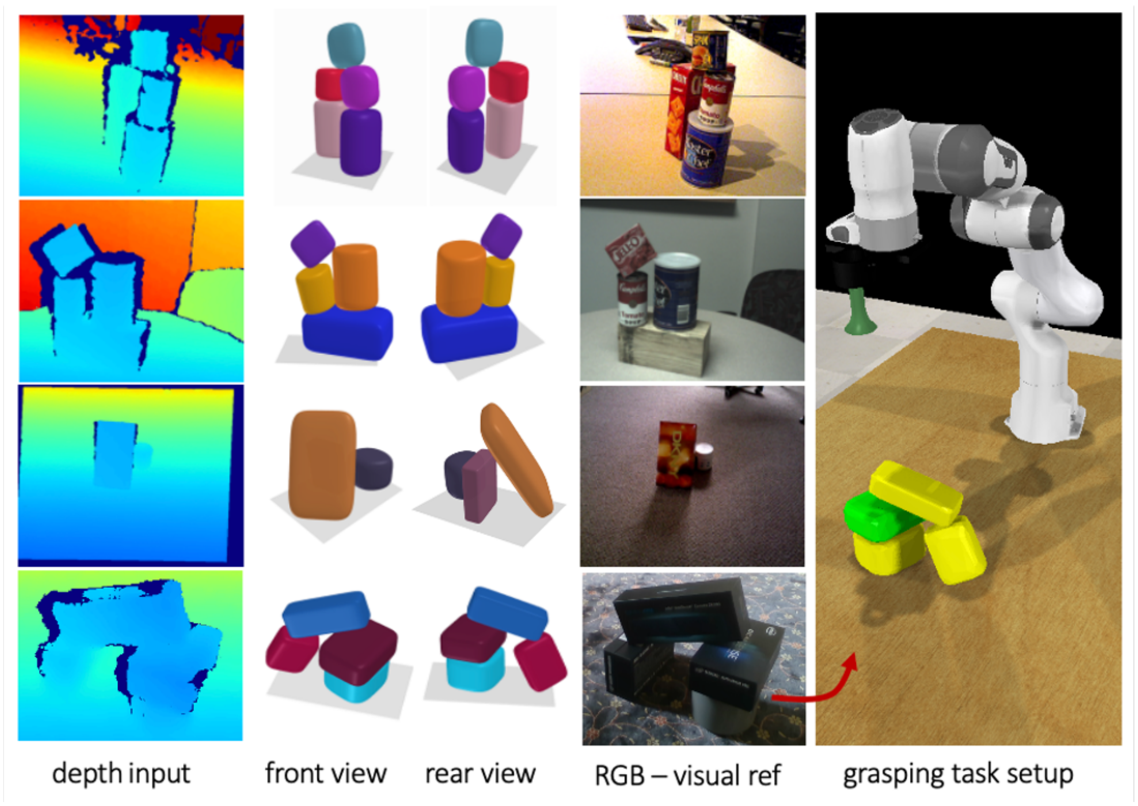


Figure 5.29: **Top to bottom**: SIMstack outputs object shapes and instances from a single depth view on two YCB sequences and two real data examples, one with a fully occluded object supporting a leaning box (3rd row), for which our model predicts a plausible proposition. **Right**: Grasping demo setup (green: target object).

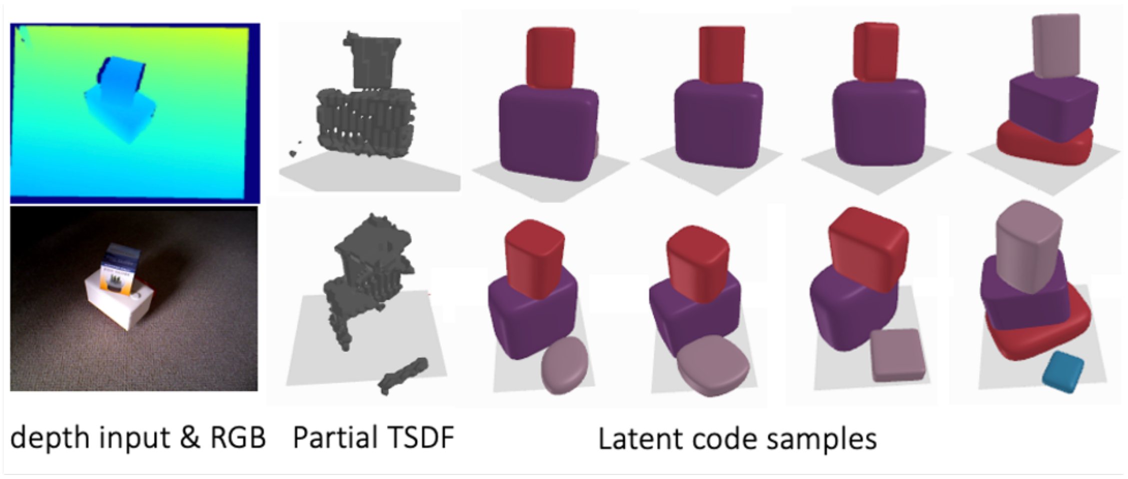


Figure 5.30: Sampling from the latent space for a real data example. We see multiple pile samples which all look similar from the point of view of the input depth observation (**top**) but significantly different though all plausible from the back (**bottom**).

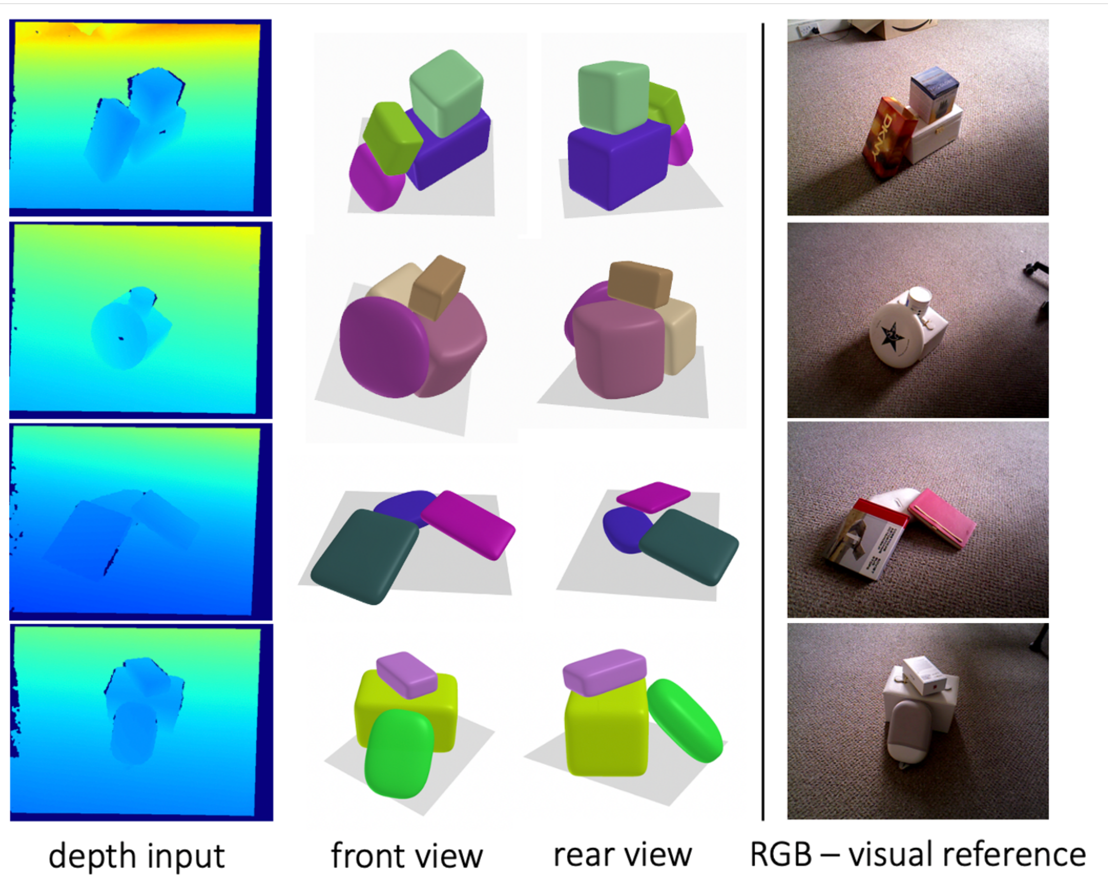


Figure 5.31: Qualitative results on real data examples. Our method generates 3D shape and instances from single depth views.

our knowledge about the decomposition of the scene into objects.

Our method is able to generalise to real-world data, even though it was only trained on synthetic scenes. Note that this sim-to-real transfer is largely due to the fact that we use depth inputs instead of RGB images. The difference between simulated and real depth is much smaller than that between simulated and real RGB images, which are highly susceptible to lighting variations and the properties of materials such as brilliance and specularity. Note however, that the system still requires the depth maps to be of a certain quality and that the observed objects are of similar scale compared to the objects seen at training time.

5.3 Integrating multiple views

Providing a shape and instance decomposition estimate for a scene from a single view is useful, but not very scalable in real-world settings if the initial estimate cannot be optimised from additional observations: Single-view predictions from different viewpoints would produce different estimates in their respective occluded regions and stand-alone, these estimates would have to be fused into a final reconstruction. To this end, the above method is extended to integrate multi-view reconstruction in two different ways: *multi-view conditioning* and *multi-view optimisation*.

5.3.1 Multi-view conditioning

The generative decoder can be conditioned on a partial TSDF generated from fusing multiple depth images. To implement multi-view conditioning, multiple depth views are backprojected into the scene using the inverse raytracing method described in Section 5.2.1. We update every voxel with the minimum distance as follows:

$$v_n = \min(w_{tril_n} * d_i, d_n) \tag{5.19}$$

where d_n is current depth at the voxel v_n and d_i is the depth obtained from the new measurement. The TSDF voxel grid is initialised to the truncation value before backprojecting.

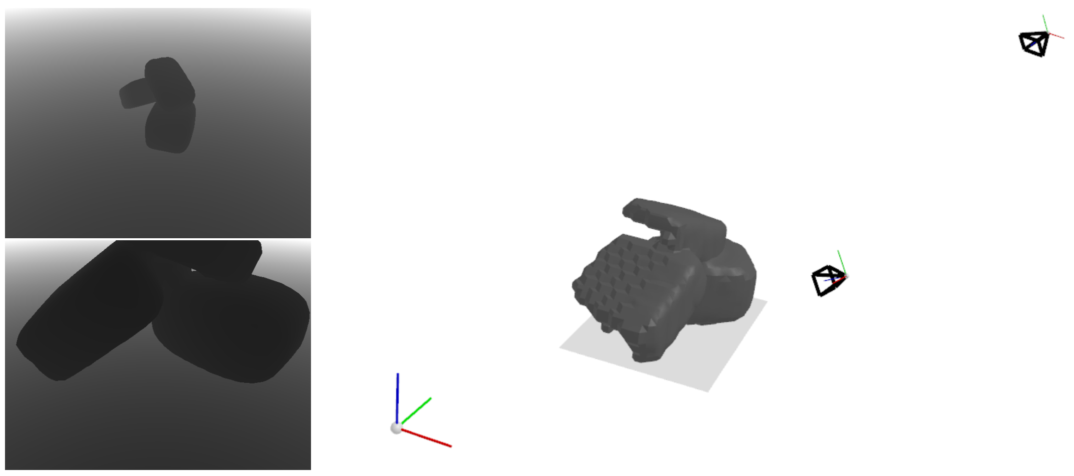


Figure 5.32: **Left**: 2 arbitrary depth images rendered from a test SQ scene. **Right**: the fused multi-view partial TSDF.

5.3.2 Multi-view optimisation

Our method encodes a 3D multi-object scene into a lower dimensional representation in form of a latent code. Given variational training, this latent code is actually a distribution over the entire dataset, with an approximately gaussian shape. This means it is smooth, as demonstrated in the interpolation experiments in Section 5.3.5. This property allows for optimization with respect to external information, e.g., novel views as previously demonstrated in [Sucar et al., 2020]. At test time, the model extracts features from one partial observation, which condition the generative decoder of the shape and instance VAE. A random latent code sample from the learnt prior then produces a full 3D shape and instance decomposition, given the conditioning depth view. While the conditioned decoder features are kept constant, locking the representation of the first depth view, the latent code can be optimized against additional views. To achieve this, we have to use differentiable rendering, which allows backpropagation of an error from the rendered view to the latent code (see Figure 5.33).

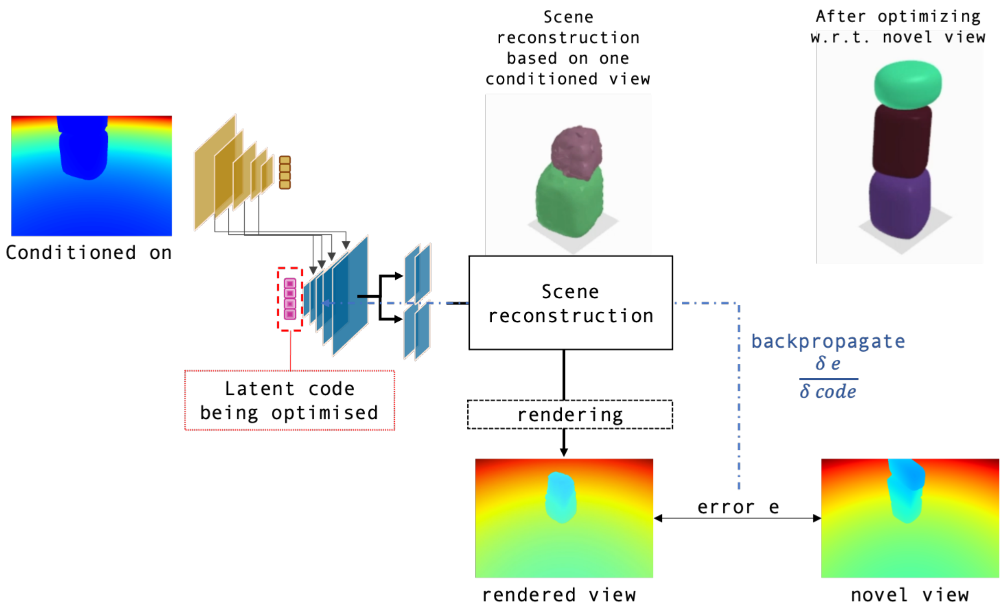

Scene reconstruction based on one conditioned view
After optimizing w.r.t. novel view
Conditioned on
Scene reconstruction
backpropagate
$\frac{\delta e}{\delta code}$
Latent code being optimised
rendering
error e
rendered view
novel view

Figure 5.33: Illustration of differentiable rendering to optimise the learnt scene prior w.r.t a novel view.

5.3.3 A differentiable renderer for raytracing a SDF

Our differentiable depth renderer is based on SDF raytracing. Similarly to [Jiang et al., 2020], we use sphere tracing to render depth. While stepping along each ray, we compute the exact SDF value using trilinear interpolation. [Jiang et al., 2020] only backpropagate the gradients into the immediate neighbourhood of each ray-surface intersection, which is correct when using a depth-image based loss. Since we use a more precise SDF based loss (Equation 5.20), we need gradients in the entire field of view of the camera. Although there has been recent work on fully differentiable sphere tracing [Liu et al., 2020], we choose a simpler approximation, sampling SDF values at regular intervals along every ray outside the surface to backpropagate gradients within the entire camera frustrum. We estimate our gradients using the binary loss described by Equation (5.21). Parallelised, our method can render the full cost image at an average runtime of $0.192s$ at a resolution of 640×480. In comparison, [Liu et al., 2020] render an image of 512×512 in $0.99s$.

Cost function

Although related approaches use a depth loss to optimise their latents [Jiang et al., 2020, Sucar et al., 2016], we observe that for a TSDF representation, using a depth-based loss can lead to discontinuities at occlusion boundaries. To this end, we design an SDF-based cost function composed of two parts: one describing the loss at the visible surface and one for the visible, unoccupied region of the scene.

Let π^{-1} be the function which backprojects a pixel $\mathbf{u_i}$ of the depth image into the 3D scene and I is the trilinear interpolation function which obtains the TSDF value at that point. The surface loss is:

$$L_{\text{surface}} = \sum_{\Omega} I(\pi^{-1}(\mathbf{u_i})). \tag{5.20}$$

The current TSDF is optimised towards alignment with this surface data, but this loss doesn't constrain on visible regions of empty space. We therefore define an empty space loss which penalises the code if it produces a negative TSDF value in observed empty space. We sample at regular intervals along rays in all regions of observed empty space and define the empty space loss, which has a 'space carving' effect: $L_{\text{empty\_space}} = \sum_s L_{\text{empty\_space},s}$, where:

$$L_{\text{empty\_space},s} = \begin{cases} |I(s)| & \text{if } I(s) < 0 \\ 0 & \text{if } I(s) > 0 \end{cases}. \tag{5.21}$$

Here, s is a sampled TSDF value. Our final cost function is the simple sum of both losses:

$$L = L_{\text{surface}} + L_{\text{empty\_space}}. \tag{5.22}$$

Figure 5.34 shows the cost volume for surface intersection areas as well as the cost volume after the additional sampling in the visible camera frustrum.

Optimisation (implementation details) We use first-order optimisation to optimise our latent code against additional depth images. Given the initial, conditioning view, additional views for

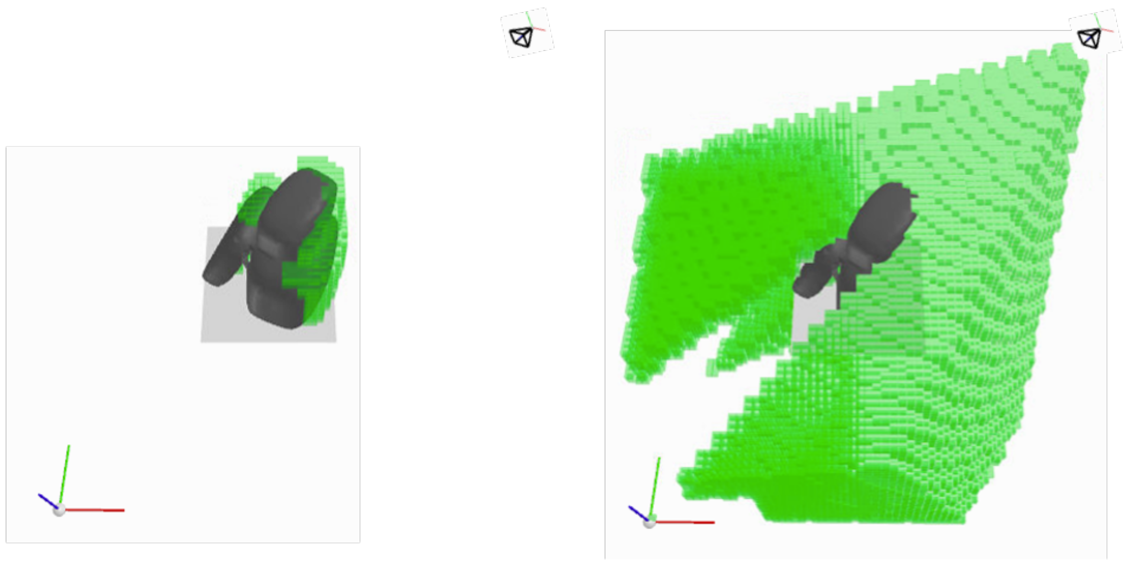


Figure 5.34: Visualisation of the cost volume generated by Equation 5.22. **Left:** the cost volume generated by considering the surface intersection areas of the backprojected depth. **Right:** the full cost volume obtained after sampling the visible area and applying equation 5.21

latent code optimisation can be selected from any new viewpoint. Once the loss L is computed, we backpropagate the gradients into the voxel-grid using inverse raytracing and trilinear interpolation, parallelised on the GPU. For multiple depth images, we accumulate the gradients of all views. We then leverage PyTorch autograd to backpropagate the gradients through the generative decoder of our conditional VAE and use the Adam optimiser to generate gradient updates.

Runtime Our multi-view conditioning method's runtime only depends on the time to generate a partial TSDF ($5.2s$ for 6 views using our TSDF fusion method) from multiple views as the forward pass time stays constant. It clearly outperforms our multi-view optimisation method which takes $21s$ and $75s$ to optimise against 1 and 6 views respectively, for 30 iterations.

5.3.4 Multi-view experiments and evaluation

We evaluate the performance of our method using multi-view conditioning and multi-view optimisation. We use simple TSDF Fusion as a baseline which we implement using the method described in 5.2.1. For a fixed sequence of 6 viewpoints that are selected manually (see Figure 5.35 for examples), we compare the reconstruction quality for the superquadric test dataset. For multi-view conditioning, we average reconstruction accuracy over 2 latent code samples. Note that the first view covers on average 30% of the test scenes. For multi-view optimization, we condition on the first view and optimise the latent code against the additional views. Our experiments show that multi-view conditioning generates the best reconstruction and that both our multi-view methods outperform TSDF fusion, even at very high visibility. We attribute this to the fact that our generative method also reconstructs the bottom of objects, which will always be occluded and therefore cannot be reconstructed by simple TSDF fusion. Our example in Figure 5.36 shows how a scene reconstruction is updated with additional views.

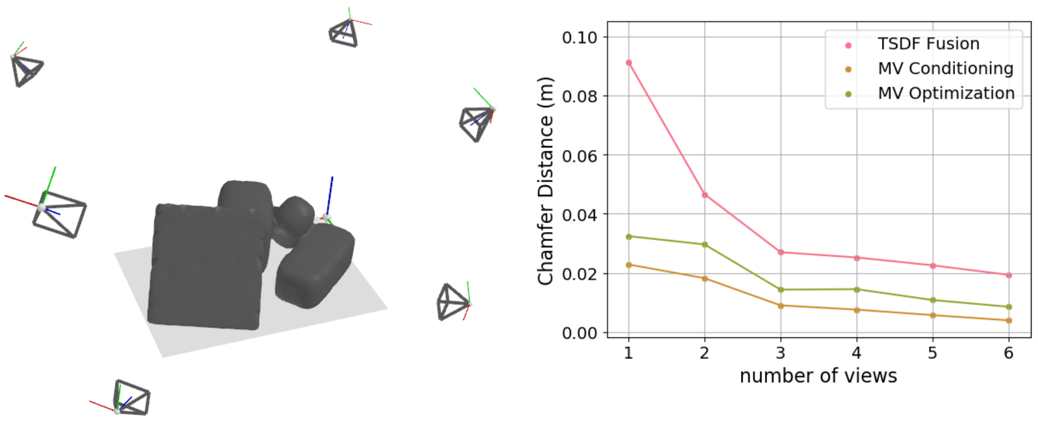


Figure 5.35: Multi-view estimation. **Left**: the 6 viewpoints for which we evaluate. **Right**: Chamfer Distance (1000 points) for increasing views of TSDF Fusion, multi-view conditioning (MV Conditioning) and multi-view optimisation (MV Optimisation).

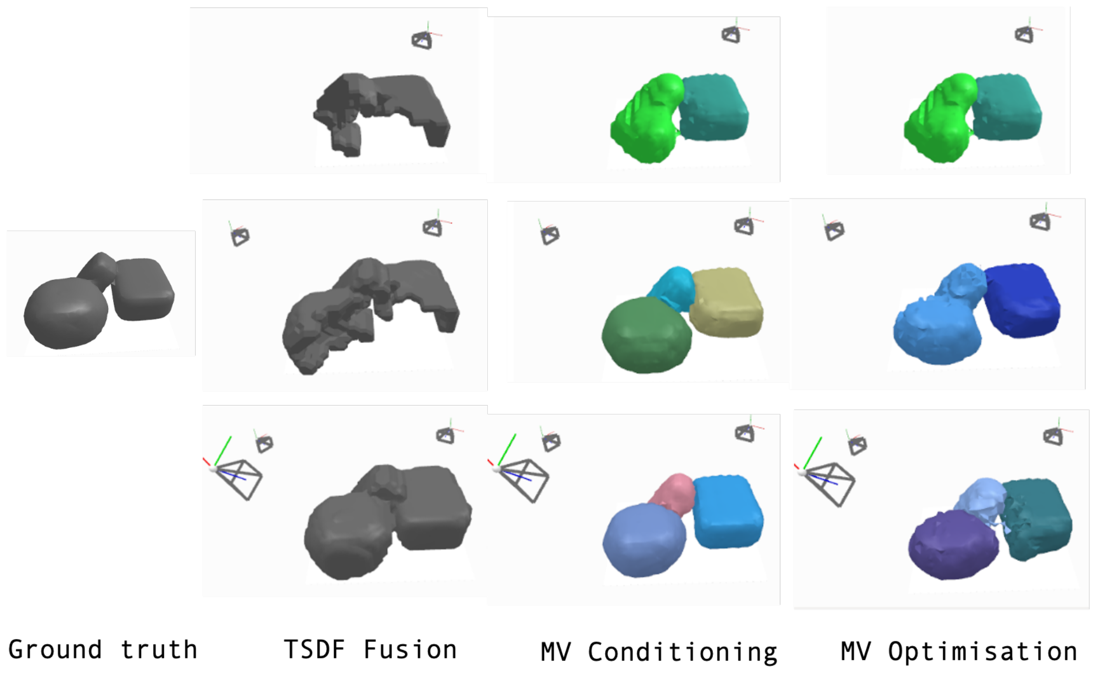


Figure 5.36: Visual example comparing the result of the multi-view Conditioning and multi-view optimisation methods with the output of simple TSDF fusion using the same viewpoints. **Rows 1-3** reconstruction (and instance segmentation) after 1 to 3 views respectively. Both multi-view conditioning and optimisation methods generate plausible shape reconstruction and instance segmentation in occluded regions.

5.3.5 Latent code analysis

In the following experiments, the learnt latent space (shape and instance prior) is analysed for sample quality and variety as well as smoothness and consistency.

Sampling from the latent space Given the design of our C-VAE, sampling from its latent space generates different proposals for occluded regions, while reconstruction of the visible surface stays constant. We show example of this on our Superquadrics test dataset in Figure 5.37. We show the mean scene (zero code scene) along with random latent code samples for a random viewpoint.

Sampling with multi-view information We condition our VAE on depth information to improve reconstruction in visible regions and to allow the latents to focus on occluded regions; sampling from those latents generates a variety of propositions for shape and instance segmentation of those hidden regions. Adding additional views increases the information about 3D space and should show a decreasing variety in latent space samples. We demonstrate this on an example from our Superquadrics test dataset in Figure 5.38.

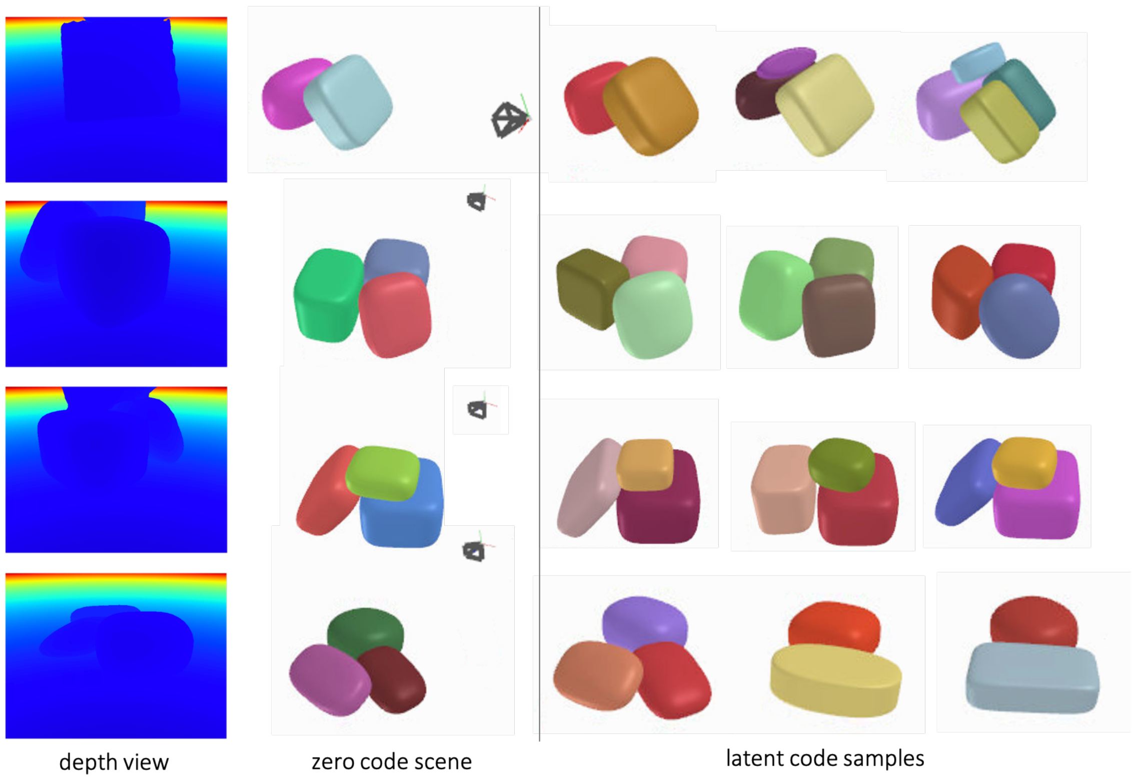


Figure 5.37: Sampling from the latent code of our conditional VAE. Given a single depth image of the superquadric test dataset (**left**), we generate the zero code scene and three latent code samples from the network.

Latent code interpolation We qualitatively evaluate the smoothness of our latent code by interpol-

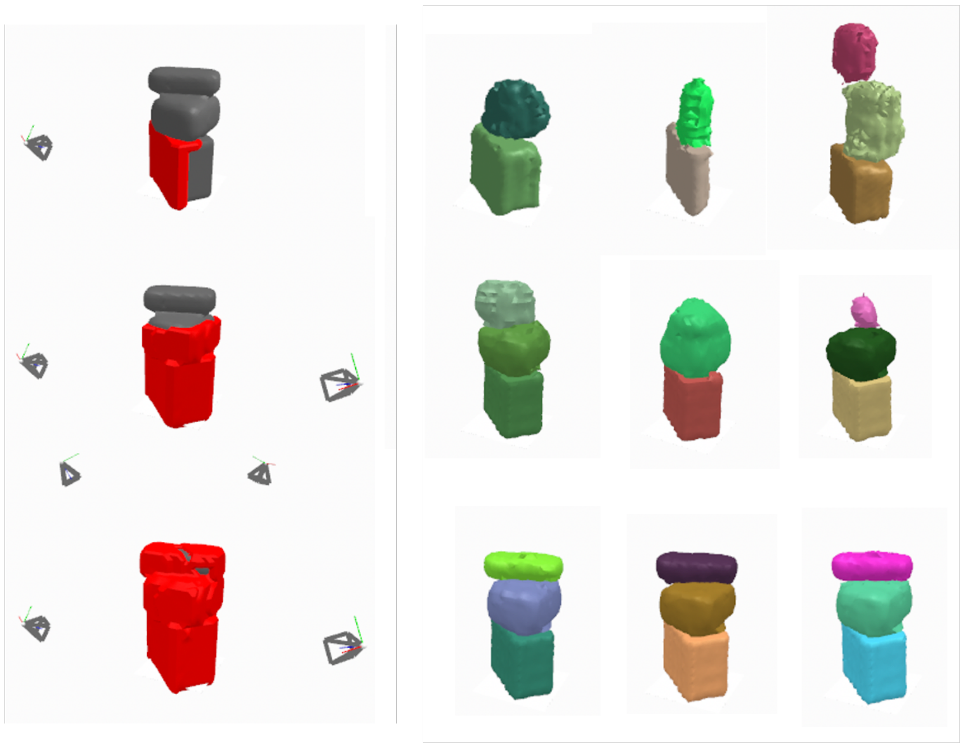


Figure 5.38: How sampling variety changes as views are added. **Left**: Ground truth and visible mesh area overlay (red). **Right**: 3 random latent space samples (without SQ fitting) for each view; as data is added the samples are increasingly constrained.

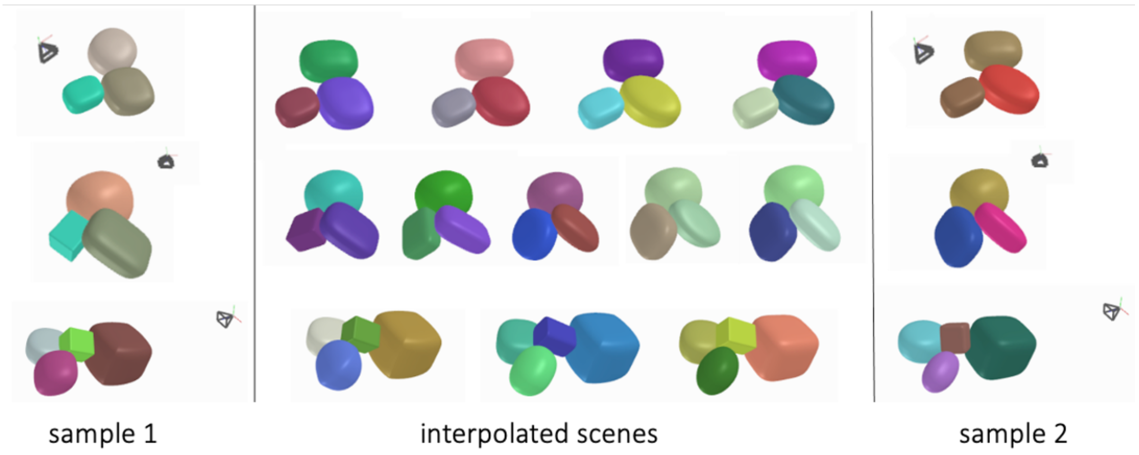


Figure 5.39: Latent code interpolations between two random latent code samples, conditioned on one view. We show three examples of interpolating between scenes with the *same number of instances* .

ating between random latent code samples of a depth-conditioned 3D reconstruction. Given a test scene and one viewpoint, we interpolate between two latent code samples to generate intermediary scenes. Our interpolations show a visibly smooth transition between scenes with the same number of instances (see Figure 5.39) as well as realistic intermediary scenes for interpolations between scenes of varying numbers of instances (see Figure 5.40): in the first example, the small object on the top left present in sample 1 becomes smaller, then merges into a long object which then shortens as interpolation approaches sample 2.

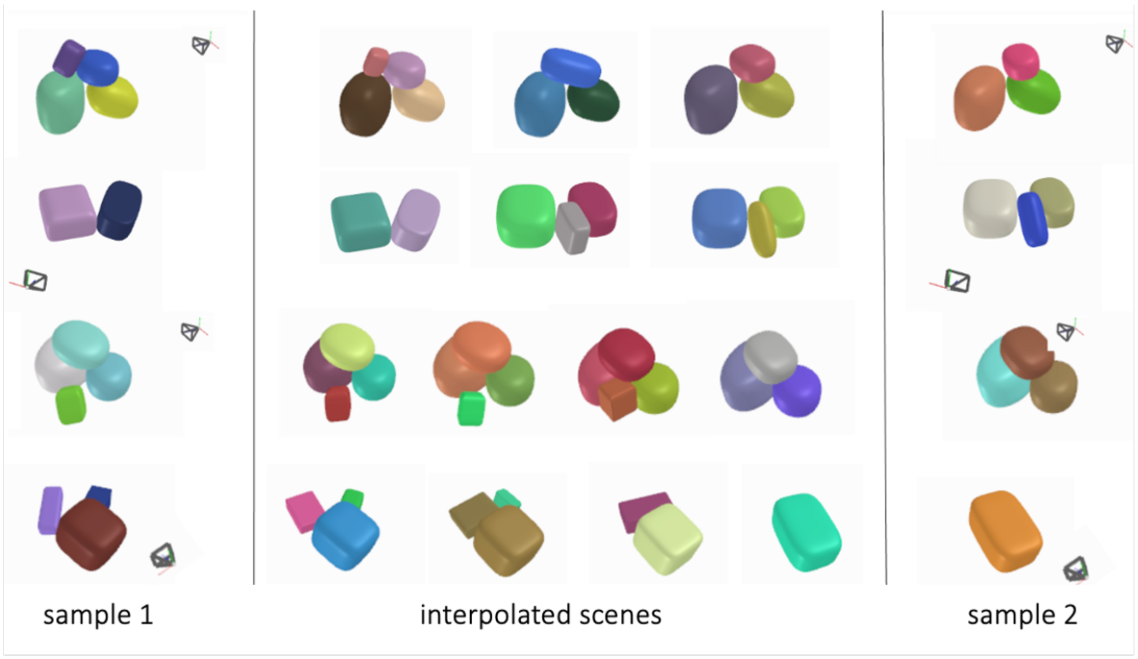


Figure 5.40: Latent code interpolations between two random latent code samples, conditioned on one view. We show four examples of interpolating between scenes of *varying numbers of instances*.

5.4 The pipeline: SIMstack

An overview of the full training pipeline of the SIMstack method can be seen in Figure 5.41. At test time our method can take as input one or more depth images and generates both the complete TSDF of the object stack and a center-voting vector field, which, processed by our 3D Hough Voting algorithm, provides instance segmentation. The output is then further refined by fitting a superquadric to each predicted mesh (overview in Figure 5.42). In practice, depth images obtained from a real camera have to be aligned to the floor plane, rectifying roll and pitch as was described in more detail in section 5.2.4.

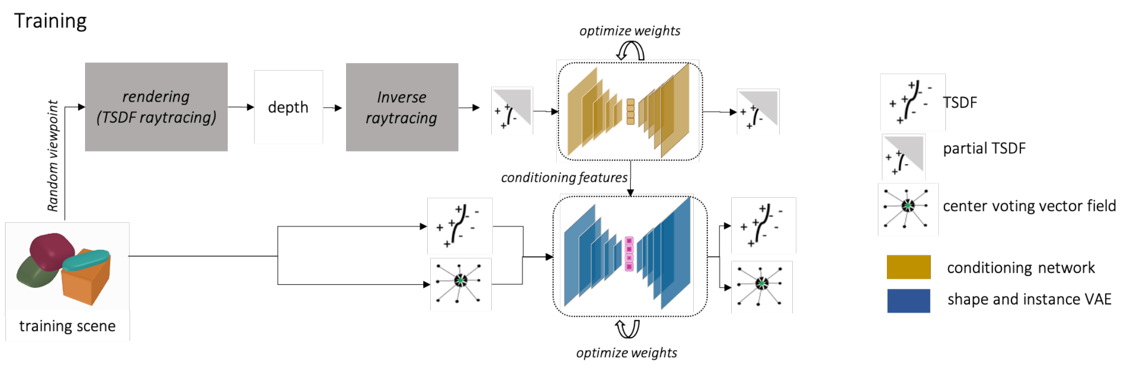


Figure 5.41: Overview of our method at train time. Networks are trained jointly. Pre-processing steps (grey) are not optimised.

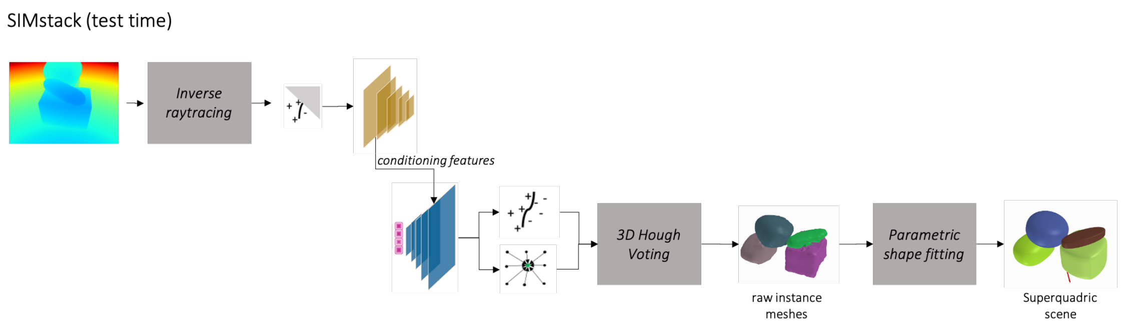


Figure 5.42: Overview of our method at test time for a single depth view input. Note that it is also possible to combine multiple depth images into a joint partial TSDF for multi-views scene prediction.

5.5 Application: precise robot manipulation

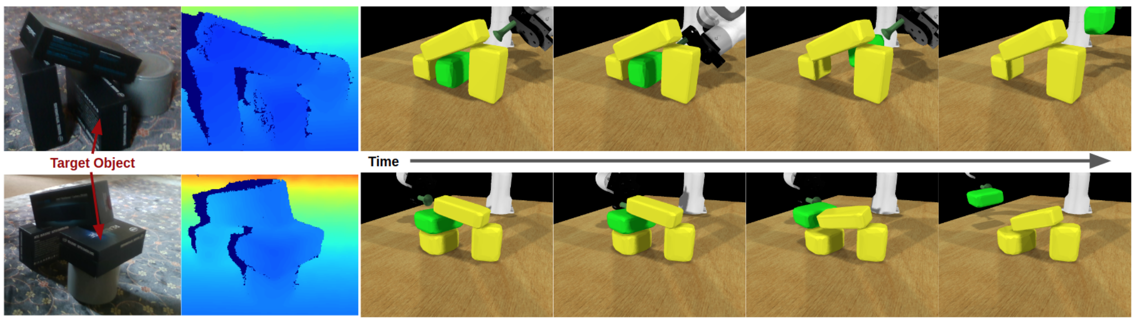


Figure 5.43: Two examples from our grasping system. Target objects are highlighted in green. The first example shows the robot sliding the target object sideways, leaving the other objects undisturbed. The second example grasps the object from the side and pulls it out such that the top resting object slides onto a lower supporting object, causing minimum disruption to the stack.

Robotic grasping is a well studied problem [Bohg et al., 2014] and an active area of research [Pinto and Gupta, 2016, Mahler et al., 2017, James et al., 2019b]. However, many state-of-the-art grasping systems perform indiscriminate grasping, with no regard of how the grasp might effect surrounding objects. However, multi-object reasoning is essential when grasping in cluttered scenes; grasping in an imprecise manner may topple a stack and cause damage to fragile objects. In this section, we show how SIMstack can be used to perform precise 6D grasping of a target object while minimising disruption to surrounding objects.

Our grasping demo consists of a real stack of (unknown) objects, a Franka Panda robot arm, and a suction gripper. An image of the scene is fed to SIMstack which outputs meshes and poses. These, along with a target object, are loaded into CoppeliaSim/PyRep [Rohmer et al., 2013, James et al., 2019a] where 5 virtual cameras are used to create a pointcloud of the visible surface of the stack. The target object's pointcloud is extracted, and grasping locations are sampled based on surface normals. We exhaustively simulate each valid grasp and measure the mean displacement of all objects (excluding the target object) after the grasp has been made. The grasp which produces the lowest mean displacement is chosen to run on the real platform. Note that due to COVID restrictions, we were unable to show the grasps running on the real robot. Two examples of successful grasps in simulations based on SIMstack reconstructions of real object piles are shown in Figure 5.43.

5.6 Performance on more complex scenes

Although only trained on 3-4 objects, SIMstack generalises well to scenes with up to 7 objects (see Figures 5.44 and 5.44). We attribute this to the fact that our formulation of instance segmentation is independent of the number of objects in the scene and that our model learns from scenes with varying number of objects. Our method also shows some ability to generalise to non-convex objects (see Figure 5.46).

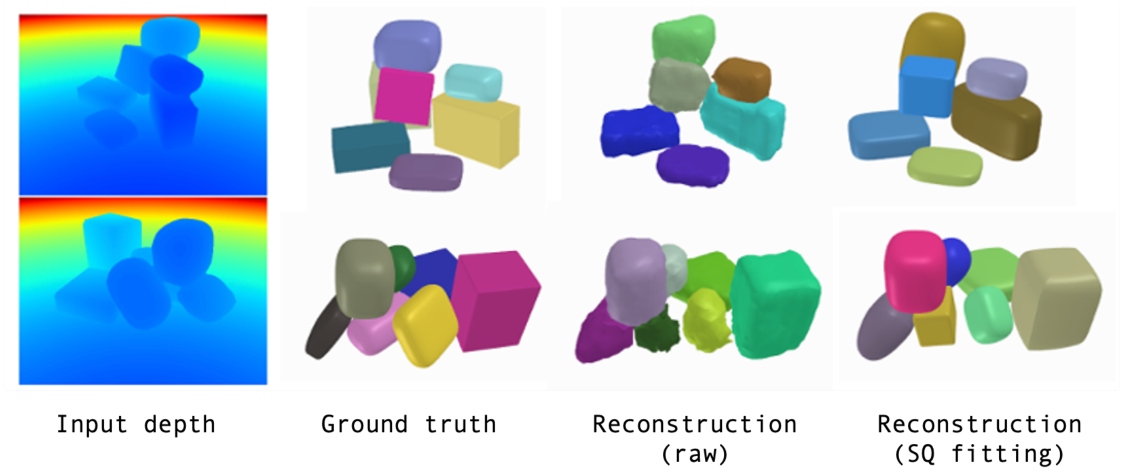


Figure 5.44: Qualitative results on scenes with 7 objects, demonstrating the ability of our method to generalise to more objects.

5.7 Conclusion

This chapter proposed a novel method to generate 3D shape and class-agnostic instance segmentation for multiple stacked objects from a single depth image. The method is based on the idea that a learnt scene-prior which encodes both the shape and instance segmentation of scenes can act as an intuitive physics prior for realistic object stacks, improving reconstruction and segmentation in occluded regions. We propose to learn the scene-prior using a VAE which is conditioned on depth features obtained from a random viewpoint. To test our hypothesis, we learn such a scene prior

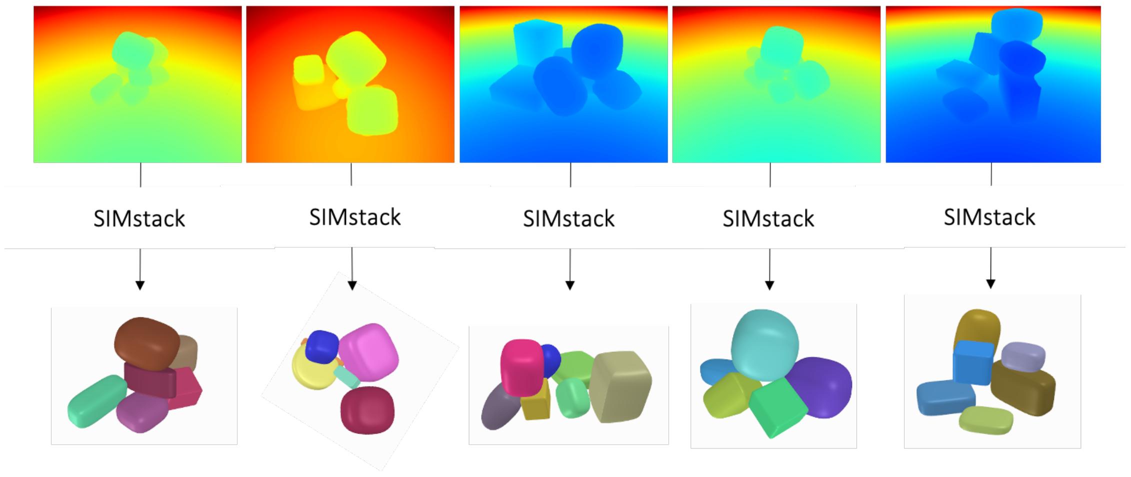


Figure 5.45: Qualitative results on scenes with 7 objects, demonstrating the ability of our method to generalise to more objects.

for a dataset of parametric shapes, randomly assembled under physics simulation and validate its performance on real data examples. The final system, termed *SIMstack*, is further tested on an application of non-disruptive grasping. Finally, we propose two methods for integrating additional view into the system to make it a candidate for incremental mapping systems.

SIMstack can generate a 3D shape and instance decomposition of a collection of (convex) objects from a single depth view. This output allows for a quick estimate of shape and instance decomposition, which could be used for downstream applications (e.g to initialise a multi-view scanning system to capture more detail [Wada et al., 2020]) and allows for fast multi-object reasoning, useful for interactive tasks such as the precise (non-disruptive) grasping we demonstrate. We believe our approach can play an important role in rapid scene understanding to help embodied AI systems make physically intuitive interpretations of ambiguous scenes.

One of the limitations of the system is over- and under-segmentation of objects which are in close proximity to one another and cannot easily be distinguished from a depth image alone. Leveraging the texture from RGB images will likely allow the system to more accurately segment ambiguous data. However, using raw RGB input makes the generalisation task harder, due to lighting variations and the properties of materials such as brilliance and specularity. Instead, one could leverage intrinsic image decomposition [Ma et al., 2017b] to obtain the reflectance image, which is free of any lighting dependent features. We believe this would be an interesting extension of our work. Another limitation of SIMstack is that it cannot represent non-convex shapes. Extending this work to non-convex objects would be another promising future research direction.

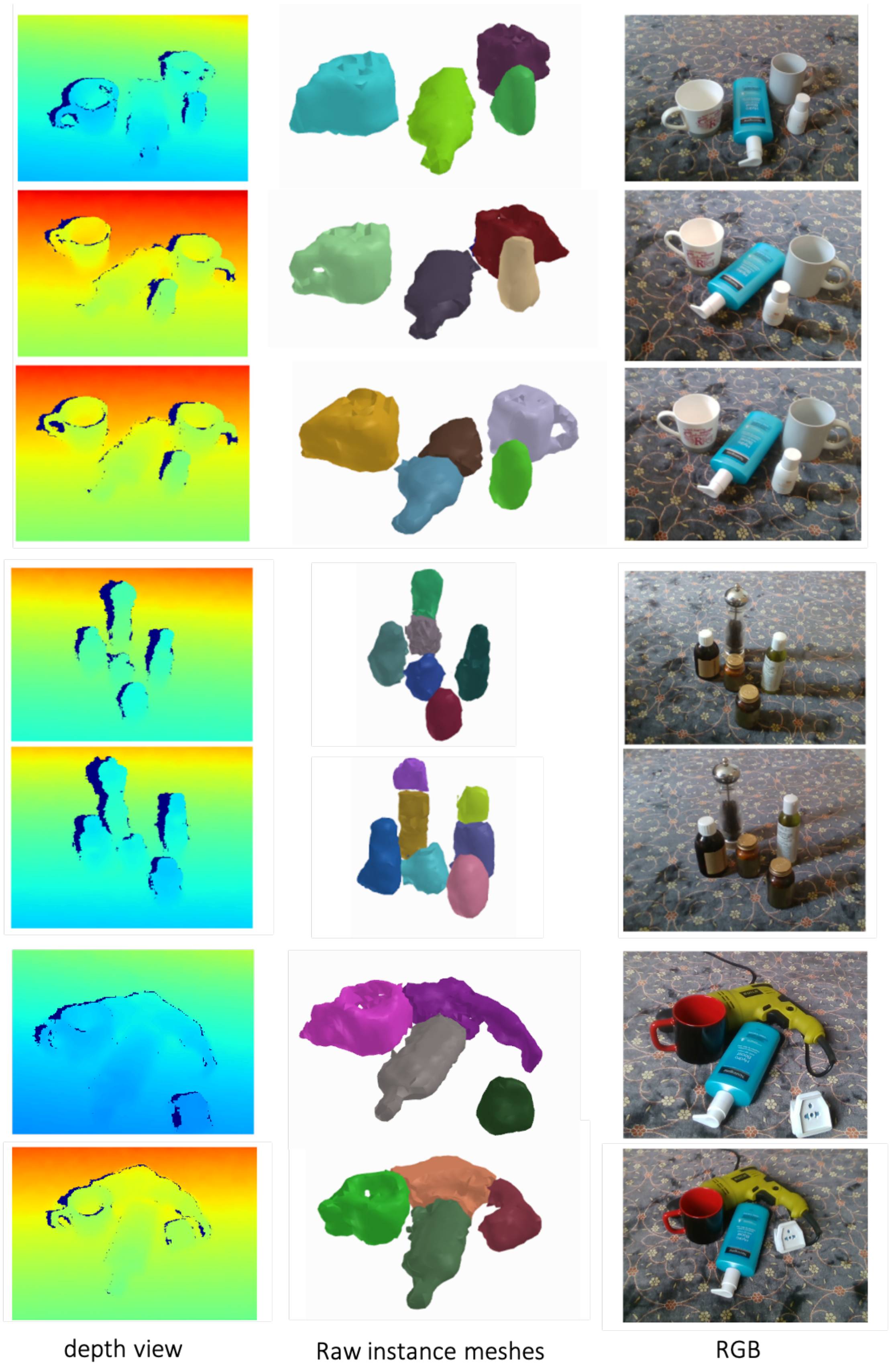


Figure 5.46: Qualitative results on real scenes with non-convex objects. The raw mesh segmentation of SIMstack is able to estimate shape and decomposition of scenes with non-convex objects.

CHAPTER 6

Decomposing multi-object scenes into factorised latent spaces

Contents

6.1 Introduction

The last chapters addressed the task of semantically annotating 3D scenes when a large amount of views is available and how to reason about missing data in terms of shape and instance segmentation when only one or few views are available. Both projects assume a global scene representation, explicitly, as a height field and implicitly, as a joint scene latent code. For the height map representation (Chapter 4), the segmentation is added externally to every scene element and one difficulty lies in accurately labelling the geometry, in particular, in border regions. Furthermore, explicitly representing geometry and annotations can become expensive for large scenes.

The global scene code of SIMstack (Chapter 5) offers a more compact representation and both the shape and instance segmentation of a full scene are jointly encoded. However, it is not possible to extract the encoded representation of an individual scene component and generate a separate reconstruction of it. For this, the scene encoding would have to be compositional, i.e. *factorised*.

This chapter is concerned with creating such a *factorised* representation of a scene - which we refer to as *object-centric* scene representation, a representation that consists of a collection of individual scene components, each separately encoded into their respective lower dimensional latent code. We focus on the particular setting of SIMstack (see Chapter 5), where a compositional scene representation will alleviate the pipeline of some of the post-processing methods and will likely make the system more scalable to larger collections of objects. Furthermore, a seperate latent representation per object could facilitate segmentation and reconstruction of non-convex object shapes, since object features would be disentangled and object-specific. Extending SIMstack to a system which can handle very large collections of non-convex objects would make it a powerful depth to 3D reconstruction and segmentation system that can be applied in many real-world settings with arbitrary shape types. Understanding how to generate an object-centric scene representation will also be of value for scene representations in general, making them more interpretable and facilitating scene manipulation and interactions such as object removal or addition for the composition of novel scenes.

6.1.1 Explicit, implicit and latent scene representations

How a scene is represented and stored in memory is a crucial aspect of scene understanding, as it governs both the memory requirements for the system using that representation, as well as the way downstream applications (e.g. path-planning or grasping algorithms) will interact with it. Most current state of the art SLAM systems use global scene representations such as a voxelgrid [Huang et al., 2021], surfel-based representations [Whelan et al., 2015] or global scene mesh representations [Bloesch et al., 2019]. In some cases, these representations are augmented with semantic or instance labels, and these external annotations can be used to segment respective areas of the global scene representation into its individual components. While a global representation has the advantage of simplicity and geometric consistency, having to decompose the full representation based on external annotations can have several drawbacks. First, both semantic and instance segmentation methods are far from accurate and labelling is often inconsistent and bleeds over object borders; this can lead to very noisy and inaccurate decompositions. Secondly, a top-down approach always needs to start from the full 3D representation to segment, which can be inefficient.

With the goal to obtain more compact and expressive scene representations, much research has been focussing on lower-dimensional representations of 2D and 3D space with deep latent variable models (e.g. autoencoders, variational autoencoders (VAE), generative adversarial networks (GAN)). In addition to providing a compact scene encoding, latent models are very effective feature extractors and are important for *representation learning*. Finding a lower dimensional encoding of a scene means being able to extract descriptive features which in turn is an important prerequisite for a model to make useful predictions. While some deep latent variable models such as autoencoders use their representations as a deterministic encoding, generative models such as VAEs and GANs learn a latent space which represents the underlying data distribution. The powerful latent representations learnt by such models have been used for advanced applications such as natural image generation [Karras et al., 2019, Park et al., 2019b, Wang et al., 2020a], image inpainting [Greff et al., 2019] or even lip motion transfer [K R et al., 2019]. Most works still focus on images and single object representations [Park et al., 2019a], however, some have attempted to learn latent representations for scenes [Kosiorek et al., 2021] and SLAM system components [Bloesch et al., 2018]. A recently emerged scene encoding method which is gaining popularity are neural scene representations, whereby an entire object or scene is encoded inside neural network weights [Mescheder et al., 2019, Park et al., 2019a]. These methods are often referred to as *implicit* representations and have proven effective for several applications such as novel-view synthesis and shape generation [Mildenhall et al., 2020, Chen and Zhang, 2019].

While encoded scene representations offer compactness, they generate highly correlated representations of all contents of a scene, a characteristic often referred to as *entanglement*. Entanglement reduces interpretability and accessibility of a representation, since individual features can't be easily viewed, extracted or manipulated. Furthermore, in many cases, detail is lost because all elements of a large scene are encoded into a very compressed representation. A whole line of work has been dedicated to the generation of *disentangled latent representations* [Watters et al., 2019, Higgins et al., 2017], whereby the encoding is generated in such a way that individual parts of the latent space are disentangled from others and are individually responsible for different features of an object or a scene. Such an disentangled representations can then be leveraged to change individual features of the scene such as colour, rotation or shape [Higgins et al., 2017].

The factorised, object-centric scene encoding we are aiming for is essentially a fully disentangled latent representation, whereby the features are clustered per object or scene component. A few approaches have attempted to obtain such compositional latent spaces using different methods; an overview is provided below.

6.1.2 Methods to generate factorised scene representations

2D image decomposition

The first methods that address object-based image decomposition with deep learning methods, approach the problem from a patch composition angle, decomposing into a set of bounding boxes, one for every object, and the background: [Eslami et al., 2016] propose a deep latent variable model *Attend, Infer, Repeat (AIR)* for unsupervised object discovery. They use an RNN as the encoder model, which at each step predicts and encoding of the shape and pose of one object. The final structure is a composition of per-object latent codes z^i, which are then decoded into patches which compose the original image. Interestingly, they find that using this structural bias of scene factorisation in their representation improves the generalisation capabilities of their model. [Crawford and Pineau, 2019] combine ideas from the AIR system [Eslami et al., 2016] and one-stage object detection methods such as YOLO [Redmon et al., 2015] into a more scalable unsupervised object detector. They replace the recurrent encoder network of AIR with a CNN that predicts a global feature volume. This feature volume is interpreted as object feature vectors in a grid (similarly to the YOLO detection pipeline - see Figure 3.12). Object representing feature vectors are filtered by predicted existence probability and are further refined into object descriptors through a joint MLP. Finally, they are individually decoded into their RGB appearance and used with a decoder to compose the final image. To reduce duplicate object latents, for every object descriptor prediction, neighbouring object descriptors are sampled and concatenated before refined by the MLP (a hand-crafted equivalent to attention-based mechanisms used in some current state of the art object detectors). Although these patch-based decomposition approaches show promising results, their bounding box representation of individual components lacks descriptiveness of the actual object shape.

Approaching the problem from a different angle, several researchers have developed pixel-wise gaussian mixture models of scenes, whereby every gaussian component k represents an image-sized scene component $x_k \in \mathbb{R}^{NxM}$. [Greff et al., 2017] propose *Neural Expectation Maximisation* (NEM), a scene representation modelled as a spatial mixture model, parameterised by a collection of neural networks which each learn a function that describes one individual object in the scene. [van Steenkiste et al., 2018] build on the work of [Greff et al., 2017] and extend their spatial mixture model with the ability to infer object interactions. They are motivated by the goal of physical modelling and argue that to be able to reason about objects and their interactions, a compositional representation of the world is necessary. With their Multi-Object Network

(MONet), [Burgess et al., 2019] are among the first to introduce the concept of *slots*, latent codes dedicated to represent an individual object in a scene of interest. Their method models scene decomposition using a recurrent attention-based neural network, which produces a set of masks that decompose the image into its components. Together with the original image, each mask is passed to a representation VAE, which learns a distribution over the masked image region and the mask itself and reconstructs a scene component (partial reconstruction of the image). The final output is modelled as a composition of all scene components. Intuitively, they argue that the task of image reconstruction will become easier when different parts of the learnt representation can be reused (e.g. one representation to model backgrounds) which motivates a compositional representation. (see [Burgess et al., 2019] for details). Instead of using auto-regressive inference, [Greff et al., 2019] propose IODINE, which uses an iterative variational inference scheme based on [Marino et al., 2018] to predict the posterior of a set of object-specific representations. Additionally, they use spatial broadcasting [Watters et al., 2019] in their generative decoder to further encourage disentanglement of each encoded entity z^i. With their GENESIS model, [Engelcke et al., 2020] add explicit modelling of object interactions using a graphical model to the object discovery task, which allows for the decomposition of scenes trained in an unsupervised manner as well as the generation of novel scene compositions. While their architecture is similar to that of MONet, the graph-based structure allows for sequential generation of scene elements, since individual components know about the existence of each other. Furthermore, GENESIS' encoders and decoders are run in parallel, which makes the model more scalable to scenes with many scene components. [Locatello et al., 2020] propose slot-attention, a modular component for scene decomposition, based on iterative attention. Their neural network module maps a feature representation obtained from a pre-processing step such as a CNN encoding to a set of slots, each representing the location and appearance of an object (similarly to the slot representation used in MONet and IODINE). While their results on unsupervised object discovery show similar reconstruction and segmentation accuracy to MONet, IODINE and GENESIS, they demonstrate advantages in computational and memory efficiency, due to their architecture, which uses iterations in latent space instead of iterative encoding steps.

Current methods have shown successful decompositions for simple 2D scenes with a few objects (see Figure 6.1). However, how to scale these methods for more complex scenes and in particular, for the 3D domain, is the topic of ongoing research.

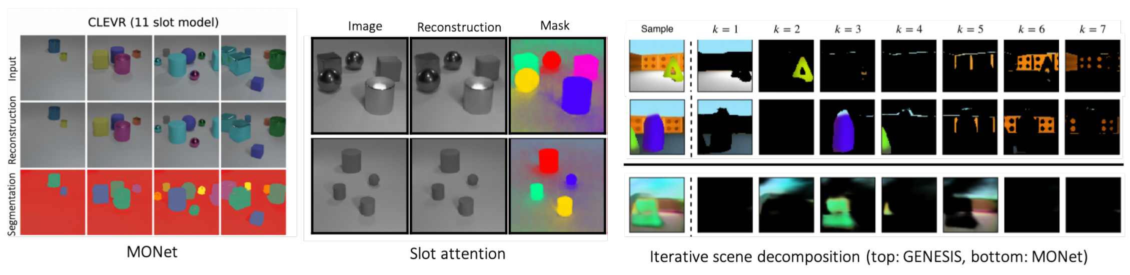


Figure 6.1: Visual decomposition results of state of the art methods on simple 2D scenes. **Left**: 2D scene decomposition results of MONet [Burgess et al., 2019] on scenes of the CLEVR dataset. **Centre**: decomposition results on intensity images of the CLEVR dataset by [Locatello et al., 2020]. **Right**: Iterative, sequential scene composition. The graph based model of GENESIS [Engelcke et al., 2020] allows for better sequential scene component generation.

3D scene decomposition

While several approaches address the problem of part-decomposition of 3D shapes using explicit representations such as polygon collections [Deng et al., 2020b, Chen et al., 2020b] or local implicit functions [Genova et al., 2020], fewer have attempted to generate a fully factorised 3D scene representation. The majority of these approaches leverage neural scene representations: [Ost et al., 2021] learn an object-centric graph-based scene representation where each object is encoded using a neural radiance field and the final image is generated using a composition of individual object representations. However, their method requires ground truth tracking information including object positions. With their work GIRAFFE, [Niemeyer and Geiger, 2021] present a method that achieves unsupervised scene decomposition into objects and background using a set of neural radiance fields. Each radiance field is conditioned on individual latent codes that, when combined, compose the final scene. However, their focus is image synthesis instead of inference. ObSuRF [Stelzner et al., 2021] combines attention-based set prediction [Locatello et al., 2020] with compositional NeRFs as in [Ost et al., 2021] and [Niemeyer and Geiger, 2021] and infer a 3D object-centric scene representation from a single image. The obtained NeRF-based representation can be leveraged to render novel views by composing the output of every individual NeRF, conditioned on one of the latent codes generated by the slot-attention encoder. They achieve impressive results, however, their experiments are limited to scenes with a maximum of 6 objects.

6.1.3 Motivation and approach

Although a few methods have explored compositional scene representations for 2D images and simple 3D scenes, the question of how to best generate such a representation remains unsolved.

In this chapter, we explore a novel method to generate an object-specific latent representation of a 3D scene. We focus on scenes with a set of small, cluttered objects and in particular, the goal is to augment the previously proposed method SIMstack with an object-centric scene representation to remove some of the post-pocessing steps. Although SIMstack produces consistent and stable reconstructions from a single view, the raw network output has to be processed using a Hough Voting algorithm that requires careful parameter tuning. With a factorised representation in latent space, the scene would be directly composed into its components and simplify the overall pipeline. Furthermore, a representation composed of a set of per-object latent codes will likely be able to better reconstruct more complex scenes, with more and potentially non-convex objects.

The majority of related approaches employ iterative or attention-based encoders to solve the set-prediction of the decomposition task at hand. We follow a simpler approach, similar to the work of [Crawford and Pineau, 2019], which selects a set of object codes within an extracted feature grid representation. Compared to [Crawford and Pineau, 2019], we don't use an existence variable, but extract latent codes from the feature volume, based on their average value (see Section 6.2.1 for the method). Similarly to [Marino et al., 2018], we use spatial broadcasting during the decoding of the latents to further promote disentaglement of the representation.

Most current methods approach the problem from the angle of unsupervised decomposition, which is motivated by the difficulty of obtaining labels for real-world data. However, unsupervised methods are usually difficult to design and train. As an alternative, supervised models can be trained in simulation and the learnt representation can later be *transferred* to the real data domain. While transfer learning is usually difficult for RGB data due to the high variety in appearance caused by lighting conditions and material properties, it is significantly easier for depth data, since the gap between simulated and real-world depth is not as large. Furthermore, our experiments in Chapter 5 showed that with the right setup and training, a depth-to-3D reconstruction method can generalise to real-world data after training on a large number of synthetic examples. We therefore explore how to obtain a scene decomposition in a supervised setting using synthetic data with the aim to be able to transfer the learnt representation to real-world settings.

6.2 Decomposing a scene into object-specific latent codes

Our goal is to explore a new method to obtain a 3D scene encoding which is factorised into its object components from a single depth view. Specifically, we want to extend the SIMstack architecture into a system which encodes a scene into multiple latent codes instead of one joint shape and instance latent code. In the following, we present a novel architecture to achieve such a decomposition.

6.2.1 Set prediction from a 3D scene

Extending SIMstack's architecture to encode a scene in a compositional way means integrating scene decomposition into the depth-conditioned VAE architecture. While the conditioning task remains the same, the main scene representation has to be modified to be a set of object specific latent codes instead of a joint shape and instance encoding. Specifically, the prior should encode a distribution over sets of object-specific latent codes. Each code holds the information necessary to represent one of the objects in the scene and is reconstructed back into the 3D shape of that object using the decoder. In the following section, a novel architecture is proposed to achieve this task.

Predicting a set of object codes has been attempted by previous methods, usually for a 2D view of that scene. Most previous approaches use recurrent, iterative or attention-based encoders [Eslami et al., 2016, Burgess et al., 2019, Marino et al., 2018] to generate a set of encodings from an image. We observe that set prediction has been solved using simple CNN encoders before [Wang et al., 2020b, Crawford and Pineau, 2019], albeit in the form of object locations and bounding box features. We suggest that with a carefully designed architecture, it should be possible to extract a set of factorised embeddings which each describe the entire shape of one object in 3D. To this end, we propose a novel architecture for set prediction from 3D scenes introduced below.

Experimental setup

Our experimental setup is similar to the one presented in the SIMstack pipeline, Chapter 5: a large amount of synthetic scene with collections of superquadric shapes is generated under physics simulation in the physics engine *Pybullet*. Compared to the setup in Chapter 5 we use an occupancy grid instead of an SDF scene representation. This choice is based on the observation that scene composition of a set of voxelgrids $\{v_0, \dots v_n\}$ as $S = v_0 + v_2 + v_3$ is well defined, while the equivalent composition using individual SDF representations is less trivial.

Our synthetic dataset is a set of scenes, each composed of a number of superquadrics and we experiment with scenes of a fixed number of 6 objects and scenes with varying number of objects (4-7 objects per scene) - see dataset examples in Figure 6.2.

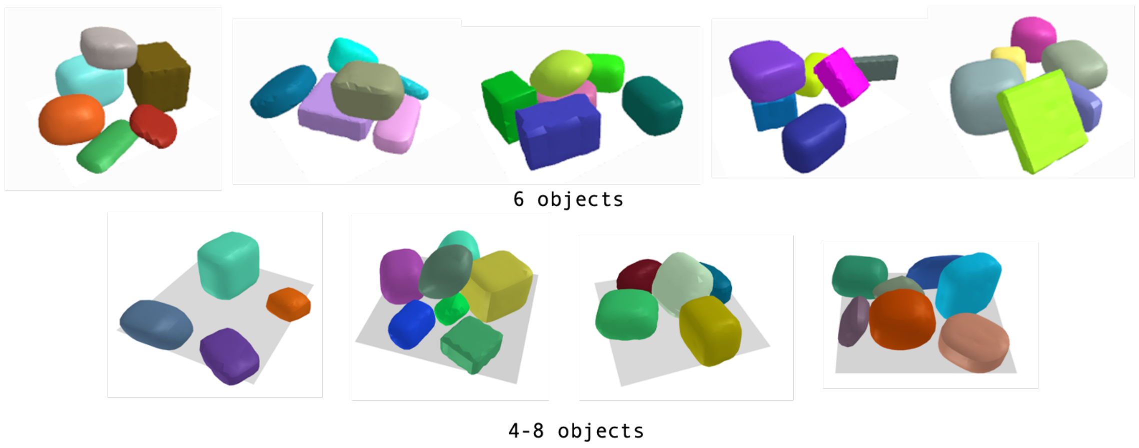


Figure 6.2: **Top:** dataset examples of scenes with a fixed number of 6 superquadrics. **Bottom:** dataset examples of scenes with a variable number of superquadrics $(4-8)$ objects.

An autoencoder for set prediction in 3D

Before attempting to generate a factorised latent space of a scene using the full depth-conditioned VAE architecture of SIMstack, we validate our proposed architecture as an autoencoder solving the set prediction in 3D and decomposing a global occupancy grid s into its individual object occupancy grids o_n: $f : s \in \mathbb{R}^3 \rightarrow \{o_1, o_2, \ldots o_N | o_n \in \mathbb{R}^3\}$. We propose the following novel architecture to decompose a 3D occupancy grid into its components.

Encoder A 3D CNN encodes a full 3D occupancy grid into a downsampled feature volume which is passed through a latent code extraction module to yield a number of N latent codes. These N latent codes are extracted as follows.

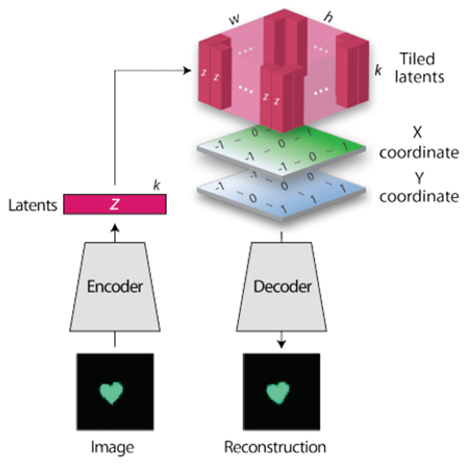
Inspired by object detection methods such as [Wang et al., 2020b], we encourage the network to predict object features at a set of locations in the downsampled grid: The feature volume is averaged along all channel dimensions and the set I of the indices of the N largest values are extracted:

$$I = \underset{A' \subset \mathbb{V}, A' = N}{\operatorname{argmax}} \sum_{a \in A'} a, \tag{6.1}$$

where $\mathbb{V}$ is the per-channel averaged feature volume and A' is any subset of size N of features in $\mathbb{F}$.

Then, the set I of indices is used to extract the corresponding set **f** of feature vectors (extracting along the channel dimension of the original feature volume $\mathbb{F}$). The resulting set **f** of feature vectors is the factorised scene encoding. Intuitively, this architecture design encourages the encoder to cluster all relevant information about individual objects at N locations, whereby each vector has the size of the channel dimension C of the feature volume $\mathbb{F}$.

Spatial Broadcasting To generate per-object feature volumes, we *broadcast* each extracted object feature vector into a feature volume, whereby $\mathbf{f}$ is duplicated along each dimension to yield and object-specific feature volume F_o: $f : \mathbf{f} \in \mathbb{R}^C \rightarrow F_o \in \mathbb{R}^{C,3}$. Broadcasting was first proposed by [Watters et al., 2019] as a method to disentangle latent spaces in VAEs. In the original implementation, the joint scene latent code obtained by the encoding is broadcasted along the spatial dimensions and concatenated with coordinate channels - one channel per dimension containing a coordinate grid (see Figure 6.3). This provides information about locality. The method was tested on 2D images and was shown to improve reconstruction quality and disentanglement of the latents, in particular for scenes with small objects. In our implementation, we follow the original



Algorithm 1 Spatial Broadcast

Input: latents $\mathbf{z} \in \mathcal{R}^k$, width w, height h

Output: tiled latents $\mathbf{z}_{sb} \in \mathcal{R}^{h \times w \times (k+2)}$

```
1: $\mathbf{z}_b = \text{TILE}(\mathbf{z}, (h, w, 1))$
2: $\mathbf{x} = \text{LINSPACE}(-1, 1, h)$
3: $\mathbf{y} = \text{LINSPACE}(-1, 1, w)$
4: $\mathbf{x}_b, \mathbf{y}_b = \text{MESHGRID}(\mathbf{x}, \mathbf{y})$
5: $\mathbf{z}_{sb} = \text{CONCAT}([\mathbf{z}_b, \mathbf{x}_b, \mathbf{y}_b], \text{axis} = -1)$
6: return $\mathbf{z}_{sb}$
```

Figure 6.3: Spatial broadcast decoder architecture proposed by [Watters et al., 2019]

design but extend it to a third dimension. Compared to [Watters et al., 2019], we broadcast every object-latent code into a separate object-specific feature volume which is decoded using a shared deconvolutional decoder.

Decoder Each object-specific feature volume F_o is decoded using a deconvolutional decoder, which brings F_o back to the original input size. The decoder weights are shared across object feature volumes. Intuitively, this design forces the network to encode all necessary information about the object appearance and location into F_o, while the decoder is optimised to map from that representation to the original resolution. Furthermore, given that the number and appearance of objects is variable and unordered, a network design with object-specific decoders would not be able to generalise.

An illustration of our architecture can be found in Figure 6.4.

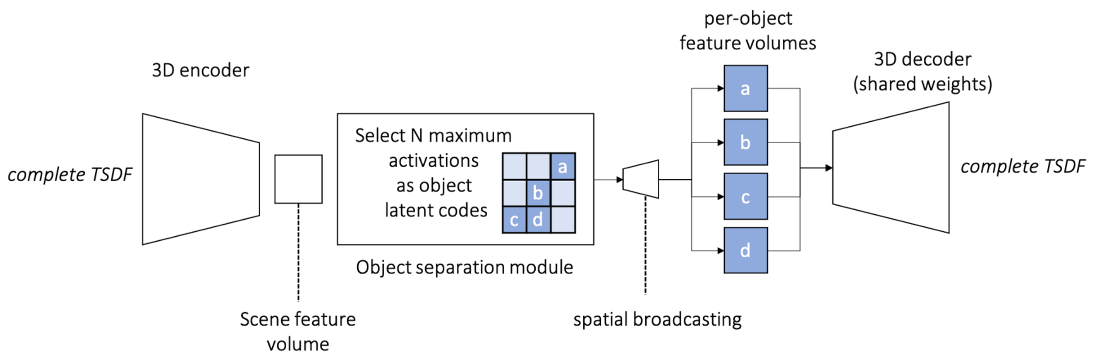

3D encoder
complete TSDF
Scene feature volume
Select N maximum activations as object latent codes
a
b
c
d
Object separation module
spatial broadcasting
per-object feature volumes
3D decoder (shared weights)
complete TSDF

Figure 6.4: The architecture of the scene factorising network. An input scene represented as a TSDF is encoded into a feature volume from which the N vectors with the largest mean value across the channel dimension are selected. Each vector is broadcasted into it's own feature volume and decoded into it's individual TSDF representation. Each reconstructed TSDF represents an individual object in the scene.

A loss function for set prediction

Any loss function applied to set prediction has to be invariant to permutations in the output, i.e. the loss should not enforce a particular order on the predicted set. In our case this means that the network output can't be compared to a pre-set list of individual ground truth objects, as this would force the network to use a particular latent code for each object, which would make generalisation impossible. Similarly to other approaches to set prediction with deep learning in literature [Carion et al., 2020], we decide to solve this by using Hungarian Matching, also called the Kuhn-Munkres algorithm [Kuhn, 1955], which finds maximum-weight matches in bipartite graphs (see Figure 6.5). This is sometimes referred to as solving the *assignment problem*. Matches are found based on a specific metric and in the case of our prediction task this metric is the similarity between predicted shape and ground truth shape for any particular object. Given that we are predicting occupancy grids, we decide to measure this similarity using the Binary Cross Entropy. After matching, the reconstruction loss is computed between each element. Intuitively, the Hungarian Algorithm will find those object assignments which maximise the Binary Cross Entropy between all prediction and ground truth object pairs.

Implementation details To use Hungarian Matching as a loss function the computed gradients have to bypass the matching operation, as it is non-differentiable. In Pytorch, this can be achieved by computing the matching inside a function decorated with a *nograd* keyword, which tells the framework to remove the involved variables from the gradient computation graph. Specifically, the output vector of individual object reconstructions is rearranged according to indices computed by the Hungarian Matching algorithm (matching prediction with ground truth) before it is compared to the ground truth objects yet again in a normal loss computation.

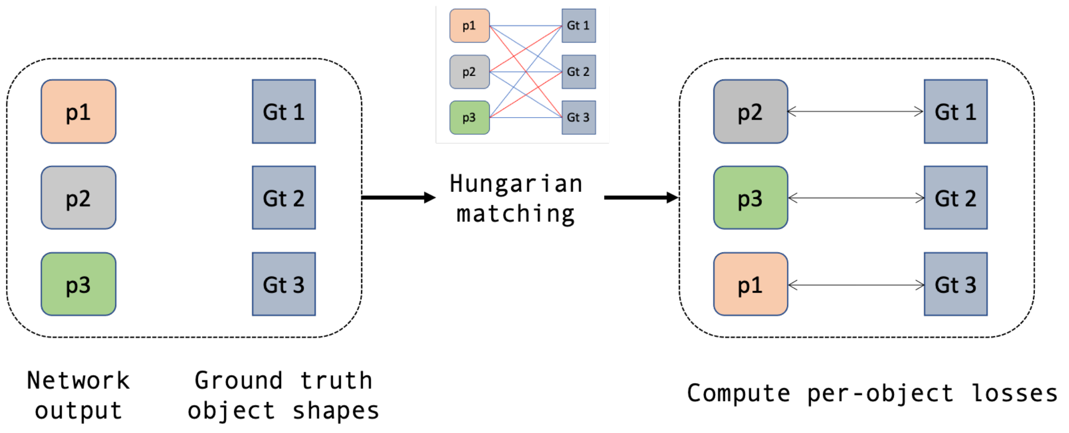


Figure 6.5: Hungarian matching applied to our loss function. The raw network outputs are matching with the ground truth objects and re-ordered according to highest overlap. Then, the loss is computed and backpropagated to update the network weights.

Experiments

We validate our proposed architecture by learning to decompose the two synthetic datasets of scenes with a fixed number of 6 objects and a varying number of $4-8$ objects. In these initial experiments, the input to the Autoencoder is the full 3D occupancy grid and the network is trained to predict the individual per-object occupancy grids as depicted in Figure 6.4.

Dataset and training details For the task of decomposing a scene of 6 objects, we train on a synthetic dataset of 8062 scenes; to learn how to decompose variably-sized scenes of 4-8 objects we train on 10765 scenes. Both tasks are optimised with an initial learning rate of $5e^{-4}$, which is adjusted using step-wise learning rate decay (step size of 10 and $\gamma = 0.999$). We use the Adam optimiser [Kingma and Ba, 2015b] for all experiments.

Architecture details For this task we set the latent code size to 128 and downsample the 3D input scene of resolution 64 to a global feature volume of resolution 4. The overall network has just over $2m$ trainable parameters.

Results As can be seen from Figures 6.6 and 6.7, the network is able to learn to decompose scenes of fixed and variable number of objects well. In the failure cases, object borders are badly segmented or in some cases, two objects are merged or split. This is not surprising, in particular in cases where objects are in contact and not easily distinguishable. Our quantitative results presented in Table 6.1 show that for both datasets only few scenes are predicted with missing or joined objects. However, for the dataset with varying number of objects, over 30% of scenes are decomposed into more objects. In these cases, either one object is split into multiple objects or the latent space is decomposed into duplicate representations — two or more latent vectors reconstruct into the same object. We comment more on this result in the discussion in Section 6.3.

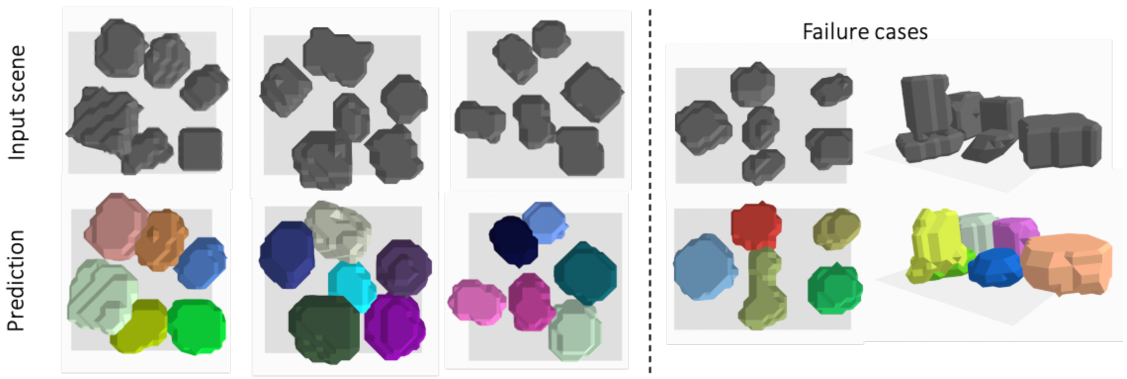


Figure 6.6: Results on predicting scene decomposition for occupancy grids with 6 objects. On the **right** are examples of failure cases

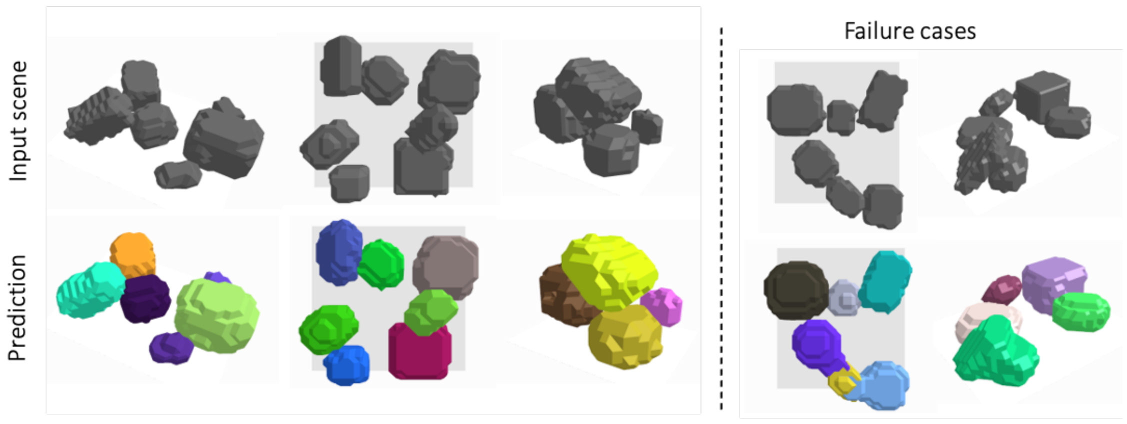


Figure 6.7: Results on predicting scene decomposition for occupancy grids with $4-8$ objects. The examples on the left show successful scene decomposition; on the **right** are examples of failure cases: one with bad object segmentation in border regions and one with a set of joined objects.

Latent space structure We verify that our object feature extraction method works, by visualising the feature volume and the extracted per-object latent codes. Figure 6.8 plots a subset of non-object codes against object codes. While object code values stay close to zero, non-object code values vary largely between positive and negative values, overall resulting in lower average values compared to object codes (Figure 6.8, right). However, a surprising result is that the gap between object and non-object codes is not very large, which suggests that mistakes such as missing objects could be a likely scenario. Although we do not often observe missing objects in our results, we expect that motivating greater separation between object and non-object codes in latent space will likely improve prediction quality.

We notice that instead of using the centre location of each object as a latent code location, the model learns to pick N random, but fixed locations within the feature volume into which it places the feature vectors of the detected objects. These locations appear to be independent of the input scene (see Figure 6.9). This shows that the network does not implicitly learn object location as part of the decomposition task. An open question remains whether external supervision of object centres would improve the decomposition. We leave this exploration for future work.

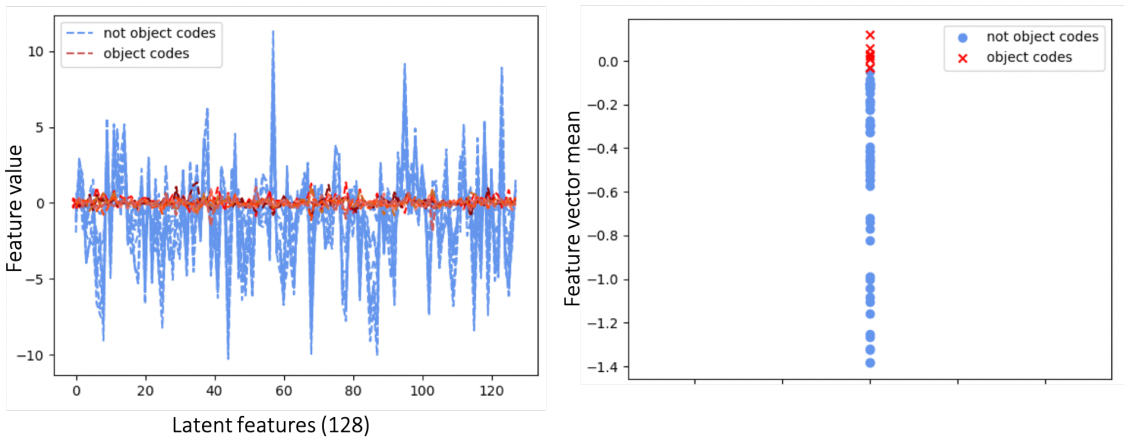


Figure 6.8: **Left**: Object-latent features (red) and a randomly sampled subset of empty latent features (blue). Empty latent features vary largely over all 128 values and predominantly into the negative region, while object representing latent features stay in the positive region and show much less variation. **Right**: The feature vector mean (over all 128 values) for a subset of empty latent vectors (blue) and the object representing latent vectors (red).

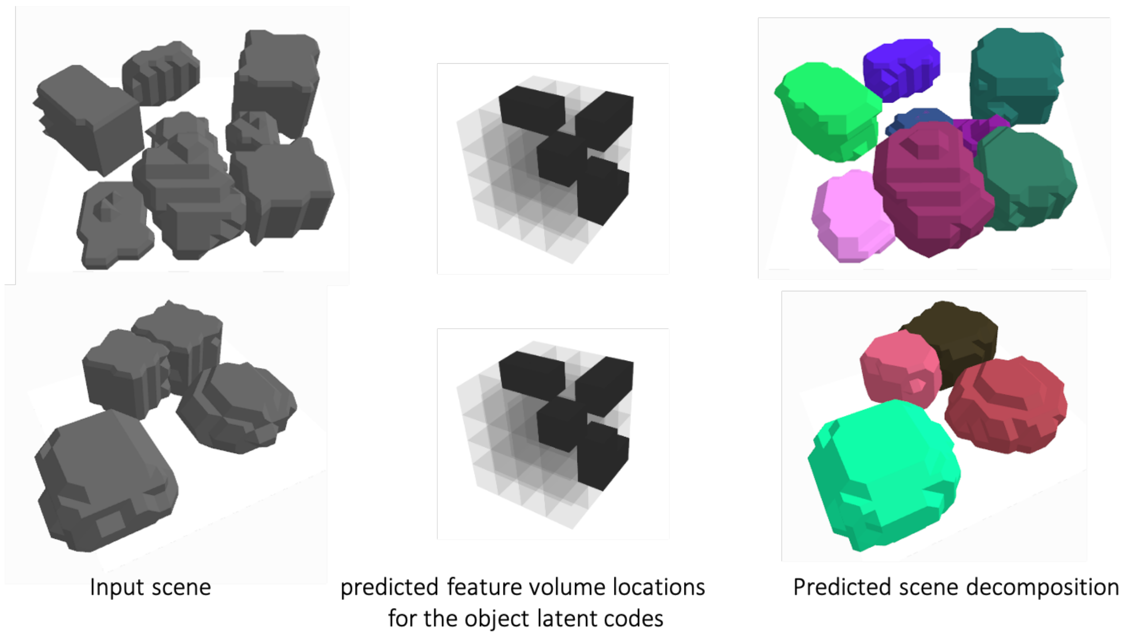

Input scene
predicted feature volume locations for the object latent codes
Predicted scene decomposition

Figure 6.9: Object representations producing different scene decompositions (**right**) are generated in the same locations (visualised in black) in the extracted feature grid (**centre**) of the input scene (**left**).

6.2.2 Scene decomposition from a single view

Having validated the proposed decomposition architecture using a full 3D input, we test how well a network can learn to complete the scene while decomposing into its individual objects from a

Dataset	Dataset size	Avg. BCE Loss	Missing/joined objects	Duplicated/split objects
6 objects	82	0.0014	2.4 %	–
4-8 objects	95	0.0021	5.2 %	34.5 %

Table 6.1: Quantitative results for the 3D set prediction task. Average Binary Cross Entropy is reported, as well as the percentage of scenes with missing or joined objects and those with duplicate or split objects.

single depth view. We follow the reasoning previously elaborated in Section 5.2.1, and transform the depth input into a partial TSDF to alleviate the network from having to learn a re-projection task.

Experimental setup We choose a similar experimental setup to that presented in Chapter 5. For every training scene, a random view is generated from a range of $-1m$ to $1m$ in the x and y dimensions, and a height range of $33cm$ to $83cm$. All views are selected with a minimum distance of $50cm$ from the scene centre. The depth image is generated using TSDF raytracing as described in Section 5.2.1; after rendering the depth, a partial TSDF is generated using the inverse rendering method presented in Section 5.2.1. An example of a partial TSDF generated from a training scene is displayed in Figure 6.10.

Network Architecture Since the depth view is transformed into a partial TSDF, the network architecture remains unchanged, the only difference being, that it now has to learn a function mapping from a partial TSDF to a set of per-object occupancy grids.

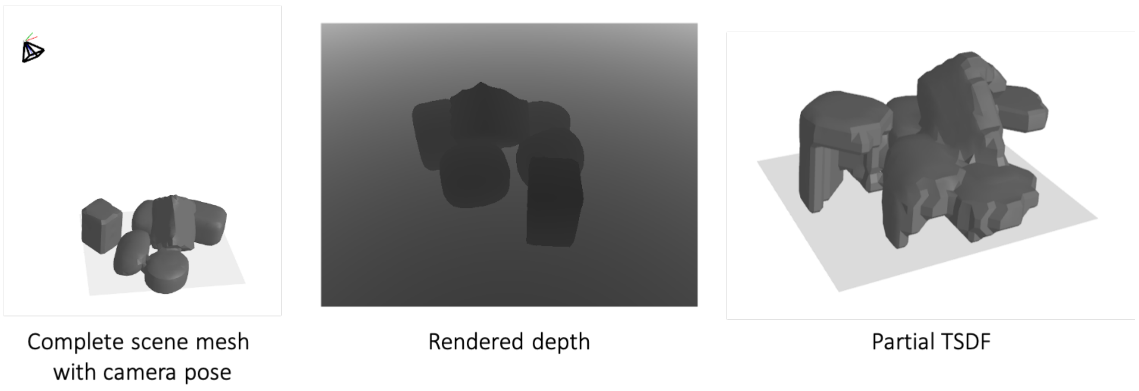


Figure 6.10: Illustration of the experimental setup for the depth to 3D decomposition task.

Results Qualitative results in Figure 6.11 demonstrate that our proposed architecture to predict an object-centric latent space can be used in a depth to 3D scene decomposition task. The network can successfully generate a per-object latent representation of a novel scene observed from a random viewpoint. Interestingly, our quantitative results (Table 6.2) show that training the network from random viewpoints, produces an overall better decomposition, with a marginally better average BCE Loss and a lower percentage of wrongly segmented scenes, compared to producing

Dataset	Dataset size	Avg. BCE Loss	Missing/joined objects	Duplicated/split objects
4-8 objects	95	0.0020	10.8 %	17.8 %

Table 6.2: Quantitative results for the depth to 3D decomposition task. Average Binary Cross Entropy is reported, as well as the percentage of scenes with missing or joined objects as well as those with duplicate or split objects.

the decomposition from a 3D occupancy grid. It is likely that the additional input-output variety provided by the random input viewpoints, allows for a better representation to be learnt. Note how, compared to the task of predicting the scene decomposition from the entire 3D occupancy grid, fewer scenes with duplicate representations or split objects are generated, while in more cases, objects are joined or missed.

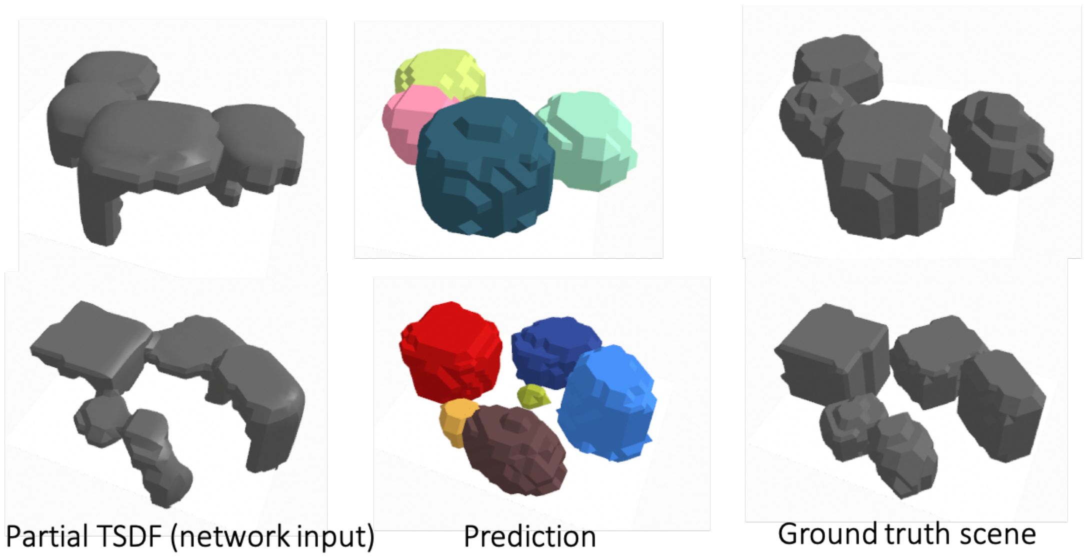


Figure 6.11: Qualitative results of the depth to 3D decomposition task.

6.2.3 A depth-conditioned VAE for single-view scene decomposition in latent space

Finally, we embed the proposed module to generate a factorised scene representation into a depth conditioned VAE, which learns a scene prior as in the SIMstack architecture in Chapter 5, but with an object-centric latent representation. This factorised latent space can then be reconstructed into individual objects, alleviating the model from post-processing the output with Hough-Voting to produce instance segmentation. Furthermore, the VAE encoder and decoder architectures are simplified, since they don't need task-specific heads for geometry and instance vector-voting fields (see Section 5).

Experimental setup Similarly to the previous experiment, the task is to predict a 3D scene decomposition from a random viewpoint; however, in this setting we also aim to learn a scene prior at the same time and use the full 3D occupancy grid as an input to the network as well. We train on the dataset of varying objects ($4-8$ objects per scene) with 10765 training examples. To generate random viewpoints, we follow the same procedure as before (Section 6.2.2).

Architecture details The architecture of the depth-conditioned VAE largely resembles the one used in SIMstack, except for the VAE latent embedding and the 3D scene decoder. The conditioning autoencoder network is unchanged and learns how to encode and decode a partial TSDF while providing expressive, view-specific features maps from its encoder to the 3D scene VAE. Similarly to the SIMstack model, during training, the network is also provided with the encoding from a full 3D scene. However, in this experiment, a scene occupancy grid is encoded instead of a TSDF. This choice is mainly made for consistency, since we model the scene output as a composition of per-object occupancy grids (see Section 6.2.1). The 3D scene encoder, conditioned on the view-features (through concatenation of output features), predicts a distribution over scenes in form of a global feature volume, from which a scene is then sampled. From this scene, N object codes are extracted using our proposed module and individually reconstructed into per-object occupancy grids using a shared decoder. Similarly to the 3D scene encoder, the feature maps of the shared decoder are concatenated with the input view features - the decoder is also conditioned on the random viewpoint features. An overview of the architecture is presented in Figure 6.12. For the experiments we use 64 latent channels for the partial TSDF encoding and the scene feature volume, which is downsampled from a resolution of 64 to a resolution of 4. The full depth-conditioned VAE has just over $2.1m$ parameters. Similarly to the SIMstack model, we use Squeeze and Excitation units [Hu et al., 2018] and residual connections [He et al., 2016] in both 3D encoder and decoders.

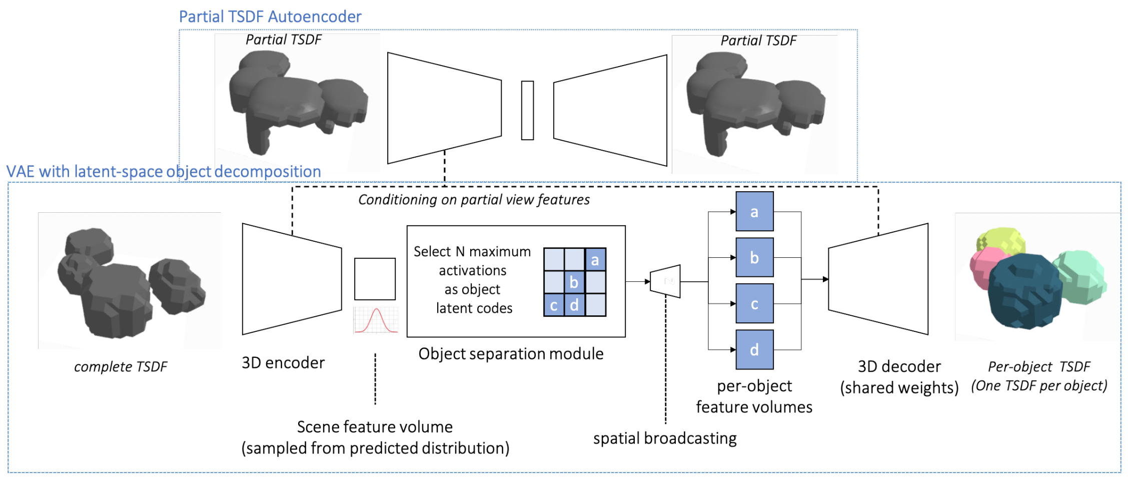

Partial TSDF Autoencoder
Partial TSDF
Partial TSDF
VAE with latent-space object decomposition
Conditioning on partial view features
complete TSDF
3D encoder
Select N maximum activations as object latent codes
a
b
c
d
Object separation module
Scene feature volume (sampled from predicted distribution)
spatial broadcasting
per-object feature volumes
3D decoder (shared weights)
Per-object TSDF (One TSDF per object)

Figure 6.12: An illustration of the depth-conditioned scene decomposition VAE architecture.

Loss function While we train the auxiliary task of the partial TSDF reconstruction with a simple L1 loss as $|\hat{y}_{partial_{TSDF}} - y_{partial_{TSDF}}|$, the main decomposition task is optimised using Hungarian Matching $M(\cdot)$ and a Binary Cross Entropy Loss BCE between the occupancy grid prediction $\hat{y}_{occupancy}$ and ground truth $y_{occupancy}$, as in previous experiments. At the same time, we add a variational optimisation component, to assure that the global feature grid encoding stays true to a multinomial gaussian distribution $\mathbb{G}$ that can be sampled from. We follow the method used in Chapter 5 and use the Maximum-Mean Discrepancy D_{MMD} [Zhao et al., 2017b] instead of a standard KL-divergence [Kingma and Welling, 2014] to compute the distance between the predicted distribution $p(z|x)$ and the multinomial gaussian $\mathbb{G}$; our final loss is set as:

$$\begin{aligned} L = \alpha\ &|\hat{y}_{partial_{TSDF}} - y_{partial_{TSDF}}| + \\ &\beta\ D_{MMD}(p(z|x), \mathbb{G}) + \\ &\gamma\, BCE(M(\{\hat{y}_{occupancy}\}, \{y_{occupancy}\})), \end{aligned} \tag{6.21}$$

where the respective weightings α, β and γ are set to $1, 1$ and 200, which we find empirically.

Training details Similarly to previous experiments, we train our model with the Adam optimizer [Kingma and Ba, 2015b] and an initial learning rate of $5e^{-4}$, which is adjusted using step-wise learning rate decay (step size of 10 and $\gamma = 0.999$).

Results We evaluate the depth-conditioned scene decomposition network in terms of its reconstruction accuracy, as well as the quality of its sample space. We find that the network is able to reconstruct unseen scenes well when given both 3D and 2D input modalities (see Figure 6.13, top row). However, with the current architecture and training settings, the model fails to learn a representative prior over the factorised latent space – sampling the learnt distribution while conditioning on a depth view results in incomplete reconstructions (see Figure 6.13, bottom row). We attribute this failure to the current architecture design, which does not enforce any structure in the generated distribution, which is expected to generate sparse feature volumes with neatly clustered object features. It is likely that more structure and guidance during training is needed. One possible architecture adjustment which could improve learning would be to sample the number of objects and their features separately.

6.3 Discussion and future work

In the presented experiments, we validate our proposed architecture to predict an object-centric, factorised latent representation on synthetic data, both from a full 3D occupancy grid, as well as

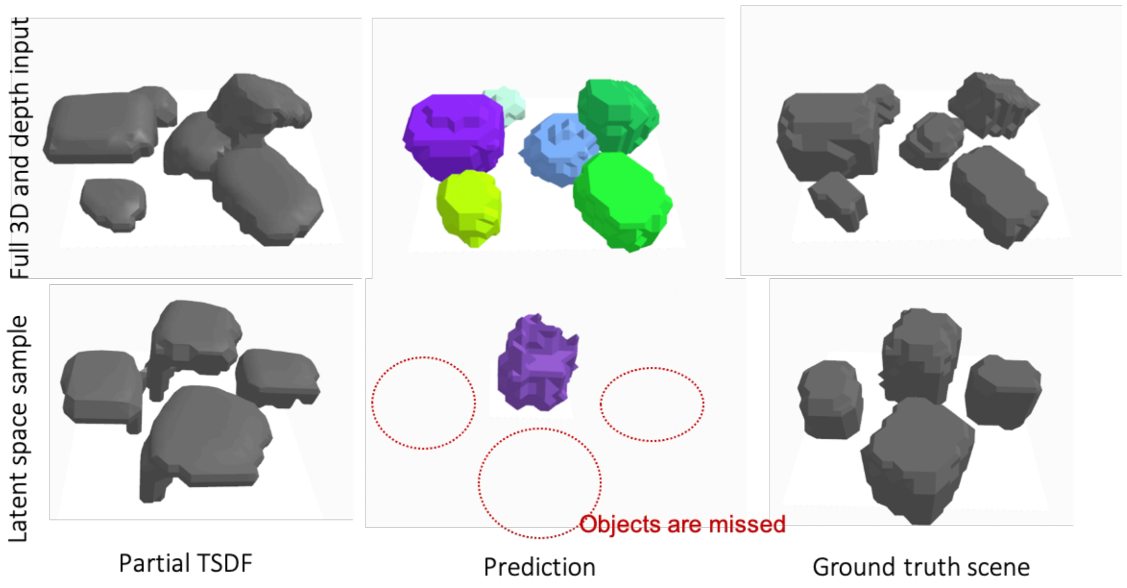


Figure 6.13: Initial results of the depth to 3D scene decomposition VAE architecture, trained on our synthetic dataset with $4-8$ objects. **Top row**: the reconstruction of a validation scene when providing the model with all input modalities. **Bottom row**: reconstruction of a random latent space sample, conditioned on a depth view.

from a random depth view. We find that from both input modalities, our method is able to predict the decomposition and reconstruction of previously unseen scenes with varying numbers of Superquadric shapes, randomly placed under physics simulation. We display visual results in Figures 6.11, 6.6 and 6.7, as well as quantitative results in Tables 6.1 and 6.2. Our final experiments on training the scene factorisation module within a full depth-conditioned VAE model did not yield convincing results and we were at this stage not able to extend the full SIMstack pipeline with an object-centric scene representation. However, our initial results provide a promising direction and in the following, we discuss some of the open questions and possible points of continuation for future work.

Improving the latent representation Firstly, a more in-depth analysis of the latent space representation learnt by our model might give insights into how to improve the architecture. We provide an initial analysis of the latent space in Figure 6.8, which shows that the difference between the mean-value of object vectors and those representing empty space is small. This may lead to selecting a latent that represents empty space during the N-max operation in the object separation module and could explain the missing objects in some of our results. However, a more in-depth analysis of the latent space using e.g. t-distributed Stochastic Neighbour Embedding (t-SNE) could lead to a better understanding of how the network reasons about and separates different object shapes, sizes and locations.

Secondly, we find that the network stores the object feature vectors in a fixed set of N locations within the feature volume (see Figure 6.9). This is sub-optimal since the feature vectors have to encode position as well as appearance. It would be better if the network learned to use each object's centre location for it's respective feature vector. This could be achieved by training with external object-centre supervision or by explicitly separating the prediction of object feature and object spatial transform.

Our method does not yet explicitly handle duplicated object latent codes, which are generated in some of the cases. Approaches in literature that handle similar set prediction tasks, either use post-processing methods such as non-maximum suppression of bounding boxes [Ren et al., 2015] or object centre predictions [Wang et al., 2020b], or, use attention-based methods such as the Transformer model for end-to-end object detection proposed by [Carion et al., 2020]. Others still, use iterative encoding methods [Burgess et al., 2019], which implicitly propagate knowledge about respective existence between the individual scene component encodings and avoids duplication. Our method specifically avoids iterative encoding methods for simplicity, but could potentially benefit from attention-based layers being introduced before or after the per-object latent representation is extracted. Another option would be to use post-processing methods in the flavour of non-maximum suppression on scenes which contain duplicate object encodings. Even if such duplications which currently lie below 18% (or even lower, since in some cases the wrongly predicted number of objects is due to over-segmentation) can't be further reduced, the method will remain in principle more efficient than the shape and instance prediction model proposed in Chapter 5 which requires post-processing for object extraction in every scene. As already mentioned in Section 6.2.1, further improvements could be obtained through external supervision on placing each detected object's encoding in the object's centre, as well as numerically encouraging a larger separation between objects and free space representation in the lower dimensional manifold.

Testing on real data To be fully validated, the proposed approach will have to be tested on real data examples, similarly to those presented in Chapter 5. As demonstrated in Chapter 5, we expect the transfer to real data to be possible without additional transfer learning methods, since simulated and real depth maps are sufficiently similar given good calibration and the further smoothing introduced by the transformation of depth views to partial TSDF grids (see the method section of Chapter 5).

Adding RGB cues Texture and color-based cues obtained from the integration of RGB information in the view-conditioning may reduce over-segmentation of objects. However, such additional features may also hinder the generalisation capabilities to real data, since the disparities between simulated and real RGB images are often very large.

Output representation Lastly, although formulating the scene factorisation as a decomposition into individual occupancy grids presents a principled representation, it is limiting, in particular, for objects of more complex shape. A natural extension of the approach will therefore be to train for a reconstruction of either SDF grids or more memory efficient representations such as neural implicit functions or neural radiance fields.

6.4 Conclusion

In this chapter, we present and test a novel method to generate a factorised latent representation of a 3D scene. The proposed architecture generates a global feature volume with a simple CNN encoder, from which *N* object embeddings are extracted and broadcasted into object-specific feature volumes using our object extraction module. Each object feature volume is then decoded into an object specific occupancy grid; together all reconstructed occupancy grids compose the full scene.

Our method is simple compared to many other approaches, which use iterative or recurrent encoding methods, and is able to decompose synthetic scenes of up to 8 Superquadric shapes, both from a full 3D occupancy scene representation and a single depth view.

A factorised latent representation of a scene offers a compact, disentangled and versatile representation and is interpretable compared to one joint scene encoding. Adopting such a representation within the shape and instance generation model SIMstack presented in Chapter 5, will remove post-processing steps and simplify the pipeline as well as the architecture. Although the proposed scene decomposition module fails to learn a factorised prior when integrated within a depth-conditioned VAE, our preliminary results on decomposing a scene from 3D occupancy and a single depth view present a promising direction to be further explored by future work.

CHAPTER 7

Discussion and Conclusion

The research presented in this thesis addresses several aspects of visual scene understanding, with a focus on 3D scenes that contain a collection of small objects. In particular, research questions involving the labelling, reasoning about, and the decomposition of such scenes were explored. In this final chapter, each project is briefly revisited, discussed and put into context with the latest state of the art. Finally, some directions for future work are suggested.

Visual scene understanding is a crucial component for intelligent and autonomous systems and despite the substantial progress the field of computer vision has seen, since the introduction of deep learning, many problems remain unsolved. Applying deep learning methods to the field of 2D image classification and segmentation has pushed these algorithms to human level performance and in some cases, even surpassed it. However, the transfer of these tasks to 3D scenes remains difficult, as does the understanding, modelling and efficient representation of the 3D domain.

While current systems are able to reconstruct the geometry of a scene to a high degree of accuracy, while navigating through it (using state of the art SLAM algorithms) [Engel et al., 2014a, Whelan et al., 2015], many questions remain open on how to best annotate these reconstructions with semantics and detect objects within them. Although many systems have proposed solutions (e.g. [McCormac et al., 2017a, Nie et al., 2020b]), some even handling dynamic scenes [Rünz and Agapito, 2017], final results in 3D segmentation and object detection/instance segmentation often lack precision and consistency. Systems also still struggle to generalise and often fail once presented with different lighting conditions or previously unseen object categories. In addition, it is still unclear if 3D annotation is best achieved through label reprojection from 2D segmentation results or obtained through directly labelling the reconstruction itself.

In the first chapter of this thesis, a study on this question is presented. For the setting of a table-top scene which is extensively scanned using a real-time SLAM system, methods for view-based and map-based semantic labelling are designed and compared. To allow for a principled comparison, a height map representation was chosen for the mapping component; this enabled the use of the same CNN architecture for obtaining 2D semantic labels through the view-based method and 3D semantic labels from directly labelling the reconstruction. The results demonstrated that as long as the semantic segmentation network for map-based labelling provides reliable segmentation, directly labelling the reconstruction (once the scene has been fully reconstructed) results in cleaner segmentation borders and is in principle more efficient, since every scene element is only labelled once. In contrast, view-based labelling with re-projection of the labels and subsequent label-fusion into the map results in more errors in object border regions and is principally less efficient, since during extensive scanning every scene element is seen and labelled multiple times. However, in the case where only labelling networks with reduced accuracy are available, more labelling errors will be recovered during the view-based method thanks to the Bayesian label fusion.

The results from this study provide a principled comparison within our confined setting of a height map representation; for applications with different 3D representations (e.g. pointclouds, voxel-grids), they would have to be considered in combination with additional factors such as the cost of directly labelling the 3D map using representation-specific network architectures such as a 3D CNN or PointNet. A direct point of continuation of this project would therefore be a more extensive study including the common 3D representations for real-time SLAM. A particularly interesting and challenging aspect of moving to full 3D representations will be studying the trade-off between view-based and map-based labelling for larger and complex objects whose shape doesn't fit into one viewpoint and makes it difficult to reach all viewpoints necessary to obtain a full reconstruction.

As already mentioned in the discussion of Chapter 4, most applications would in fact benefit from a joint application of view-based and map-based labelling, where the view-based method provides labels early on with a higher tolerance to mislabelling and the map-based method provides regular refinements which benefit from larger context integration due to processing a full 3D representation. In addition it would be interesting to integrate semantic labels into the overall optimisation pipeline of SLAM. This has recently been explored by [Hempel and Al-Hamadi, 2022], whose system leverages obtained labels to improve tracking accuracy. Other approaches such as [Zhi et al., 2019] have focused on jointly optimising geometry and semantics, removing the independence assumption between individual labels usually adopted in view-based methods.

Finally, with the recent emergence of neural scene representation networks [Park et al., 2019a, Mildenhall et al., 2020] and their application to real-time SLAM [Sucar et al., 2021], the question arises of whether it will remain relevant to label an explicit 3D representation or if future research should focus on how to robustly add labels to neural scene representations. Some research such as [Zhi et al., 2021] already exists in this direction.

In current reconstruction systems, the target scene has to be scanned extensively to obtain a full 3D model. These scanning trajectories can be time-consuming and in many cases, not every viewpoint can be reached to obtain a full reconstruction without missing parts [Dai et al., 2017a]. As humans we are able to estimate shape in areas invisible to us and generate a 3D model from a partial view (in our minds). We can even reason about object interactions — i.e. given a partial view, we can generate a full reconstruction and simulate into the future; if something looks unstable, we can guess where and how far objects will fall or roll and we can make a guess about how many objects lie underneath and behind an object that is visible to us. Many consider this ability a crucial feature for fully intelligent systems. Not only does the ability to reason and complete the unseen bring autonomous systems closer to human-level reasoning, but it also presumes a consistent and meaningful representation of the world, which will be helpful for many other tasks. In addition, being able to reconstruct and simulate from a partial view will alleviate systems from the need of comprehensively scanning a scene, as was the setting in the study of Chapter 4.

Chapter 5 of this thesis addresses this topic and proposes a novel method, SIMstack, to estimate shape and instances in occluded regions from a single depth image. Compared to related work, which either completes the 3D shape of single or multiple objects from a single observation or predicts instance segmentation for a 2D image, the proposed method combines both tasks in a generative formulation for a 3D scene. The dataset is selected as a set of Superquadrics, which provide a large variety of convex shapes. An additional extension of the model for integrating multiple views through either multi-view conditioning or multi-view optimisation with differentiable rendering is also presented.

The proposed method is able to provide instance segmentation in visible regions while completing shape and generating plausible decompositions in occluded areas. While the model can complete scenes with up to 7 objects and generalises to real data, it was not tested on much larger scenes, which would be one of the direct points of continuation. Another open question is how well it extends to non-convex shapes. Although some experiments were provided in Section 5.6, it would be interesting to see how well SIMstack can perform when trained on non-convex shapes and using an appropriate post-processing method. Finally, the method could benefit from more structured representations of the objects themselves, using per-object latent codes (which is explored in Chapter 6) or from more explicitly modelling object relations. The latter could be achieved by using a graph representation instead of a joint latent code.

Finally, while SIMstack learns to imitate physical *intuition* by learning a distribution over realistic and stable (under physics simulation) piles, it would be interesting to integrate more explicit knowledge about physical properties into the system; one could take inspiration from approaches on stability prediction [Groth et al., 2018], approaches which learn and model interactions [Engelcke et al., 2020, Ehrhardt et al., 2020] and recently proposed fully differentiable physics engines [Freeman et al., 2021, de Avila Belbute-Peres et al., 2018].

The ability to reason about 3D scenes will be highly correlated with the way those scenes are represented. Most current methods either use full-scale 3D geometry to represent scenes or compressed encodings in latent spaces, in which all features are highly correlated. One possible option to introduce disentangled features and scene components, while retaining the memory efficiency of encoded latent representations are *factorised* latent spaces. Some approaches have already explored such representations, mainly for 2D images [Burgess et al., 2019, Greff et al., 2017, Locatello et al., 2020].

Chapter 6 explores how to obtain such a factorised latent representation in 3D, with a focus on extending SIMstack to encode a scene as a composition of object-specific latent codes, all from a single depth image. A novel object-extraction module is proposed and tested on synthetic scenes of Superquadric collections, similar to those of the training dataset of SIMstack. While the module is able to predict factorised latent spaces from full 3D scenes as well as partial views in a discriminatory task, embedding it into the full generative pipeline of SIMstack did not yield good results. One likely cause is the lack of structural adaptation of the latent space to the generative task. Finding a more appropriate formulation for this structure would be the first point of continuation of this project. Several other improvements and extensions are possible including a more compact encoding of the objects as individual neural scene representations. Inspirations for such compositional neural representations include [Niemeyer and Geiger, 2021] and [Ost et al., 2021].

In **conclusion**, with the goal to improve the visual scene understanding of autonomous vision systems, the research presented in this thesis has provided insights in 3D real-time semantic labelling and proposed novel solutions for reasoning about shape and instances and the decomposition of 3D scenes. Future developments will likely move from explicit 3D geometry to scene representations which are encoded inside neural network weights (neural scene representations) or compact latent code structures. To be able to reason about 3D space, scene encodings will likely benefit from being modular and representing a scene as a composition of objects and regions. Other aspects which will be defining the perceptual intelligence of AI systems are the ability to simulate the environment with its possible future outcomes, understanding causality and complex relationships between entities and scenarios. Self-learning through exploration and self-analysis to understand its own limitations will be important components as well.

While we are progressing at high speed and approaching a new era where artificial intelligence and robotics are present in everyday life, many interesting problems on the way are still waiting to be solved.

Bibliography

[Abadi et al., 2016] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I. J., Harp, A., Irving, G., Isard, M., Jia, Y., Józefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D. G., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P. A., Vanhoucke, V., Vasudevan, V., Viégas, F. B., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2016). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. CoRR, abs/1603.04467. 44

[Amenta et al., 2001] Amenta, N., Choi, S., and Kolluri, R. K. (2001). The power crust, unions of balls, and the medial axis transform. Comput. Geom., 19(2-3):127–153. 54

[Arora, 2020] Arora, S. (2020). https://www.aitude.com/supervised-vs-unsupervised-vs-reinforcement/. Last accessed: 19/04/2022. 20

[Aulinas et al., 2008] Aulinas, J., Pétillot, Y. R., Salvi, J., and Lladó, X. (2008). The slam problem: a survey. In CCIA. 26

[Ballard, 1981] Ballard, D. (1981). Generalizing the hough transform to detect arbitrary shapes. Pattern Recognit., 13:111–122. 93

[Bardinet et al., 1995] Bardinet, E., Cohen, L. D., and Ayache, N. (1995). A parametric deformable model to fit unstructured 3D data. RR-2617, INRIA. 98

[Bay et al., 2008] Bay, H., Ess, A., Tuytelaars, T., and Gool, L. V. (2008). Speeded-up robust features (surf). Comput. Vis. Image Underst., 110:346–359. 16

[Bishop, 2006] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer-Verlag New York, Inc. 39, 40

[Bishop and Nasrabadi, 2007] Bishop, C. M. and Nasrabadi, N. M. (2007). Pattern recognition and machine learning. J. Electronic Imaging, 16:049901. 32

[Bloesch et al., 2018] Bloesch, M., Czarnowski, J., Clark, R., Leutenegger, S., and Davison, A. J. (2018). CodeSLAM — learning a compact, optimisable representation for dense visual SLAM. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 18, 129

[Bloesch et al., 2019] Bloesch, M., Laidlow, T., Clark, R., Leutenegger, S., and Davison, A. J. (2019). Learning meshes for dense visual SLAM. In Proceedings of the International Conference on Computer Vision (ICCV). 128

[Bohg et al., 2014] Bohg, J., Morales, A., Asfour, T., and Kragic, D. (2014). Data-driven grasp synthesis—a survey. IEEE Transactions on Robotics (T-RO). 123

[Bolya et al., 2019] Bolya, D., Zhou, C., Xiao, F., and Lee, Y. J. (2019). Yolact: Real-time instance segmentation. 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pages 9156–9165. 91

[Boston Dynamics Team, 2021a] Boston Dynamics Team (2021a). Atlas. https://blog.bostondynamics.com/atlas. Last accessed 31/01/2022. 8

[Boston Dynamics Team, 2021b] Boston Dynamics Team (2021b). Spot 3.0. https://blog.bostondynamics.com/spot-release-3.0-flexible-autonomy-and-repeatable-data-capture. Last accessed 31/01/2022. 8

[Brachmann et al., 2014] Brachmann, E., Krull, A., Michel, F., Gumhold, S., Shotton, J., and Rother, C. (2014). Learning 6d object pose estimation using 3d object coordinates. In ECCV. 84

[Bruno and Colombini, 2021] Bruno, H. M. S. and Colombini, E. L. (2021). LIFT-SLAM: a deep-learning feature-based monocular visual SLAM method. CoRR, abs/2104.00099. 18

[Burgess et al., 2019] Burgess, C. P., Matthey, L., Watters, N., Kabra, R., Higgins, I., Botvinick, M. M., and Lerchner, A. (2019). Monet: Unsupervised scene decomposition and representation. ArXiv, abs/1901.11390. 131, 132, 134, 146, 152

[Cadena et al., 2016a] Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., and Leonard, J. J. (2016a). Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. IEEE Transactions on Robotics (T-RO), 32(6):1309–1332. 48

[Cadena et al., 2016b] Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I. D., and Leonard, J. J. (2016b). Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. IEEE Transactions on Robotics, 32:1309–1332. 24, 26, 27, 28

[Cadena et al., 2015] Cadena, C., Dick, A. R., and Reid, I. D. (2015). A fast, modular scene understanding system using context-aware object detection. 2015 IEEE International Conference on Robotics and Automation (ICRA), pages 4859–4866. 17

[Canny, 1986] Canny, J. (1986). A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-8(6):679–698. 14, 16

[Carion et al., 2020] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. (2020). End-to-end object detection with transformers. In Proceedings of the European Conference on Computer Vision (ECCV). 41, 82, 83, 91, 137, 146

[Castle et al., 2007] Castle, R. O., Gawley, D. J., Klein, G., and Murray, D. W. (2007). Towards simultaneous recognition, localization and mapping for hand-held and wearable cameras. In Proceedings 2007 IEEE International Conference on Robotics and Automation, pages 4102–4107. 18

[Ceron, 2019] Ceron, R. (2019). https://www.ibm.com/blogs/systems/ai-machine-learning-and-deep-learning-whats-the-difference/. Last accessed: 08/07/2022. 19

[Chabra et al., 2020] Chabra, R., Lenssen, J. E., Ilg, E., Schmidt, T., Straub, J., Lovegrove, S., and Newcombe, R. (2020). Deep local shapes: Learning local SDF priors for detailed 3D reconstruction. In Proceedings of the European Conference on Computer Vision (ECCV). 82

[Chang et al., 2015] Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., and Yu, F. (2015). ShapeNet: An information-rich 3D model repository. arXiv preprint arXiv:1512.03012. 66, 86

[Chen et al., 2020a] Chen, H., Sun, K., Tian, Z., Shen, C., Huang, Y., and Yan, Y. (2020a). Blendmask: Top-down meets bottom-up for instance segmentation. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 8570–8578. 15

[Chen et al., 2017] Chen, L.-C., Papandreou, G., Schroff, F., and Adam, H. (2017). Rethinking atrous convolution for semantic image segmentation. ArXiv, abs/1706.05587. 15

[Chen et al., 2019] Chen, X., Girshick, R. B., He, K., and Dollár, P. (2019). Tensormask: A foundation for dense object segmentation. 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pages 2061–2069. 15, 91

[Chen et al., 2020b] Chen, Z., Tagliasacchi, A., and Zhang, H. (2020b). Bsp-net: Generating compact meshes via binary space partitioning. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 42–51. 132

[Chen and Zhang, 2019] Chen, Z. and Zhang, H. (2019). Learning implicit fields for generative shape modeling. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 5932–5941. 129

[Cheriet et al., 1998] Cheriet, M., Said, J. N., and Suen, C. Y. (1998). A recursive thresholding technique for image segmentation. IEEE transactions on image processing : a publication of the IEEE Signal Processing Society, 7 6:918–21. 14

[Chevalier et al., 2003] Chevalier, L., Jaillet, F., and Baskurt, A. (2003). Segmentation and superquadric modeling of 3D objects. Journal of WSCG, 11(10). 98

[Choy et al., 2016] Choy, C. B., Xu, D., Gwak, J., Chen, K., and Savarese, S. (2016). 3D-R2N2: A unified approach for single and multi-view 3D object reconstruction. In Proceedings of the European Conference on Computer Vision (ECCV). 81

[Cigla and Alatan, 2008] Cigla, C. and Alatan, A. A. (2008). Region-based image segmentation via graph cuts. In Proceedings of the International Conference on Image Processing, ICIP 2008, October 12-15, 2008, San Diego, California, USA, pages 2272–2275. IEEE. 14

[Conveyer Concepts Inc, 2020] Conveyer Concepts Inc (2020). Future of industrial robots. `https://conveyorconceptsinc.com/blog/future-of-industrial-robotics/`. Last accessed: 19/03/2022. 8

[Coumans and Bai, 2019] Coumans, E. and Bai, Y. (2016–2019). PyBullet, a Python module for physics simulation for games, robotics and machine learning. `http://pybullet.org`. 88

[Crawford and Pineau, 2019] Crawford, E. and Pineau, J. (2019). Spatially invariant unsupervised object detection with convolutional neural networks. In AAAI. 130, 133, 134

[Czarnowski et al., 2020] Czarnowski, J., Laidlow, T., Clark, R., and Davison, A. J. (2020). Deepfactors: Real-time probabilistic dense monocular slam. IEEE Robotics and Automation Letters, 5(2):721–728. 18

[Dai et al., 2017a] Dai, A., Chang, A. X., Savva, M., Halber, M., Funkhouser, T., and Nießner, M. (2017a). ScanNet: Richly-annotated 3d reconstructions of indoor scene. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 151

[Dai and Nießner, 2018] Dai, A. and Nießner, M. (2018). 3dmv: Joint 3d-multi-view prediction for 3d semantic scene segmentation. In ECCV. 51

[Dai and Nießner, 2019] Dai, A. and Nießner, M. (2019). Scan2Mesh: From unstructured range scans to 3D meshes. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 81

[Dai et al., 2018] Dai, A., Ritchie, D., Bokeloh, M., Reed, S., Sturm, J., and Nießner, M. (2018). ScanComplete: Large-scale scene completion and semantic segmentation for 3D scans. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 15, 50, 81

[Dai et al., 2017b] Dai, A., Ruizhongtai Qi, C., and Nießner, M. (2017b). Shape completion using 3D-Encoder-Predictor CNNs and shape synthesis. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 82

[Davison, 2018] Davison, A. J. (2018). FutureMapping: The computational structure of Spatial AI systems. arXiv preprint arXiv:arXiv:1803.11288. 48

[Davison et al., 2007] Davison, A. J., Molton, N. D., Reid, I., and Stasse, O. (2007). MonoSLAM: Real-Time Single Camera SLAM. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 29(6):1052–1067. 17

[Davison and Ortiz, 2019] Davison, A. J. and Ortiz, J. (2019). FutureMapping 2: Gaussian Belief Propagation for Spatial AI. arXiv preprint arXiv:arXiv:1910.14139. 9

[Dawson-Haggerty, 2019] Dawson-Haggerty (2019). trimesh. 66

[de Avila Belbute-Peres et al., 2018] de Avila Belbute-Peres, F., Smith, K. A., Allen, K. R., Tenenbaum, J. B., and Kolter, J. Z. (2018). End-to-end differentiable physics for learning and control. In NeurIPS. 152

[Deep Mind, 2015] Deep Mind (2015). https://www.deepmind.com/research/highlighted-research/alphago. Last accessed 28/04/2022. 19

[Deep Mind, 2021] Deep Mind (2021). https://www.deepmind.com/blog/alphafold-a-solution-to-a-50-year-old-grand-challenge-in-biology. Last accessed 28/04/2022. 19

[Deep Mind, 2022] Deep Mind (2022). https://www.deepmind.com/blog/accelerating-fusion-science-through-learned-plasma-control. Last accessed 28/04/2022. 19

[Dellaert, 2021] Dellaert, F. (2021). Factor graphs: Exploiting structure in robotics. 28

[Deng et al., 2020a] Deng, B., Genova, K., Yazdani, S., Bouaziz, S., Hinton, G., and Tagliasacchi, A. (2020a). CvxNet: Learnable convex decomposition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 82

[Deng et al., 2020b] Deng, B., Genova, K., Yazdani, S., Bouaziz, S., Hinton, G. E., and Tagliasacchi, A. (2020b). Cvxnet: Learnable convex decomposition. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 31–41. 132

[Dosovitskiy et al., 2020] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. CoRR, abs/2010.11929. 21

[Durner et al., 2021] Durner, M., Boerdijk, W., Sundermeyer, M., Friedl, W., Marton, Z.-C., and Triebel, R. (2021). Unknown object segmentation from stereo images. 91

[Durrant-Whyte and Bailey, 2006] Durrant-Whyte, H. F. and Bailey, T. (2006). Simultaneous localisation and mapping (slam) : Part i the essential algorithms. 16

[Dyson UK, 2022] Dyson UK (2022). Dyson 360 Heurist. https://www.dyson.co.uk/vacuum-cleaners/robot-vacuums/dyson-360-heurist/dyson-360-heurist-overview. Last accessed: 28/01/2022. 8

[Ehrhardt et al., 2020] Ehrhardt, S., Groth, O., Monszpart, A., Engelcke, M., Posner, I., Mitra, N., and Vedaldi, A. (2020). Relate: Physically plausible multi-object scene synthesis using structured latent spaces. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, Advances in Neural Information Processing Systems, volume 33, pages 11202–11213. Curran Associates, Inc. 152

[Engel et al., 2014a] Engel, J., Schoeps, T., and Cremers, D. (2014a). LSD-SLAM: Large-scale direct monocular SLAM. In Proceedings of the European Conference on Computer Vision (ECCV). 149

[Engel et al., 2014b] Engel, J., Schöps, T., and Cremers, D. (2014b). Lsd-slam: Large-scale direct monocular slam. In ECCV. 17

[Engelcke et al., 2020] Engelcke, M., Kosiorek, A. R., Jones, O. P., and Posner, I. (2020). Genesis: Generative scene inference and sampling with object-centric latent representations. ArXiv, abs/1907.13052. 131, 132, 152

[Engelmann et al., 2020] Engelmann, F., Bokeloh, M., Fathi, A., Leibe, B., and Nießner, M. (2020). 3d-mpa: Multi-proposal aggregation for 3d semantic instance segmentation. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 9028–9037. 92

[Engineered Arts, 2022] Engineered Arts (2022). Robotics - Ameca robot. https://www.engineeredarts.co.uk/robot/ameca/. Last accessed 31/01/2022. 8

[Eslami et al., 2016] Eslami, S. M. A., Heess, N. M. O., Weber, T., Tassa, Y., Szepesvari, D., Kavukcuoglu, K., and Hinton, G. E. (2016). Attend, infer, repeat: Fast scene understanding with generative models. In NIPS. 130, 134

[Farabet et al., 2013] Farabet, C., Couprie, C., Najman, L., and LeCun, Y. (2013). Learning hierarchical features for scene labeling. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35:1915–1929. 14

[Firman et al., 2016] Firman, M., Mac Aodha, O., Julier, S., and Brostow, G. J. (2016). Structured prediction of unobserved voxels from a single depth image. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 81, 83

[Fischler and Bolles, 1981] Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM, 24:381–395. 25

[Foegleman, 2014] Foegleman, M. (2014). `https://www.michaelfogleman.com/projects/hirise/`. Last accessed 21/04/2022. 30

[Freeman et al., 2021] Freeman, C. D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I., and Bachem, O. (2021). Brax - a differentiable physics engine for large scale rigid body simulation. ArXiv, abs/2106.13281. 152

[Fukushima, 1969] Fukushima, K. (1969). Visual feature extraction by a multilayered network of analog threshold elements. IEEE Trans. Syst. Sci. Cybern., 5(4):322–333. 19

[Garland, 1998] Garland, M. (1998). Fast polygonal approximation of terrains and height fields. 30

[Gebrehiwot et al., 2019] Gebrehiwot, A., Hashemi-Beni, L., Thompson, G., Kordjamshidi, P., and Langan, T. E. (2019). Deep convolutional neural network for flood extent mapping using unmanned aerial vehicles data. Sensors, 19(7). 34

[Genova et al., 2020] Genova, K., Cole, F., Sud, A., Sarna, A., and Funkhouser, T. A. (2020). Local deep implicit functions for 3d shape. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 4856–4865. 132

[Girshick, 2015] Girshick, R. (2015). Fast r-cnn. In Proceedings of the International Conference on Computer Vision (ICCV). 40

[Girshick et al., 2014] Girshick, R. B., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. 2014 IEEE Conference on Computer Vision and Pattern Recognition, pages 580–587. 14

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep Learning. MIT Press. 33

[Greff et al., 2019] Greff, K., Kaufman, R. L., Kabra, R., Watters, N., Burgess, C. P., Zoran, D., Matthey, L., Botvinick, M. M., and Lerchner, A. (2019). Multi-object representation learning with iterative variational inference. ArXiv, abs/1903.00450. 129, 131

[Greff et al., 2017] Greff, K., van Steenkiste, S., and Schmidhuber, J. (2017). Neural expectation maximization. In NIPS. 130, 152

[Grinvald et al., 2019] Grinvald, M., Furrer, F., Novkovic, T., Chung, J., Cadena, C., Siegwart, R., and Nieto, J. (2019). Volumetric instance-aware semantic mapping and 3d object discovery. IEEE Robotics and Automation Letters, 4:3037–3044. 49, 50

[Groth et al., 2018] Groth, O., Fuchs, F. B., Posner, I., and Vedaldi, A. (2018). Shapestacks: Learning vision-based physical intuition for generalised object stacking. ArXiv, abs/1804.08018. 152

[Groueix et al., 2018] Groueix, T., Fisher, M., Kim, V. G., Russell, B. C., and Aubry, M. (2018). A papier-mâché approach to learning 3D surface generation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 82

[Han et al., 2020] Han, L., Zheng, T., Xu, L., and Fang, L. (2020). OccuSeg: Occupancy-aware 3D instance segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 82

[Häne et al., 2013] Häne, C., Zach, C., Cohen, A., Angst, R., and Pollefeys, M. (2013). Joint 3d scene reconstruction and class segmentation. 2013 IEEE Conference on Computer Vision and Pattern Recognition, pages 97–104. 17

[Hanocka et al., 2019] Hanocka, R., Hertz, A., Fish, N., Giryes, R., Fleishman, S., and Cohen-Or, D. (2019). Meshcnn: a network with an edge. ACM Transactions on Graphics (TOG), 38:1 – 12. 20, 52

[Harris and Stephens, 1988] Harris, C. G. and Stephens, M. J. (1988). A combined corner and edge detector. In Alvey Vision Conference. 16

[Hart, 1996] Hart, J. (1996). Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. The Visual Computer, 12:527–545. 43, 95

[Hazirbas et al., 2016] Hazirbas, C., Ma, L., Domokos, C., and Cremers, D. (2016). FuseNet: incorporating depth into semantic segmentation via fusion-based CNN architecture. In Proceedings of the Asian Conference on Computer Vision (ACCV). 68

[He et al., 2017] He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask R-CNN. In Proceedings of the International Conference on Computer Vision (ICCV). 15, 40, 82

[He et al., 2015] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. 2015 IEEE International Conference on Computer Vision (ICCV), pages 1026–1034. 20, 51

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 20, 33, 99, 102, 143

[Hempel and Al-Hamadi, 2022] Hempel, T. and Al-Hamadi, A. (2022). An online semantic mapping system for extending and enhancing visual slam. Engineering Applications of Artificial Intelligence. 150

[Herbst et al., 2011] Herbst, E. V., Ren, X., and Fox, D. (2011). Rgb-d object discovery via multi-scene analysis. 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 4850–4856. 17

[Hermans et al., 2014] Hermans, A., Floros, G., and Leibe, B. (2014). Dense 3D semantic mapping of indoor scenes from RGB-D images. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA). 50, 56

[Hesch et al., 2014] Hesch, J. A., Kottas, D. G., Bowman, S. L., and Roumeliotis, S. I. (2014). Camera-imu-based localization: Observability analysis and consistency improvement. The International Journal of Robotics Research, 33:182 – 201. 26

[Higgins et al., 2017] Higgins, I., Matthey, L., Pal, A., Burgess, C. P., Glorot, X., Botvinick, M. M., Mohamed, S., and Lerchner, A. (2017). beta-vae: Learning basic visual concepts with a constrained variational framework. In ICLR. 106, 129

[History Computer, 2021] History Computer (2021). https://history-computer.com/arthur-samuel-biography-history-and-inventions/. Last accessed 31/01/2022. 19

[Hou et al., 2019] Hou, J., Dai, A., and Niessner, M. (2019). 3d-sis: 3d semantic instance segmentation of rgb-d scans. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 51, 82, 92

[Hough, 1959] Hough, P. V. C. (1959). Machine Analysis of Bubble Chamber Pictures. Conf. Proc. C, 590914:554–558. 93

[Hu et al., 2018] Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-excitation networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 99, 102, 143

[Huang et al., 2021] Huang, J., Huang, S.-S., Song, H., and Hu, S. (2021). Di-fusion: Online implicit 3d reconstruction with deep priors. 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 8928–8937. 128

[Hutmacher, 2019] Hutmacher, F. (2019). Why is there so much more research on vision than on any other sensory modality? Frontiers in Psychology, 10. 13

[iRobot, 2022] iRobot (2022). Roomba Robot Vacuums. `https://www.irobot.co.uk/roomba/`. Last accessed: 28/01/2022. 8

[Izadi et al., 2011] Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R. A., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A. J., and Fitzgibbon, A. (2011). KinectFusion: Real-Time 3D Reconstruction and Interaction Using a Moving Depth Camera. In Proceedings of ACM Symposium on User Interface Software and Technolog (UIST). 17

[Jakli et al., 2000] Jakli, A., Leonardis, A., and Solina, F. (2000). Segmentation and recovery of superquadrics. In Computational Imaging and Vision. 85

[James et al., 2019a] James, S., Freese, M., and Davison, A. J. (2019a). Pyrep: Bringing v-rep to deep robot learning. arXiv preprint arXiv:1906.11176. 123

[James et al., 2019b] James, S., Wohlhart, P., Kalakrishnan, M., Kalashnikov, D., Irpan, A., Ibarz, J., Levine, S., Hadsell, R., and Bousmalis, K. (2019b). Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 123

[Jiang et al., 2020] Jiang, Y., Ji, D., Han, Z., and Zwicker, M. (2020). Sdfdiff: Differentiable rendering of signed distance fields for 3d shape optimization. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 116, 117

[Jordan, 2018] Jordan, J. (2018). `https://www.jeremyjordan.me/evaluating-image-segmentation-models/`. Last accessed 21/04/2022. 38

[K R et al., 2019] K R, P., Mukhopadhyay, R., Philip, J., Jha, A., Namboodiri, V., and Jawahar, C. V. (2019). Towards automatic face-to-face translation. In Proceedings of the 27th ACM

International Conference on Multimedia, MM '19, page 1428–1436, New York, NY, USA. Association for Computing Machinery. 129

[Kaess et al., 2008] Kaess, M., Ranganathan, A., and Dellaert, F. (2008). isam: Incremental smoothing and mapping. IEEE Transactions on Robotics, 24:1365–1378. 17

[Kaiming et al., 2017] Kaiming, H., Georgia, G., Piotr, D., and Ross, G. (2017). Mask r-cnn. In Proceedings of the IEEE international conference on computer vision, pages 2961–2969. 41, 91

[Karras et al., 2019] Karras, T., Laine, S., and Aila, T. (2019). A style-based generator architecture for generative adversarial networks. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 4396–4405. 20, 21, 129

[Katsoulas, 2004] Katsoulas, D. K. (2004). Superquadrics for object representation. University of Freiburg. 86

[Kingma and Ba, 2015a] Kingma, D. P. and Ba, J. (2015a). Adam: A method for stochastic optimization. CoRR, abs/1412.6980. 64, 68

[Kingma and Ba, 2015b] Kingma, D. P. and Ba, J. (2015b). Adam: A method for stochastic optimization. In Proceedings of the International Conference on Learning Representations (ICLR). 100, 106, 138, 144

[Kingma and Welling, 2014] Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. In Proceedings of the International Conference on Learning Representations (ICLR). 105, 144

[Kingma and Welling, 2019] Kingma, D. P. and Welling, M. (2019). An introduction to variational autoencoders. Found. Trends Mach. Learn., 12:307–392. 34, 37, 103

[Kirillov et al., 2019] Kirillov, A., He, K., Girshick, R. B., Rother, C., and Dollár, P. (2019). Panoptic segmentation. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 9396–9405. 15

[Klein and Murray, 2007] Klein, G. S. W. and Murray, D. W. (2007). Parallel tracking and mapping for small ar workspaces. 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, pages 225–234. 17

[Knopp et al., 2010] Knopp, J., Prasad, M., and Van Gool, L. (2010). Orientation invariant 3d object classification using hough transform based methods. 3DOR '10, page 15–20, New York, NY, USA. Association for Computing Machinery. 93

[Koppula et al., 2011] Koppula, H. S., Anand, A., Joachims, T., and Saxena, A. (2011). Semantic Labeling of 3D Point Clouds for Indoor Scenes. In Neural Information Processing Systems (NeurIPS). 17

[Kosiorek et al., 2021] Kosiorek, A. R., Strathmann, H., Zoran, D., Moreno, P., Schneider, R., Mokr'a, S., and Rezende, D. J. (2021). Nerf-vae: A geometry aware 3d scene generative model. ArXiv, abs/2104.00587. 129

[Kottas et al., 2012] Kottas, D. G., Hesch, J. A., Bowman, S. L., and Roumeliotis, S. I. (2012). On the consistency of vision-aided inertial navigation. In ISER. 26

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, Advances in Neural Information Processing Systems, volume 25. Curran Associates, Inc. 20

[Kuhn, 1955] Kuhn, H. W. (1955). The hungarian method for the assignment problem. Naval Research Logistics Quarterly, 2:83–97. 137

[KUKA AG, 2022] KUKA AG (2022). robotics systems. https://www.kuka.com/en-gb/products/robotics-systems/. Last accessed: 27/01/2022. 9

[Landrieu and Simonovsky, 2018] Landrieu, L. and Simonovsky, M. (2018). Large-scale point cloud semantic segmentation with superpoint graphs. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 4558–4567. 50

[Lee, 2019] Lee, D. (2019). https://medium.com/aiC2B3-theory-practice-business/reinforcement-learning-part-1-a-brief-introduction-a53a849771cf. Last accessed: 21/04/2022. 20

[Lehmann et al., 2010] Lehmann, A. D., Leibe, B., and Gool, L. V. (2010). Fast prism: Branch and bound hough transform for object class detection. International Journal of Computer Vision, 94:175–197. 93

[Leibe et al., 2007] Leibe, B., Leonardis, A., and Schiele, B. (2007). Robust object detection with interleaved categorization and segmentation. International Journal of Computer Vision, 77:259–289. 93

[Leutenegger et al., 2011] Leutenegger, S., Chli, M., and Siegwart, R. Y. (2011). Brisk: Binary robust invariant scalable keypoints. In 2011 International Conference on Computer Vision, pages 2548–2555. 25

[Li et al., 2020] Li, K., Rünz, M., Tang, M., Ma, L., Kong, C., Schmidt, T., Reid, I., Agapito, L., Straub, J., Lovegrove, S., and Newcombe, R. A. (2020). Frodo: From detections to 3d objects. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 14708–14717. 49

[Li et al., 2019] Li, L., Sung, M., Dubrovina, A., Yi, L., and Guibas, L. (2019). Supervised fitting of geometric primitives to 3d point clouds. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 2647–2655. 84

[Li et al., 2018] Li, Y., Bu, R., Sun, M., Wu, W., Di, X., and Chen, B. (2018). PointCNN: Convolution On $\mathcal{X}$-Transformed Points. arXiv preprint arXiv:1801.07791. 52

[Lin et al., 2017] Lin, G., Milan, A., Chunhua, S., and Reid, I. (2017). RefineNet: Multi-Path Refinement Networks for High-Resolution Semantic Segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 64

[Liu et al., 2018] Liu, S., Qi, L., Qin, H., Shi, J., and Jia, J. (2018). Path aggregation network for instance segmentation. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 8759–8768. 91

[Liu et al., 2020] Liu, S., Zhang, Y., Peng, S., Shi, B., Pollefeys, M., and Cui, Z. (2020). Dist: Rendering deep implicit signed distance function with differentiable sphere tracing. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 117

[Locatello et al., 2020] Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., and Kipf, T. (2020). Object-centric learning with slot attention. ArXiv, abs/2006.15055. 131, 132, 152

[Long et al., 2015] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 15, 20, 34

[Longuet-Higgins, 1981] Longuet-Higgins, H. C. (1981). A computer algorithm for reconstructing a scene from two projections. Nature, 293:133–135. 16

[Lorensen and Cline, 1987] Lorensen, W. E. and Cline, H. E. (1987). Marching Cubes: A high resolution 3D surface construction algorithm. In Proceedings of SIGGRAPH. 87

[Lowe, 1999] Lowe, D. (1999). Object recognition from local scale-invariant features. In Proceedings of the Seventh IEEE International Conference on Computer Vision, volume 2, pages 1150–1157 vol.2. 25

[LoweDavid, 2004] LoweDavid, G. (2004). Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision. 16

[Lowry et al., 2016] Lowry, S. M., Sünderhauf, N., Newman, P., Leonard, J. J., Cox, D. D., Corke, P., and Milford, M. (2016). Visual place recognition: A survey. IEEE Transactions on Robotics, 32:1–19. 28

[Luo et al., 2016] Luo, W., Li, Y., Urtasun, R., and Zemel, R. (2016). Understanding the effective receptive field in deep convolutional neural networks. In Advances in neural information processing systems, pages 4898–4906. 70

[Ma et al., 2017a] Ma, L., Stückler, J., Kerl, C., and Cremers, D. (2017a). Multi-view deep learning for consistent semantic mapping with rgb-d cameras. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 598–605. IEEE. 50

[Ma et al., 2017b] Ma, Y., Feng, X., Jiang, X., Xia, Z., and Peng, J. (2017b). Intrinsic image decomposition: A comprehensive review. In ICIG. 125

[Maas, 2013] Maas, A. (2013). `http://deeplearning.stanford.edu/tutorial/supervised/OptimizationStochasticGradientDescent/`. Last accessed: 07/03/2022. 39

[Mahler et al., 2017] Mahler, J., Liang, J., Niyaz, S., Laskey, M., Doan, R., Liu, X., Ojea, J. A., and Goldberg, K. (2017). Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. Proceedings of Robotics: Science and Systems (RSS). 123

[Marino et al., 2018] Marino, J., Yue, Y., and Mandt, S. (2018). Iterative amortized inference. In ICML. 131, 133, 134

[Maturana and Scherer, 2015] Maturana, D. and Scherer, S. (2015). VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition. In

Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS). 20, 50, 94

[McCormac et al., 2018] McCormac, J., Clark, R., Bloesch, M., Davison, A. J., and Leutenegger, S. (2018). Fusion++: Volumetric object-level slam. 2018 International Conference on 3D Vision (3DV), pages 32–41. 18

[McCormac et al., 2016] McCormac, J., Handa, A., Davison, A. J., and Leutenegger, S. (2016). SemanticFusion: Dense 3D semantic mapping with convolutional neural networks. In arXiv preprint:1609.05130. 15, 49

[McCormac et al., 2017a] McCormac, J., Handa, A., Davison, A. J., and Leutenegger, S. (2017a). Semanticfusion: Dense 3d semantic mapping with convolutional neural networks. 2017 IEEE International Conference on Robotics and Automation (ICRA), pages 4628–4635. 18, 149

[McCormac et al., 2017b] McCormac, J., Handa, A., Davison, A. J., and Leutenegger, S. (2017b). SemanticFusion: Dense 3D semantic mapping with convolutional neural networks. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA). 50, 56, 57, 60

[McCormac et al., 2017c] McCormac, J., Handa, A., Leutenegger, S., and Davison, A. J. (2017c). SceneNet RGB-D: Can 5M synthetic images beat generic ImageNet pre-training on indoor segmentation? In Proceedings of the International Conference on Computer Vision (ICCV). 66

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, 5:115–133. 19

[mei Zhou et al., 2008] mei Zhou, Y., yi Jiang, S., and lin Yin, M. (2008). A region-based image segmentation method with mean-shift clustering algorithm. 2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery, 2:366–370. 14

[Mescheder et al., 2019] Mescheder, L. M., Oechsle, M., Niemeyer, M., Nowozin, S., and Geiger, A. (2019). Occupancy networks: Learning 3d reconstruction in function space. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 4455–4465. 129

[Micheletti et al., 2015] Micheletti, N., Chandler, J. H., and Lane, S. N. (2015). Structure from motion (sfm) photogrammetry. 16

[Mildenhall et al., 2020] Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. (2020). NeRF: Representing scenes as neural radiance fields for view synthesis. In Proceedings of the European Conference on Computer Vision (ECCV). 82, 129, 150

[Mo et al., 2019] Mo, K., Guerrero, P., Yi, L., Su, H., Wonka, P., Mitra, N. J., and Guibas, L. J. (2019). StructureNet: Hierarchical graph networks for 3D shape generation. In SIGGRAPH Asia. 82

[Mohanty et al., 2016] Mohanty, V., Agrawal, S., Datta, S., Ghosh, A., Sharma, V. D., and Chakravarty, D. (2016). Deepvo: A deep learning approach for monocular visual odometry. ArXiv, abs/1611.06069. 18

[Montemerlo et al., 2002] Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2002). Fastslam: a factored solution to the simultaneous localization and mapping problem. In AAAI/IAAI. 17

[Mourikis and Roumeliotis, 2007] Mourikis, A. I. and Roumeliotis, S. I. (2007). A multi-state constraint kalman filter for vision-aided inertial navigation. Proceedings 2007 IEEE International Conference on Robotics and Automation, pages 3565–3572. 26

[Mur-Artal et al., 2015a] Mur-Artal, R., Montiel, J. M. M., and Tardós, J. D. (2015a). Orb-slam: A versatile and accurate monocular slam system. IEEE Transactions on Robotics, 31:1147–1163. 17

[Mur-Artal et al., 2015b] Mur-Artal, R., Montiel, J. M. M., and Tardós, J. D. (2015b). ORB-SLAM: a Versatile and Accurate Monocular SLAM System. IEEE Transactions on Robotics (T-RO), 31(5):1147–1163. 54

[Mur-Artal and Tardós, 2017] Mur-Artal, R. and Tardós, J. D. (2017). ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. IEEE Transactions on Robotics (T-RO), 33(5):1255–1262. 110

[Murphy, 2012] Murphy, K. P. (2012). Machine learning - a probabilistic perspective. In Adaptive computation and machine learning series. 32

[Narita et al., 2019] Narita, G., Seno, T., Ishikawa, T., and Kaji, Y. (2019). Panopticfusion: Online volumetric semantic mapping at the level of stuff and things. 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 4205–4212. 50

[Naseer et al., 2019] Naseer, M., Khan, S., and Porikli, F. (2019). Indoor scene understanding in 2.5/3d for autonomous agents: A survey. IEEE Access, 7:1859–1887. 14

[Newcombe et al., 2011a] Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohli, P., Shotton, J., Hodges, S., and Fitzgibbon, A. (2011a). KinectFusion: Real-time dense surface mapping and tracking. In Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR). 17, 50, 53, 81

[Newcombe et al., 2011b] Newcombe, R. A., Lovegrove, S., and Davison, A. J. (2011b). Dtam: Dense tracking and mapping in real-time. 2011 International Conference on Computer Vision, pages 2320–2327. 17

[Nicastro et al., 2019] Nicastro, A., Clark, R., and Leutenegger, S. (2019). X-Section: Cross-section prediction for enhanced RGB-D fusion. In Proceedings of the International Conference on Computer Vision (ICCV). 81, 83

[Nie et al., 2020a] Nie, Y., Han, X., Guo, S., Zheng, Y., Chang, J., and Zhang, J. J. (2020a). Total3dunderstanding: Joint layout, object pose and mesh reconstruction for indoor scenes from a single image. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 52–61. 15

[Nie et al., 2020b] Nie, Y., Han, X., Guo, S., Zheng, Y., Chang, J., and Zhang, J. J. (2020b). Total3dunderstanding: Joint layout, object pose and mesh reconstruction for indoor scenes from a single image. In IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 149

[Niemeyer and Geiger, 2021] Niemeyer, M. and Geiger, A. (2021). Giraffe: Representing scenes as compositional generative neural feature fields. 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 11448–11459. 132, 152

[Niu et al., 2018] Niu, C., Li, J. Y., and Xu, K. (2018). Im2struct: Recovering 3d shape structure from a single rgb image. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 4521–4529. 84

[Noh et al., 2015] Noh, H., Hong, S., and Han, B. (2015). Learning deconvolution network for semantic segmentation. In Computer Vision (ICCV), 2015 IEEE International Conference on. 35

[Ost et al., 2021] Ost, J., Mannan, F., Thuerey, N., Knodt, J., and Heide, F. (2021). Neural scene graphs for dynamic scenes. 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pages 2855–2864. 132, 152

[Otsu, 1979] Otsu, N. (1979). A threshold selection method from gray level histograms. IEEE Transactions on Systems, Man, and Cybernetics, 9:62–66. 14

[Özyesil et al., 2017] Özyesil, O., Voroninski, V., Basri, R., and Singer, A. (2017). A survey of structure from motion * . Acta Numerica, 26:305 – 364. 16

[Park et al., 2019a] Park, J. J., Florence, P., Straub, J., Newcombe, R., and Lovegrove, S. (2019a). DeepSDF: Learning continuous signed distance functions for shape representation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 82, 129, 150

[Park et al., 2019b] Park, T., Liu, M.-Y., Wang, T.-C., and Zhu, J. (2019b). Gaugan. ACM SIGGRAPH 2019 Real-Time Live! 20, 21, 129

[Paschalidou et al., 2020] Paschalidou, D., Gool, L. V., and Geiger, A. (2020). Learning unsupervised hierarchical part decomposition of 3d objects from a single rgb image. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 82

[Paschalidou et al., 2021a] Paschalidou, D., Katharopoulos, A., Geiger, A., and Fidler, S. (2021a). Neural parts: Learning expressive 3d shape abstractions with invertible neural networks. ArXiv, abs/2103.10429. 82

[Paschalidou et al., 2021b] Paschalidou, D., Katharopoulos, A., Geiger, A., and Fidler, S. (2021b). Neural parts: Learning expressive 3d shape abstractions with invertible neural networks. In Conference on Computer Vision and Pattern Recognition (CVPR). 84

[Paschalidou et al., 2019] Paschalidou, D., Ulusoy, A. O., and Geiger, A. (2019). Superquadrics revisited: Learning 3D shape parsing beyond cuboids. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 82, 84

[Pham et al., 2015] Pham, T. T., Reid, I. D., Latif, Y., and Gould, S. (2015). Hierarchical higher-order regression forest fields: An application to 3d indoor scene labelling. 2015 IEEE International Conference on Computer Vision (ICCV), pages 2246–2254. 17

[Pillai and Leonard, 2015] Pillai, S. and Leonard, J. J. (2015). Monocular slam supported object recognition. ArXiv, abs/1506.01732. 17

[Pinto and Gupta, 2016] Pinto, L. and Gupta, A. (2016). Supersizing self-supervision: Learning to grasp from 50K tries and 700 robot hours.

Proceedings of the IEEE International Conference on Robotics and Automation (ICRA). 123

[Pizzoli et al., 2014] Pizzoli, M., Forster, C., and Scaramuzza, D. (2014). Remode: Probabilistic, monocular dense reconstruction in real time. 2014 IEEE International Conference on Robotics and Automation (ICRA), pages 2609–2616. 17

[Pliant Energy Systems, 2022] Pliant Energy Systems (2022). Robotics - Velox robot. https://www.pliantenergy.com/robotics. Last accessed 31/01/2022. 8

[Porter, 2021] Porter, J. (2021). Boston dynamic's spot adds self-charging to live at remote sites forever. https://www.theverge.com/2021/2/2/22261932/boston-dynamics-spot-enterprise-self-charging-scout-web-based-control-software-robotic-arm. Last accessed 31/01/2022. 8

[Pritchett and Zisserman, 1998] Pritchett, P. and Zisserman, A. (1998). Matching and reconstruction from widely separated views. In Koch, R. and Gool, L. V., editors, 3D Structure from Multiple Images of Large-Scale Environments, European Workshop, SMILE'98, Freiburg, Germany, June 6-7, 1998, volume 1506 of Lecture Notes in Computer Science, pages 78–92. Springer. 16

[Pronobis and Jensfelt, 2012] Pronobis, A. and Jensfelt, P. (2012). Large-scale semantic mapping and reasoning with heterogeneous modalities. In Proceedings of the 2012 IEEE International Conference on Robotics and Automation (ICRA), Saint Paul, MN, USA. 17

[Qi et al., 2019a] Qi, C., Litany, O., He, K., and Guibas, L. (2019a). Deep hough voting for 3d object detection in point clouds. 2019 IEEE/CVF International Conference on Computer Vision (ICCV), pages 9276–9285. 93

[Qi et al., 2020] Qi, C. R., Chen, X., Litany, O., and Guibas, L. J. (2020). ImVoteNet: Boosting 3D object detection in point clouds with image votes. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 82

[Qi et al., 2019b] Qi, C. R., Litany, O., He, K., and Guibas, L. J. (2019b). Deep Hough voting for 3D object detection in point clouds. In Proceedings of the International Conference on Computer Vision (ICCV). 82

[Qi et al., 2017] Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017). Pointnet: Deep Learning on Point Sets for 3D Classification and Segmentation. In

Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 652–660. 20, 50, 52

[Raja Kumar et al., 1989] Raja Kumar, R., Tirumalai, A., and Jain, R. (1989). A nonlinear optimization algorithm for the estimation of structure and motion parameters. In Proceedings CVPR '89: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 136–143. 16

[Redmon et al., 2015] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2015). You only look once: Unified, real-time object detection. arXiv preprint arXiv:1506.02640. 40, 41, 130

[Ren et al., 2015] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In Neural Information Processing Systems (NeurIPS), pages 91–99. 40, 91, 146

[Riegler et al., 2017] Riegler, G., Ulusoy, A. O., Bischof, H., and Geiger, A. (2017). Octnetfusion: Learning depth fusion from data. In 2017 International Conference on 3D Vision (3DV), pages 57–66. IEEE. 50

[Roddick et al., 2018] Roddick, T., Kendall, A., and Cipolla, R. (2018). Orthographic feature transform for monocular 3d object detection. arXiv preprint arXiv:1811.08188. 51

[Rohmer et al., 2013] Rohmer, E., Singh, S. P., and Freese, M. (2013). V-rep: A versatile and scalable robot simulation framework. In Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS). IEEE. 123

[Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation. In Proceedings of the International Conference on Medical Image Computing and Computer Assisted Intervention (M
20, 34, 60, 64

[Rose, 2018] Rose, A. (2018). The fourth industrial revolution. `https://www.cadm.com/the-fourth-industrial-revolution/`. Last accessed: 21/01/2022. 7

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65 6:386–408. 19

[Rublee et al., 2011] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. R. (2011). Orb: An efficient alternative to sift or surf. 2011 International Conference on Computer Vision, pages 2564–2571. 16, 25, 26

[Rumelhart et al., 1986] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323:533–536. 19

[Rünz and Agapito, 2017] Rünz, M. and Agapito, L. (2017). Co-fusion: Real-time segmentation, tracking and fusion of multiple objects. In 2017 IEEE International Conference on Robotics and Automation (ICRA), pages 4471–4478. 149

[Rünz and de Agapito, 2017] Rünz, M. and de Agapito, L. (2017). Co-fusion: Real-time segmentation, tracking and fusion of multiple objects. 2017 IEEE International Conference on Robotics and Automation (ICRA), pages 4471–4478. 49

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV), 115(3):211–252. 64

[Rusu and Cousins, 2011] Rusu, R. B. and Cousins, S. (2011). 3D is here: Point Cloud Library (PCL). In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA). 110

[Salas-Moreno et al., 2013] Salas-Moreno, R. F., Newcombe, R. A., Strasdat, H., Kelly, P. H. J., and Davison, A. J. (2013). SLAM++: Simultaneous Localisation and Mapping at the Level of Objects. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 18

[Schwab, 2016] Schwab, K. (2016). The fourth industrial revolution. https://www.weforum.org/agenda/2016/01/the-fourth-industrial-revolution-what-it-means-and-how-to-respond/. Last accessed: 21/01/2022. 6

[Sharir et al., 2021] Sharir, G., Noy, A., and Zelnik-Manor, L. (2021). An image is worth 16x16 words, what is a video worth? ArXiv, abs/2103.13915. 51

[Sharma et al., 2018] Sharma, G., Goyal, R., Liu, D., Kalogerakis, E., and Maji, S. (2018). Csgnet: Neural shape parser for constructive solid geometry. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 5515–5523. 84

[Shin et al., 2019] Shin, D., Ren, Z., Sudderth, E. B., and Fowlkes, C. C. (2019). 3D scene reconstruction with multi-layer depth and epipolar transformers. In Proceedings of the International Conference on Computer Vision (ICCV). 81, 83

[Shotton et al., 2013] Shotton, J., Glocker, B., Zach, C., Izadi, S., Criminisi, A., and Fitzgibbon, A. (2013). Scene coordinate regression forests for camera relocalization in rgb-d images. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 84

[Simonyan and Zisserman, 2015] Simonyan, K. and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. In Proceedings of the International Conference on Learning Representations (ICLR). 34, 56

[Smith et al., 1986] Smith, R. C., Self, M., and Cheeseman, P. C. (1986). Estimating uncertain spatial relationships in robotics. In UAI 1986. 16

[Smith et al., 1988] Smith, R. C., Self, M., and Cheeseman, P. C. (1988). A stochastic map for uncertain spatial relationships. 16

[Sommer et al., 2020] Sommer, C., Sun, Y., Bylow, E., and Cremers, D. (2020). Primitect: Fast continuous hough voting for primitive detection. 2020 IEEE International Conference on Robotics and Automation (ICRA), pages 8404–8410. 84

[Song et al., 2017] Song, S., Yu, F., Zeng, A., Chang, A. X., Savva, M., and Funkhouser, T. (2017). Semantic scene completion from a single depth image. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 34, 81, 99, 101

[Stachniss et al., 2016] Stachniss, C., Leonard, J. J., and Thrun, S. (2016). Simultaneous localization and mapping. In Springer Handbook of Robotics, 2nd Ed. 26, 27

[Stanford University, 1979] Stanford University (1979). https://web.stanford.edu/ learnest/sail/oldcart.html. 19

[Stelzner et al., 2021] Stelzner, K., Kersting, K., and Kosiorek, A. R. (2021). Decomposing 3d scenes into objects via unsupervised volume segmentation. ArXiv, abs/2104.01148. 132

[Stückler and Behnke, 2014] Stückler, J. and Behnke, S. (2014). Multi-resolution surfel maps for efficient dense 3d modeling and tracking. Journal of Visual Communication and Image Representation, 25(1):137–147. 50

[Stutz and Geiger, 2018] Stutz, D. and Geiger, A. (2018). Learning 3D shape completion from laser scan data with weak supervision. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 81

[Sucar et al., 2021] Sucar, E., Liu, S., Ortiz, J., and Davison, A. J. (2021). imap: Implicit mapping and positioning in real-time. 2021 IEEE/CVF International Conference on Computer Vision (ICCV), pages 6209–6218. 150

[Sucar et al., 2020] Sucar, E., Wada, K., and Davison, A. (2020). Nodeslam: Neural object descriptors for multi-view shape reconstruction. 2020 International Conference on 3D Vision (3DV), pages 949–958. 116

[Sucar et al., 2016] Sucar, E., Wada, K., and Davison, A. J. (2016). NodeSLAM: Neural object descriptors for multi-view shape reconstruction. In Proceedings of the International Conference on 3D Vision (3DV). 117

[Sünderhauf et al., 2017] Sünderhauf, N., Pham, T. T., Latif, Y., Milford, M., and Reid, I. D. (2017). Meaningful maps with object-oriented semantic mapping. 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 5079–5085. 18

[Sung et al., 2015] Sung, M., Kim, V. G., Angst, R., and Guibas, L. (2015). Data-driven structural priors for shape completion. ACM Transactions on Graphics (TOG), 34(6). 82

[Svoboda, 2019] Svoboda, E. (2019). 529. Last accessed: 19/04/2022. 8

[Szeliski and Kang, 1994] Szeliski, R. and Kang, S. B. (1994). Recovering 3d shape and motion from image streams using nonlinear least squares. Journal of Visual Communication and Image Representation, 5(1):10–28. 16

[Tateno et al., 2015] Tateno, K., Tombari, F., and Navab, N. (2015). Real-time and scalable incremental segmentation on dense slam. In 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 4465–4472. IEEE. 50

[Thrun et al., 1998] Thrun, S., Burgard, W., and Fox, D. (1998). A probabilistic approach to concurrent mapping and localization for mobile robots. Autonomous Robots, 5:253–271. 17

[ThrunSebastian and MontemerloMichael, 2006] ThrunSebastian and MontemerloMichael (2006). The graph slam algorithm with applications to large-scale mapping of urban structures. The International Journal of Robotics Research. 17

[Tian et al., 2019] Tian, Y., Luo, A., Sun, X., Ellis, K., Freeman, W. T., Tenenbaum, J. B., and Wu, J. (2019). Learning to infer and execute 3d shape programs. arXiv preprint arXiv:1901.02875. 84

[TomasiCarlo and KanadeTakeo, 1992] TomasiCarlo and KanadeTakeo (1992). Shape and motion from image streams under orthography. International Journal of Computer Vision. 16

[Tuli et al., 2021] Tuli, S., Dasgupta, I., Grant, E., and Griffiths, T. (2021). Are convolutional neural networks or transformers more like human vision? ArXiv, abs/2105.07197. 51

[Tulsiani et al., 2017] Tulsiani, S., Su, H., Guibas, L. J., Efros, A. A., and Malik, J. (2017). Learning shape abstractions by assembling volumetric primitives. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 82, 84

[Valentin et al., 2013] Valentin, J., Sengupta, S., Warrell, J., Shahrokni, A., and Torr, P. (2013). Mesh Based Semantic Modelling for Indoor and Outdoor Scenes. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 17

[van Steenkiste et al., 2018] van Steenkiste, S., Chang, M., Greff, K., and Schmidhuber, J. (2018). Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. ArXiv, abs/1802.10353. 130

[Vaswani et al., 2017] Vaswani, A., Shazeer, N. M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. ArXiv, abs/1706.03762. 20

[Velizhev et al., 2012] Velizhev, A., Shapovalov, R., and Schindler, K. (2012). Implicit shape models for object detection in 3d point clouds. ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, pages 179–184. 93

[Vineet et al., 2015] Vineet, V., Mikík, O., Lidegaard, M., Nießner, M., Golodetz, S., Prisacariu, V. A., Kähler, O., Murray, D. W., Izadi, S., Pérez, P., and Torr, P. H. S. (2015). Incremental dense semantic stereo fusion for large-scale semantic scene reconstruction. 2015 IEEE International Conference on Robotics and Automation (ICRA), pages 75–82. 17

[Viswanathan, 2011] Viswanathan, D. (2011). Features from accelerated segment test (fast). 25, 26

[Wada et al., 2020] Wada, K., Sucar, E., James, S., Lenton, D., and Davison, A. J. (2020). MoreFusion: Multi-object reasoning for 6D pose estimation from volumetric fusion. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 84, 125

[Wang and Posner, 2015] Wang, D. Z. and Posner, I. (2015). Voting for voting in online point cloud object detection. In Robotics: Science and Systems. 93

[Wang et al., 2017] Wang, P.-S., Liu, Y., Guo, Y.-X., Sun, C.-Y., and Tong, X. (2017). O-cnn: Octree-based convolutional neural networks for 3d shape analysis. ACM Transactions on Graphics (TOG), 36(4):72. 50

[Wang et al., 2020a] Wang, X., Takaki, S., Yamagishi, J., King, S., and Tokuda, K. (2020a). A vector quantized variational autoencoder (vq-vae) autoregressive neural f_0 model for statistical parametric speech synthesis. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 28:157–170. 129

[Wang et al., 2020b] Wang, X., Zhang, R., Kong, T., Li, L., and Shen, C. (2020b). Solov2: Dynamic, faster and stronger. ArXiv, abs/2003.10152. 15, 41, 134, 135, 146

[Watters et al., 2019] Watters, N., Matthey, L., Burgess, C. P., and Lerchner, A. (2019). Spatial broadcast decoder: A simple architecture for learning disentangled representations in vaes. CoRR, abs/1901.07017. 129, 131, 136

[Weng et al., 1989] Weng, J., Ahuja, N., and Huang, T. (1989). Optimal motion and structure estimation. In Proceedings CVPR '89: IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pages 144–152. 16

[Whelan et al., 2015] Whelan, T., Leutenegger, S., Salas-Moreno, R. F., Glocker, B., and Davison, A. J. (2015). ElasticFusion: Dense SLAM without a pose graph. In Proceedings of Robotics: Science and Systems (RSS). 56, 128, 149

[Woodford et al., 2013] Woodford, O. J., Pham, M.-T., Maki, A., Perbet, F., and Stenger, B. (2013). Demisting the hough transform for 3d shape recognition and registration. International Journal of Computer Vision, 106:332–341. 93

[Wu et al., 2018a] Wu, B., Liu, Y., Lang, B., and Huang, L. (2018a). Dgcnn: Disordered graph convolutional neural network based on the gaussian mixture model. Neurocomputing, 321:346–356. 52

[Wu et al., 2017] Wu, J., Wang, Y., Xue, T., Sun, X., Freeman, W. T., and Tenenbaum, J. B. (2017). MarrNet: 3D shape reconstruction via 2.5D sketches. In Neural Information Processing Systems (NeurIPS). 82

[Wu et al., 2018b] Wu, J., Zhang, C., Zhang, X., Zhang, Z., Freeman, W. T., and Tenenbaum, J. B. (2018b). Learning shape priors for single-view 3D completion and reconstruction. In Proceedings of the European Conference on Computer Vision (ECCV). 83

[Wu et al., 2021] Wu, T., Pan, L., Zhang, J., Wang, T., Liu, Z., and Lin, D. (2021). Balanced chamfer distance as a comprehensive metric for point cloud completion. In NeurIPS. 38

[Wu et al., 2015] Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. (2015). 3D ShapeNets: A deep representation for volumetric shapes. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 50, 86

[Wurm et al., 2011] Wurm, K. M., Hennes, D., Holz, D., Rusu, R., Stachniss, C., Konolige, K., and Burgard, W. (2011). Hierarchies of octrees for efficient 3D mapping. In Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS). 50

[Xiang and Fox, 2017] Xiang, Y. and Fox, D. (2017). DA-RNN: Semantic mapping with data associated recurrent neural networks. In Proceedings of Robotics: Science and Systems (RSS). 50

[Xie et al., 2020] Xie, C., Xiang, Y., Mousavian, A., and Fox, D. (2020). Unseen object instance segmentation for robotic environments. arXiv preprint arXiv:2007.08073. 82, 91

[Xie et al., 2019] Xie, H., Yao, H., Sun, X., Zhou, S., and Zhang, S. (2019). Pix2Vox: Context-aware 3D reconstruction from single and multi-view images. In Proceedings of the International Conference on Computer Vision (ICCV). 81

[Xu et al., 2019a] Xu, B., Li, W., Tzoumanikas, D., Bloesch, M., Davison, A. J., and Leutenegger, S. (2019a). Mid-fusion: Octree-based object-level multi-instance dynamic slam. 2019 International Conference on Robotics and Automation (ICRA), pages 5231–5237. 49

[Xu et al., 2019b] Xu, Q., Wang, W., Ceylan, D., Mech, R., and Neumann, U. (2019b). DISN: Deep implicit surface network for high-quality single-view 3D reconstruction. In Neural Information Processing Systems (NeurIPS). 100

[Yang et al., 2019] Yang, B., Rosa, S., Markham, A., Trigoni, N., and Wen, H. (2019). Dense 3D object reconstruction from a single depth view.

IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 41(12):2820–2834. 81, 83

[Yang et al., 2019] Yang, B., Wang, J., Clark, R., Hu, Q., Wang, S., Markham, A., and Trigoni, N. (2019). Learning object bounding boxes for 3d instance segmentation on point clouds. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, Advances in Neural Information Processing Systems, volume 32. Curran Associates, Inc. 92

[Yang et al., 2017] Yang, B., Wen, H., Wang, S., Clark, R., Markham, A., and Trigoni, N. (2017). 3D object reconstruction from a single depth view with adversarial learning. In Proceedings of the International Conference on Computer Vision Workshops (ICCVW). 81, 83

[Yao et al., 2020] Yao, Y., Schertler, N., Rosales, E., Rhodin, H., Sigal, L., and Sheffer, A. (2020). Front2Back: Single view 3D shape reconstruction via front to back prediction. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 81, 83

[Yu and Koltun, 2016] Yu, F. and Koltun, V. (2016). Multi-scale context aggregation by dilated convolutions. CoRR, abs/1511.07122. 33, 50

[Zhao et al., 2017a] Zhao, H., Shi, J., Qi, X., Wang, X., and Jia, J. (2017a). Pyramid scene parsing network. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 6230–6239. 15

[Zhao et al., 2017b] Zhao, S., Song, J., and Ermon, S. (2017b). InfoVAE: Information maximizing variational autoencoders. arXiv preprint arXiv:1706.02262. 105, 144

[Zhi et al., 2019] Zhi, S., Bloesch, M., Leutenegger, S., and Davison, A. J. (2019). SceneCode: Monocular dense semantic reconstruction using learned encoded scene representations. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 150

[Zhi et al., 2021] Zhi, S., Laidlow, T., Leutenegger, S., and Davison, A. (2021). In-place scene labelling and understanding with implicit scene representation. In Proceedings of the International Conference on Computer Vision (ICCV). 151

[Zienkiewicz et al., 2016] Zienkiewicz, J., Tsiotsios, A., Davison, A. J., and Leutenegger, S. (2016). Monocular, Real-Time Surface Reconstruction using Dynamic Level of Detail. In Proceedings of the International Conference on 3D Vision (3DV). 54, 55, 56, 65

[Zou et al., 2017] Zou, C., Yumer, E., Yang, J., Ceylan, D., and Hoiem, D. (2017). 3d-prnn: Generating shape primitives with recurrent neural networks. 2017 IEEE International Conference on Computer Vision (ICCV), pages 900–909. 84