Inferring the Existence of Geometric Primitives to Represent Non-Discriminable Data

by

James A. Peraire-Bueno

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2021

© Massachusetts Institute of Technology 2021. All rights reserved.

Author .....
Department of Aeronautics and Astronautics
March 18, 2021

Certified by .....
Nicholas Roy
Professor, Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Zoltan Spakovsky
Professor, Aeronautics and Astronautics
Chair, Graduate Program Committee

Inferring the Existence of Geometric Primitives to Represent Non-Discriminable Data

by

James A. Peraire-Bueno

Submitted to the Department of Aeronautics and Astronautics
on March 18, 2021, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

In this thesis, we set out to find an algorithm that uses only geometric primitives to represent an input pointcloud. In addition to the problems faced in general primitive fitting, non-discriminable data presents additional data association challenges. We propose to address these challenges by estimating the existence rather than parameters of geometric primitives, and explore various options to do so. We first explore a sampling-based Markov-Chain Monte-Carlo approach together with a ray likelihood model. We then explore a neural network approach and finish by presenting a method to make the Chamfer distance differentiable with respect to primitive existence.

Thesis Supervisor: Nicholas Roy
Title: Professor, Aeronautics and Astronautics

Acknowledgments

Thank you to my family; my mom for being there whenever I needed anything - emotional or edible, my dad for his support and advice, and my brothers Olek and Anton for their friendship. I'd like to thank my friends and the staff at the MIT Sailing pavilion - they've been a lot of fun to be around! Thank you as well to all of the members of RRG, and in particular to Kyel Ok, Katherine Liu, and Nick Greene for all of the help and the many discussions they have provided over the past couple of years. Finally, I'd like to thank Nick Roy for his guidance, questions, suggestions, and advice.

Thank you all,
James

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As the autonomous systems we design become more complex, it becomes increasingly important for us to choose efficient representations to perceive and reason about the world. In this thesis, we study the problem of perception. In particular, we aim to find efficient representations to model given input scenes, where a *scene* refers to the sensor measurements corresponding to given environments. In this work, we present methods that use geometric primitives to represent entire scenes and in doing so address a gap in the existing literature. This chapter first supplies a motivation for our problem, and explains why existing work fails to solve it. We then address the difficulties in solving our problem, and conclude by providing an overview of the rest of the work.

1.1 Motivation

Geometric primitives are one useful representation used in robotic perception [56, 57, 86, 48, 47]. A *geometric primitive* (or just *primitive*) is a manifold embedded in euclidean space. By using generic primitives, various parts of a scene can often be represented using the same manifold representation. Due to this re-use,

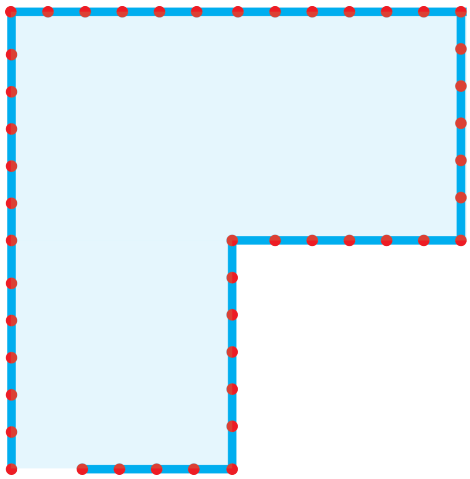


Figure 1-1: A floor plan of a simple room. Rather than use the 47 red points to reason about tangency or occupancy, we use the seven blue lines. The advantage gained in higher dimensional spaces is more extreme, as the number of points increases exponentially with each additional dimension.

representing a scene with geometric primitives is space efficient. For one, this space efficiency allows for fast and bandwidth-limited inter-agent communication. But perhaps more importantly, geometric primitives are useful as they provide a parametrized understanding of the world. A parametrized model simplifies the use of priors such as plane-object tangency constraints seen in Plane SLAM [86] or state-space features used to model constraints in work such as CBN-IRL [59]. We provide a simple two-dimensional example of primitives being used to represent a scene in figure 1-1.

Existing perception methods either fail to represent an entire scene or resort to using overparametrized representations. Algorithms like FLaME [27] and Octomap [33], which respectively store meshes and octree voxel grids, are *overparametrized*. Their internal representations store many parameters, often more than would be required to sufficiently represent a given scene for planning or navigational purposes. Overparametrized representations make it difficult to reason about the scene they represent in any capacity beyond collision avoidance. On the other

Representation	Scene reconstruction	Overparametrized
Meshes [27, 61, 71]	**Full**	Yes
Volumetric [33, 35, 80]	**Full**	Yes
Hybrid [39, 87, 86]	**Full**	Partly
CSG [43, 32, 73]	Incomplete	**No**
Missing (Our goal)	**Full**	**No**

Table 1.1: Notably missing from this table is an algorithm able to capture an entire scene without relying on overparametrized representations such as meshes, voxel grids, or point clouds.

hand, techniques such as constructive solid geometry (CSG) [43, 32, 73] provide accurate and efficient reconstructions at the cost of being prohibitively expensive to construct for large and complex scenes. Hybrid approaches [39, 87, 86] use parameter-efficient primitive representations where possible, but resort to using overparametrized pointclouds elsewhere. These details are summarized in table 1.1. Notably missing is an algorithm able to capture an entire scene without relying on any overparametrized representations.

The main goal of this work is to bridge this gap; we wish to fully represent an input scene using only geometric primitives. This process of choosing primitives to represent a scene is referred to as *fitting* primitives to a scene. In the context of this work, a scene is a pointcloud sampled from an environment. We can choose to fit primitives to a part of a scene, in which only some subset of the pointcloud is considered. Likewise, a full or complete primitive fitting is a fitting in which the entire pointcloud is considered.

1.2 Challenges

To fully understand why existing methods are unsatisfactory, it is important to make a distinction between things and stuff. Borrowing from Forsyth et al. [24], we use the term *thing* to refer to bounded objects with semantic meaning. A *thing* is an object we may care about: a car, a book, a ball, a chair. *Things* tend to have clear boundaries and "easy" data associations, at least for a human being. Given two images of a book or a bicycle, one would, with reasonable certainty, be able to identify whether they are the same book or bicycle. We refer to these correspondences between measurements and internal representations as the *data association* of our measurements. On the other hand, *stuff* is defined by textures. *Stuff* has uncertain boundaries with no distinctive shape, and the data association for stuff is generally quite difficult; it is difficult to identify two wall segments as part of the same wall, or two sections of shrubbery as part of the same shrub. But even if *stuff* is not semantically relevant to a task at hand, it must still be tracked for collision avoidance or for constraint enforcement. After all, it is no better to crash a quadrotor into a wall than it is to crash it into a car.

Existing primitive-based perception algorithms are able to use primitives to represent the things in a scene, but they are not able to fully represent the stuff. The methods that do fit primitives to stuff rely on strong assumptions about their environment [38, 87, 86]. Existing primitived-based perception methods rely on semantic information to pick out meaningful objects in the world [39, 56]. While relatively successful at mapping bounded objects with semantic meaning, these approaches are not able to fit primitives to the stuff in the scene that lacks this meaning. Most scenes contain both things and stuff, so this weakness is unacceptable if we wish to fully reconstruct input scenes.

Two challenges make it particularly difficult to fit primitives to stuff: the problem of primitive identification and the problem of data association. The first challenge of primitive identification, requires us to choose the number and types of primitives to fit to a given scene. The second challenge of data association requires us to determine the mapping between sensor measurements and the primitives which represent those measurements.

1.2.1 Primitive identification

Choosing the number and type of primitives to fit to things is relatively straightforward with the help of neural network object detections. A neural network can identify the things in the scene, and a perception system can fit either a general volumetric primitive or a primitive specific to the detected object [39, 56]. In either case, the number of primitives to fit is unambiguous: the system fits one primitive per detection.

On the other hand, neural networks only provide semantic labelings for pixels or points corresponding to stuff. The task of grouping labeled points into separate primitives is therefore left up to us. Without detections, we must estimate the number of primitives in a scene as well as their shapes.

1.2.2 Data association

Because the data associations (groupings) are not provided, we must estimate them ourselves, and this estimation is particularly difficult for stuff. For stuff, the data associations and the primitive parameters are inherently linked. Because stuff is

Figure 1-2: Two different, but equally valid ways to fit primitives to a winding road. Notice that while the two orange dots belong to different primitives on the green fitting, they belong to the same primitive on the purple fitting.

defined by unbounded textures [24], one patch of stuff can be nearly indistinguishable from a different one. As a consequence, data associations for stuff must rely heavily on metric information. The primitive parameters depend on the data associations, which, for stuff, depend on the distances between the primitives and the measurements, which in turn depend on the primitive parameters themselves.

Furthermore, stuff often has many possibly correct primitive fittings, and therefore often has many correct corresponding data associations. Consider an example in which we must fit primitives to a road, such as in figure 1-2. Both fittings use nine primitives and appear qualitatively indistinguishable. For all practical purposes, both fittings are equally correct. But we can also consider the two orange points measured on this road. In one fitting, the two points belong to the same primitive, but in the other they belong to different primitives. In the same way, both data associations are equally correct.

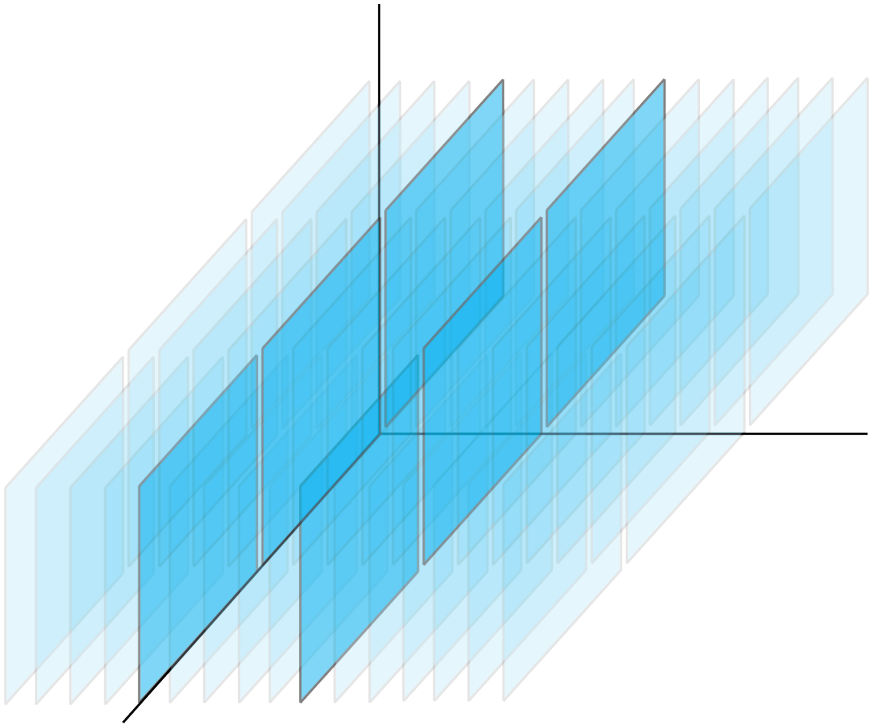


Figure 1-3: By first discretizing space, and then determining which primitive parameter combinations exist, we are able to represent an input pointcloud scene.

Existing approaches to the primitive identification and data association challenges are insufficient for our purpose of fitting primitives to stuff. Existing iterative clustering approaches to the primitive identification problem tend to rely on a good initialization and are often not flexible enough to allow for a varying number of primitives. Approaches meant to explicitly reason about the number of objects restrict the form of the clusters to simple distributions such as the Normal distribution [15]. Additionally, most prior work in data association makes assumptions about the existence of unique and correct associations [4, 11, 21].

1.3 Thesis overview

The approach at the core of this thesis is to tackle the two challenges presented by inferring over the space of primitive existence rather than by inferring over the

space of primitive parameters. To do so, we discretize the world into some finite set of primitives with fixed parameters. By fixing the primitive parameters, we remove their dependence on the data association. By then inferring which subset of these discretized primitives exist, we allow for a flexible number of primitives, sidestepping the primitive identification problem. A visualization of our approach can be seen in figure 1-3. Elaborating on the words of Vien and Toussaint [77], who write that "... the problem of reasoning with uncertainties over existence of objects is a fundamental type of uncertainty in real worlds that is not really nicely represented in practical models", we propose a question to help guide us through the remainder of this work:

How can we infer the existence of geometric primitives, and is inferring existence useful when fitting primitives to non-discriminable data?

The rest of this thesis is structured in the following manner: Chapter 2 describes the existing work related to our problem and Chapter 3 introduces key ideas and definitions central to the technical approaches. In Chapter 4, we derive from first principles a probabilistic model for primitive fitting, and show how we can use sampling techniques to optimize over it. Chapter 5 is about training a pointcloud based neural network to identify geometric primitives in a given scene. In Chapter 6, we revisit the loss function used in training the network in Chapter 5 and derive an algorithm that allows for its efficient optimization using gradient methods. Finally, we conclude this work in Chapter 7.

Chapter 2

Related Work

Recall that the goal of this work is to use only geometric primitives to fully represent an input scene. In Section 2.1 we cover alternative possible representations. In Chapter 1, we explained that there are two primary challenges when fitting primitives to stuff. One challenge is the problem of data association. Section 2.2 contains current work in that area, albeit generally applied to things rather than stuff. The other challenge is that the *number* of primitives to fit is uncertain, and must be reasoned about. We list work concerned with explicitly reasoning about a varying number of objects in Section 2.3, and cover existing primitive fitting approaches in Section 2.4. One of our approaches to primitive fitting is built on top of the PointNet++ architecture [63]. We cover various pointcloud learning works in Section 2.5. A different approach we use is based on differentiating a discrete function, and Section 2.6 contains other papers that differentiate discrete functions. Finally, Section 2.7 is about pointcloud registration: one of the evaluation metrics used in Chapter 6.

2.1 Volumetric and mesh representations

We wish to represent the world through geometric primitives, but various other approaches have also been successful for various applications. Moravec and Elfes [54] pioneered an occupancy grid model in 1985, in which space is partitioned into a grid of volumetric cubes called *voxels*. Each voxel can be marked as occupied, unoccupied, or unknown. Some work combines occupancy grids with other representations or algorithms. Thrun [74] and Konolidge et al. [40] combined occupancy grids with topological approaches by locally navigating using voxel grids but planning high-level objectives on a topological map. Wong et al. [83] introduced a method which maintained separate voxel grid and geometric primitive representations and merged them at query time. Recently, Muglikar et al. [55] used voxel grids instead of keyframes for visual SLAM.

A space-efficient occupancy grid formulation was given in Octomap [33], which used octrees to keep the voxel grid compact. Octomap was used in various robotics applications such as Pushbroom Stereo [9] or work done by Schmid et al. [67]. Various dense occupancy grid reconstructions use GPUs (Graphics Processing Units) with RGBD (RGB-Depth) cameras. KinectFusion [35] fused measurements from a low-cost Microsoft Kinect using a voxelized representation of the truncated-sign distance function (TSDF), and was extended by Chen et al. [17] to realtime performance using a memory efficient hierarchical data structure rather than a regular voxel grid. Whelan et al. [80], also using the TSDF, stored a rolling voxel grid that moved with the camera pose similar to the approach used by Chen et al., but additionally provided a method to mitigate drift and enforce global consistency.

An alternative to volumetric representations are mesh representations. Various works use meshes as lightweight reconstructions of full scenes for navigation. Pil-

lai et al. [61] used piecewise planar meshes for semi-dense reconstruction from stereo images. Various approaches [27, 71] reconstructed a scene into a lightweight mesh using monocular vision by creating and regularizing a two-dimensional Delaunay triangulation. Rosinol et al. [66] built a mesh in a VIO scene by coupling mesh regularization and state estimation to allow them to smooth the mesh over multiple frames. Not only can meshes be fit to entire scenes, they can also be fit to individual objects. ProFORMA [58] used a 3D Delaunay tetrahedralization of landmark points to create a textured surface mesh of an object. Li et al. [44] used a structured chamfer distance to train an autoencoder that fits articulated mesh models to point clouds of things.

Certain dense reconstruction approaches iteratively expanded small surface patches called surfels [29, 79, 41]. Wilkowski et al. [81] used an octree for efficient surfel map storage. A different deep learning approach by Groueix et al. [28] factorized a thing's surface representation into a template and a global feature vector and used the chamfer distance to fine tune outputs. While useful, these volumetric, mesh and surfel methods are significantly overparametrized compared to our work.

Finally, approaches such as those presented by Laidlaw et al. [43], Hoffmann et al. [32], or Thibault and Naylor [73] represented objects as sums or differences of individual polyhedra. This technique, called constructive solid geometry (CSG), has found much use in computer-aided design. While CSG provides accurate reconstructions, computational constraints preclude it from modeling entire scenes, particularly with any form of streaming measurements.

2.2 Data Association

A naïve approach to data association might be to take the data association solution implicitly given by a clustering algorithm. The k-means algorithm splits a space into clusters to minimize the intra-cluster squared Euclidean distance. In 2007, Arthur and Vassilvitskii presented k-means++ [2], a way to initialize the original k-means algorithm such that it converges faster and to better solutions. Dempster, Laird and Rubin [20] introduced the general Expectation-Maximization algorithm that solves for maximum likelihood parameters in models with latent variables or missing data. Applied to Gaussian Mixture models, the Expectation-Maximization algorithm allows for probabilistic labeling of points as well as a minimization of Mahalanobis distance rather than Euclidean distance, allowing for non-spherical clusters.

The idea of data association with measurement uncertainty originated in the target tracking literature of the mid to late 20th century. The probabilistic data association filter and joint probabilistic data association filter [7, 6] calculated probabilities that measurements were generated by particular targets when the number of targets was known and the assumed models were Gaussian or linear. Bowman et al. [12], in 2017, extended the probabilistic data association algorithm to semantic SLAM where robot and landmark poses were estimated together with the data association. These approaches work reasonably well when the data associations can be well approximated by a unimodal distribution.

Multiple hypothesis tracking (MHT) also avoids committing to hard data association labels, but explicitly reasons about multimodal distributions. With MHT, various different data association hypotheses are kept around and either improved or pruned as they become unlikely [64, 11]. Multiple-hypothesis data association

was adapted to SLAM by Cox and Leonard [18, 19]. More recent work by Doherty et al. [21] allowed for data association that is neither Gaussian nor explicitly stores multiple hypotheses by representing hypotheses as a multimodal distribution of a sensor model. Doherty et al. were able to solve the semantic SLAM graph problem with these non-Gaussian factors by using nonparametric belief propagation [70].

Work by Atanasov et al. [4] used finite random sets to represent data associations, and reduced the problem of computing association probabilities to that of computing a matrix permanent. Approximation algorithms to the problem of computing the matrix permanent give a polynomial time algorithm that allowed associations for up to two-hundred semantic objects at a time.

These various approaches attempt to solve the general data association problem, but for reasons described in Chapter 1, namely, the fact that we have no ground truth data association, are unable to solve our specific case. As we will later show in Chapter 3, our approach avoids explicitly solving the data association problem by shifting uncertainties to primitive existences rather than reasoning about uncertainties in data associations and primitive parameters.

2.3 Reasoning over varying number of entities

One of the challenges at the core of the primitive identification problem is that it is unknown how many primitives must be fit to a given scene. A weakness of the data association algorithms presented in Section 2.2 is that they assumed an *a priori* choice on the number of primitives, landmarks, or clusters. This choice led to poor results in situations where the number of primitives, landmarks, or clusters were

not known or may have been changing.

Mahler et al.'s finite set statistics [25] and the probability hypothesis density filter [50] were developed to track an unknown number of variables. Finite random sets model observations as random sets rather than random variables. Because a random set can have an arbitrary number of values, they are able to model behaviors such as missed and erroneous detections, as well as handle an unknown number of targets to track. Although initially computationally intractable, sequential Monte Carlo approximations were shown to converge to the true probability hypothesis density filter [37]. These approximations were used in various applications, such as tracking targets via radar [75], tracking feature points in a sequence of images [34] and tracking an unknown and time-varying number of speakers [82] in an audio domain. An extension to the clustering approaches in Section 2.2 was presented by Campbell et al. [15], in which they introduced a novel algorithm that did not require a prior choice on the number of clusters. By using a dependent Dirichlet process mixture model, Campbell et al. captured the behavior of a generative model that had a time-varying number of clusters. Inference on the dependent Dirichlet process mixture model was done through a novel Gibbs sampling algorithm, a special case of the more general Metropolis-Hastings algorithm. Although an impressive amount of work has been done in the area, our work avoids explicitly reasoning about varying numbers of primitives by shifting uncertainties to primitive existences.

Inference on models where computing the exact posterior is intractable, such as the ones mentioned in the previous paragraph, is generally done through either sampling methods or optimization methods. Metropolis et al. [53] presented an early work on Monte-Carlo statistical simulations. A generalization by Hastings

[31] introduced what has become known as the Metropolis-Hastings algorithm: a Markov Chain Monte Carlo method (MCMC) for sampling from a probability distribution when a proportional distribution is available but the normalizing constant is difficult to compute. One of the typical limitations of the standard Metropolis-Hastings algorithm is that the dimensionality from which the user may sample is fixed. Peter Green [26] published a method which allowed the sampler to "jump" to different dimensional sampling spaces, and as such allowed MCMC methods to be applied to problems involving Bayesian model selection. Vien and Toussaint [77] used Reversible-Jump MCMC to solve a toy problem requiring inference over the number of "bacteria" as well as their parameters. One of our approaches in Chapter 4 uses a MCMC method to sample various possible scene configurations before returning the configuration with the highest likelihood.

2.4 Primitive fitting and scene reconstruction

Recent SLAM work uses bounding geometric primitives to build a map. Kaess [38] demonstrated how to include infinite planes in a SLAM formulation. Elghor et al. [22] jointly estimated pose as well as planar landmarks. Pop-up SLAM by Yang et al. [87] assumed a structured indoor environment and "pop-up" planes were created at the ground-wall edges. Planes were then associated based on their normals and overlap. Yang and Scherer [86] extended the work to include rectangular object primitives and object-plane constraints. Nicholson et al. [56] used semantic bounding boxes as dual quadric constraints to create ellipsoidal object primitives as landmarks and Ok et al. [57] extended the algorithm to run online.

The SLAM work described thus far in this section has been concerned with fit-

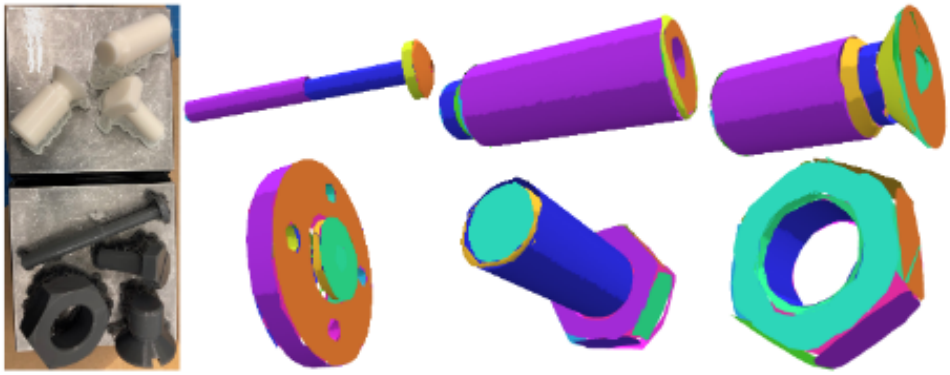


Figure 2-1: Primitive fitting results from Li et al.'s Supervised Primitive Fitting Network (SPFN) [47]. Figure reproduced with permission.

ting rough bounding primitives, such as fitting an ellipsoid to a car, rather than trying to capture the shape of the primitive being fit. The problem of precise primitive fitting has various RANSAC-based approaches [68]. A major issue with these approaches is the sensitivity of the result to the parameters which may be arbitrarily chosen. This sensitivity to parameters cause RANSAC-based methods to scale poorly, and Li et al. [48] extended these RANSAC based methods by refining extracted primitives to better fit the input. Li and Feng [45] recognized that RANSAC approaches often suffer from poor initialization and used a segmentation deep network for better initialization.

Network based approaches include ideas such as 3D-PRNN [89], a network which used cuboids to represent objects such as chairs, or work by Tulsiani et al. [76], which did the same thing but in an unsupervised manner. An approach by Paschalidou, Gool, and Geiger [60] created a parts decomposition by using a binary tree to represent simpler parts with fewer primitives than more complex parts. A different approach by Li et al. [47] presented Supervised Primitive Fitting Network (SPFN), an end-to-end neural network able to detect varying numbers of primitives to a

single object (see figure 2-1). Although impressive, SPFN had limitations. In particular, adding a new primitive to the set of four primitives that SPFN could fit required carefully writing a differentiable parameter fitting function for that geometric primitive. Additionally, SPFN was trained in a fully supervised manner, where separate primitives had to be labeled as such, and where points were labeled with their corresponding primitive. These limitations preclude SPFN from solving our problem of primitive fitting to stuff, because supervised training data is often not available for our purposes. Furthermore, any new primitive added requires a hand-written differentiable parameter fitting function – a restriction we do not require. Nevertheless, the network outperformed RANSAC approaches, even when the RANSAC approaches were allowed additional information such as point normals.

A final few ideas take different approaches. Martinović et all. [51] used a Structure from Motion (SfM) approach complemented with architectural principles to fit primitives to a facade of buildings. Eslami et al. [23] used generative models with recurrent neural networks to identify multiple objects per scene. The model chose the number of inference steps required to decompose 3D images into the various numbers of objects. Bagautdinov, Fleuret and Fua [5] used generative models but formulate the optimization variationally. And Lafarge and Mallet [42] performed primitive fitting of cities in a more straightforward segmentation and fitting approach.

2.5 Pointcloud learning

Our network in Chapter 5 is based on the PointNet line of neural networks. Qi et al. [16] presented PointNet, an architecture for deep learning on pointcloud input data. Their approach used a combination of multi-layer perceptrons and max-pooling to build a network capable of classifying a pointcloud into various classes, segment a scene into various semantic objects, or segment an object into various parts. Although it was shown that the PointNet architecture was a universal function approximator, it lacked the ability to capture local structures and therefore did not generalize well to complex scenes. Qi et al. [63] addressed this weakness by introducing multi-scale and multi-resolution grouping, a process that ran the vanilla PointNet architecture at various scales to capture both local structure as well as scene wide structure. The PointNet architectures are shown in figure 2-2. Qi et al. [62] combined deep point networks with Hough voting to create pointcloud object detection as opposed to classification.

Yi et al. presented the Generative Shape Proposal Network (GSPN) [88], which generated proposals by reconstructing shapes from noisy pointcloud input. Based on PointNet, GSPN achieved state-of-the-art performance on instance segmentation tasks. The authors suggested that its success was due to geometric understanding during object proposals. Yi et al. used multiple losses, including the chamfer loss, which is differentiable with respect to the input point coordinates. Although these networks are structured similarly to our network in Chapter 5, they output only detections, segmentations, or in the case of GSPN, primitive fittings of objects. In fact, Yi et al. wrote that the "success of GSPN largely comes from... greatly reducing proposals with low objectness". Our network addresses primitive fitting to precisely those parts of the scene GSPN does not fit to.

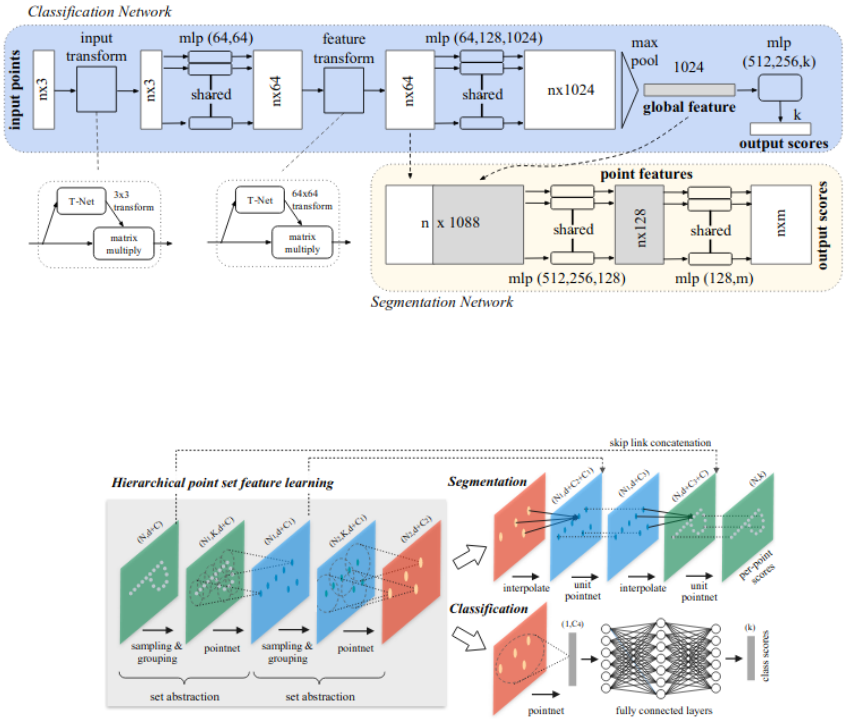


Figure 2-2: The architecture diagram for PointNet [16] (above) and PointNet++ [63] (below). Figure reproduced with permission. PointNet uses a symmetric function to introduce input order invariance. PointNet++ combines various PointNet layers to learn pointcloud information at various scales.

2.6 Differentiating discrete functions

One of our approaches, at its core, is about taking a function with discrete input and probabilistically relaxing the function to achieve differentiability. This idea of differentiating discrete functions has precedent in existing literature. Jang, Gu and Poole [36] used the Gumbel distribution to present a differentiable way of sampling from a categorical distribution by perturbing the log probabilities of sampling from any given element and finding the largest element. This trick allowed neural networks to contain categorical latent variables. More complex algorithms can also be made differentiable. DSAC, or Differentiable RANSAC [13] replaced the deterministic selection of the hypothesis for a probabilistic selection, over which the expectation could be computed. Thewlis et al. [72] rewrote the deep-matching algorithm as a convolutional neural network by representing a dynamic program as a sequence of differentiable convolutional operators. Arandjelovic et al. [1] replaced the popular "Vector of Locally Aggregated Descriptors" (VLAD) with a differentiable network layer that uses soft assignment rather than hard assignment.

Several works deal with the estimation of gradients of stochastic random variables. Bengio et al. [10] provided four strategies to estimate gradient of stochastic or non-smooth nodes in neural networks. They first considered the family of approaches using minimum variance unbiased estimators for stochastic binary neurons, then considered decomposing a stochastic binary neuron into a stochastic part and a smooth differentiable part. The third approach injected additive and multiplicative noise into a computational graph, and the fourth provided a straight-through estimate. Rezende et al. [65] developed the theory of stochastic backpropagation: backpropagation through stochastic nodes. Schulman et al. [69] provided the more general idea of a *stochastic computation graph* and used a directed acyclic graph to

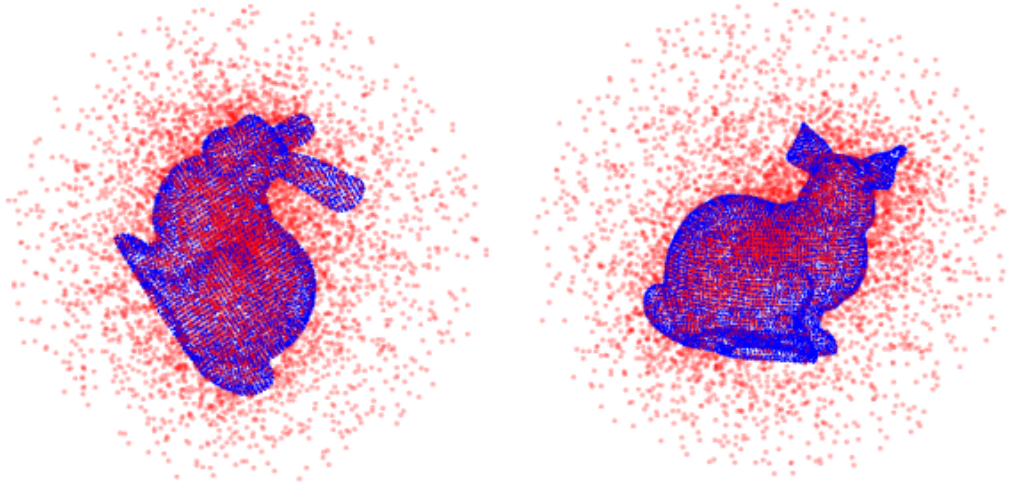


Figure 2-3: An example taken from Teaser++ [84]. Figure reproduced with permission. Although impossible for a human to verify, Teaser is able to solve for the correct pose of a bunny in a noisy pointcloud with an error of less than 4°.

compute the gradient of functions that were defined by an expectation of a collection of random variables. Although these techniques provide general tools to compute gradients of expectations, we focus on a specific expectation in Chapters 5 and 6 that allows us to write a closed form solution for the gradient.

2.7 Registration

In addition to scene reconstruction and primitive fitting, we also apply our approach to the problem of pointcloud registration, in which a rigid body transformation plus a scaling must be estimated to map one point cloud to another. ICP [3] found this transformation by alternating point correspondences with parameter estimation. Low [49] extended the algorithm from pointcloud-pointcloud matching to allow a point to plane registration. ICP, while computationally efficient, suffered from the presence of noise in the pointclouds. RANSAC methods such as

those compiled by Meer et al. [52] were fast in low noise applications with few outliers, but failed or converged more slowly on datasets with many outliers. GORE [14] was able to reduce the number of outliers and be robust in up to 95% noise, but was unable to estimate the scale and has exponential worst-case time complexity. Yang and Carlone proposed two algorithms based on formulating the registration problem as a truncated least squares problem, and using a semidefinite relaxation to efficiently solve it [85, 84]. Yang and Carlone [85] provided an algorithm robust to 99% noise, and [84] provided theoretical results on performance as well as a faster algorithm, TEASER++, that used graduated non-convexity to estimate the transformation without explicitly solving the semidefinite program. The output of the TEASER++ algorithm can be seen in figure 2-3.

Chapter 3

General overview

3.1 Preliminaries

Recall our work's guiding question:

How can we infer the existence of geometric primitives, and is inferring existence useful when fitting primitives to non-discriminable data?

To answer this question, we examine its various parts. Formally, we define a geometric primitive *type* to be a two dimensional manifold embedded in $\mathbb{R}^3$. We define

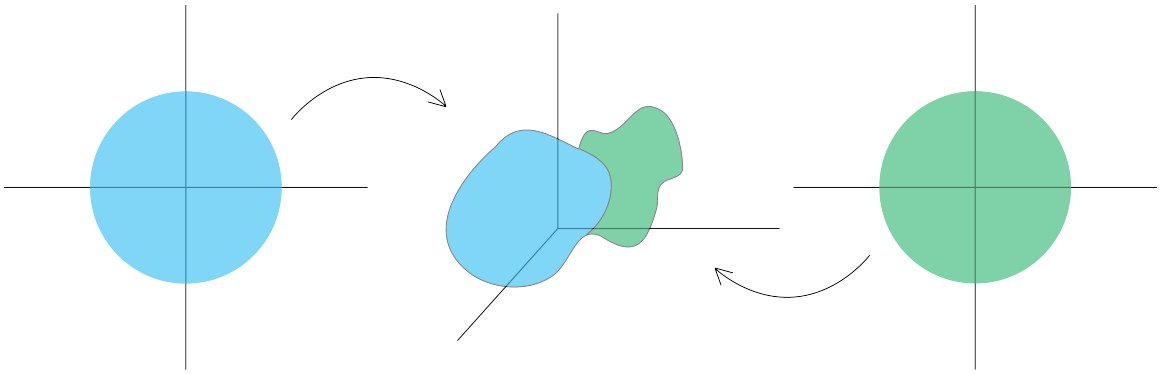


Figure 3-1: A primitive type is a manifold embedded in $\mathbb{R}^3$ given by the image of various coordinate charts from the unit disk.

this manifold via an *atlas*, or a collection of coordinate charts $\tau : B^2 \to \mathbb{R}^3$, where B^2 is the open unit disk. The geometric primitive type is the union of the image of each coordinate chart in the atlas (figure 3-1). We work with a predetermined and finite set of such manifolds, denoting the set of primitive types as $\mathcal{T}$.

A geometric primitive $\mathfrak{p}$ is then given by a primitive type η together with a rigid body transformation $\theta \in SE(3)$[1]. In particular, $\mathfrak{p} = \{\theta(x) | x \in \eta\}$, where $\theta(x)$ is the standard group action of $SE(3)$ on 3-space. We use primitives to represent occupied surfaces. Rather than store and reason about large sets of points, we are able to reason about a much smaller set of primitive types, together with well-understood rigid-body transformations. By representing surfaces and volumes as a collection of parametrized coordinate charts, we can provide fast geometric operations such as tangency and intersection queries.

In certain instances, we may wish to talk about the *existence* of a given primitive, or lack thereof. We associate with every $\mathfrak{p}_i$ a boolean variable $\phi_i \in \{0, 1\}$ to indicate its existence. A value of 1 indicates that $\mathfrak{p}_i$ exists, and a value of 0 indicates lack of existence. All together, if Ψ is the set of all primitives, Φ is the set of all such indicator variables. For notational convenience, we use $\mathcal{P} = \{\mathfrak{p}_i | \phi_i = 1\}$ to denote existing primitives.

We next examine the idea of "fitting primitives". In our work, we fix the set of primitive types $\mathcal{T}$. As a result, any set of points created by a union of our primitives is parametrized by Ψ and Φ. Fitting primitives to our data $\mathcal{D}$ then corresponds to choosing the maximum posterior estimates of these parameters. More formally, for some probability distribution p, we wish to solve:

[1] While we use SE(3), nothing in our work is specific to SE(3) and any parametrized transformation would be suitable.

$$\Psi^*, \Phi^* = \arg\max_{\Psi,\Phi} p(\Psi, \Phi|\mathcal{D}). \tag{3.1}$$

The last piece of the guiding question raises a question about the meaning of "non-discriminable". A natural approach to solving equation 3.1 is to use Bayes' theorem and to maximize some observational model $f(\mathcal{D}|\Psi, \Phi)$. For us, the key implication of our data being "non-discriminable" is that any reasonable model f will be unidentifiable. We make the assumption that the set of points that are elements of some existing primitive, $\cup\mathcal{P}$, are a *sufficient statistic* for the primitive parameters themselves. Formally, this statement means that $f(\mathcal{D}|\Psi, \Phi, \cup\mathcal{P}) = f(\mathcal{D}|\cup\mathcal{P})$. Put another way, the likelihood function does not depend on the parameters of the primitives, just the points that the primitives represent. This assumption is reasonable; our primitive parametrizations are merely a mathematical and computational tool used to represent a set of points.

This idea that $\cup\mathcal{P}$ is a sufficient statistic for the likelihood of the pointcloud has a particularly deep impact on our problem. We say data is "non-discriminable" when it corresponds to *stuff*. As described by Forsyth et al. [24], stuff "... has no specific or distinctive spatial extent or shape". Because stuff itself has no specific or distinctive spatial shape, the primitives we use to fit to stuff do not either, and in fact often have some degree of symmetry. Therefore, various different combinations of primitive parameters can result in the same aggregated set of element points $\cup\mathcal{P}$. As a consequence, any model we use that makes these assumptions is *unidentifiable*: it is impossible to learn the true values of the underlying representation used.

Our goal for this work, then, is to develop a framework that will allow us to reason about the maximization in equation 3.1. The resulting parameter values may not be unique, but uniqueness is not required when many scene representations are equally valid. In the following chapters, we propose various likelihood models and corresponding optimization techniques to find some of these maximizing parameter values.

3.2 Relaxing the exact optimization

The two common threads in our various approaches are the inference over existence and the *discretization* of primitives. Given the combination of continuous and discrete parameters, the exact solution of the mixed integer optimization problem in equation 3.1 becomes intractable. And f is overparametrized; if a given existence value is zero, the corresponding primitive's parameters are not important for scene reconstruction. Rather than attempt a full estimation, one can fix certain inputs and estimate the others. Existing clustering approaches that assume a fixed number of primitives can be seen as fixing the existence values. We propose to do the opposite: we fix the primitive parameters Ψ and optimize over the existence values Φ, as in equation 3.2.

$$\Phi^* = \arg\max_{\Phi} p(\Phi|\mathcal{D}; \Psi). \tag{3.2}$$

This alternate framing where we fix the primitive parameters might seem harder to solve than the more typical fixing of the existence values. After all, what could

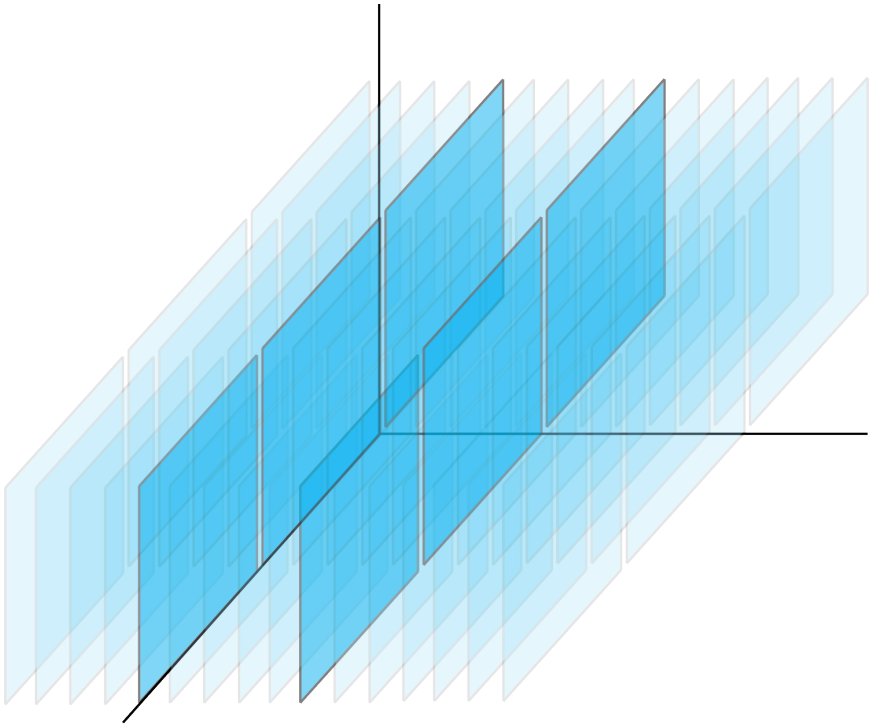


Figure 3-2: We consider only a discrete set of primitives. By selecting some subset of them, we are able to represent an input pointcloud scene.

be a continuous optimization over a relatively small number of parameter values now becomes a discrete optimization over a large number of existence values. It does, however, has a significant advantage: because the primitive parameter values are fixed, the implicit data association problem is much easier. Rather than have a cycle where the parameter values and the data associations depend on each other, we remove the dependence of the parameter values on the data associations and thus are able to compute the data associations using the fixed parameter values.

In our work, we restrict ourselves to finite-dimensional optimizations. To make our problem computationally tractable, we discretize $SE(3)$ to choose some finite set of parameters $\Theta \subset SE(3)$, and using this finite set construct a set of N primitives $\mathcal{G} = \{\mathfrak{p} | \mathfrak{p} = \theta(\eta), \theta \in \Theta, \eta \in \mathcal{T}\}$ (see figure 3-2). This construction introduces

Symbol	Definition
$\mathcal{D}$	Input pointcloud
$y \in \mathcal{D}$	Individual Pointcloud point
$\mathcal{T}$	Primitive types set
$\eta \in \mathcal{T}$	Individual primitive type
$\mathfrak{p}$	Primitive
$\Theta \subset SE(3)$	Discretized set of parameters
$\theta \in \Theta$	Primitive parameters
ϕ	Existence variable corresponding to a given primitive
Φ	Existence variables of all primitives
$\mathcal{G}$	Discretized set of primitives
$\mathcal{P}$	Set of existing primitives

Table 3.1: Various definitions used in our work.

a new type of epistemic error in any fitting: discretization error. In addition to being robust to noise, any likelihood model used must now additionally be robust to the possibility that the best primitive in Ψ may not exist in $\mathcal{G}$.

The following chapters deal with pointcloud data. In Chapter 4, we assume streaming measurements of a semantically labeled pointcloud. $\mathcal{D} = \{\mathcal{D}_t\}_{t=0}^{T}$ is the set of time-indexed pointclouds. Each pointcloud $\mathcal{D}_t = \{y_j\}_{j=0}^{J}$ is a set of J points, with each $y_j \in \mathbb{R}^3 \times S^2 \times \mathbb{N}^+$ representing a point's position, normal, and semantic class. Chapters 5 and 6 assume batch measurements with no semantic information. In the context of Chapter 5 and 6, we have $\mathcal{D} = \{y_j\}_{j=0}^{J}$, where each $y_j \in \mathbb{R}^3$ represents a point in 3-space.

For convenience, table 3.1 provides a summary of some of the definitions introduced in this chapter.

Chapter 4

Sampling methods

4.1 Intuition

In this chapter, we build a ray-based measurement model from first principles to fit geometric primitives to a hallway scene. For the purposes of exposition, suppose that we receive a single pointcloud as input, where each point contains only its own position in $\mathbb{R}^3$ – that is, for this example, our pointcloud has no information about semantic class or point normals.

The first observation we make is that each point observed provides us with two key pieces of information. A single point is evidence for occupied space at the given point, but it is also evidence that all of the space along the ray from the pointcloud sensor to the point is empty. Our measurement model considers all such rays to build a model. Using a few assumptions about noise and point generation along the ray, we specify a probabilistic model to update the likelihood of any particular primitive based on its intersections with these rays. Ray-primitive intersections far in front of detected points provide negative evidence for those primitives, while ray-primitive intersections sufficiently close to detected points

provide positive evidence.

Our probabilistic model fundamentally takes boolean inputs; it takes in some combination of existing primitives and returns a likelihood value, so we find the maximum likelihood primitive configuration by running a Markov Chain Monte-Carlo sampling scheme. We use semantic information and point normals to inform a prior that allows for more efficient sampling from the Markov Chain.

4.2 Approach

In practice, $\mathcal{D} = \{y_j\}$ is the set of time-indexed input pointclouds each containing various points $y_j \in \mathbb{R}^3 \times S^2 \times \mathbb{N}$ representing a point's location, normal, and semantic class. Our optimization problem is then to choose the set of primitives most likely to exist given the input pointcloud. Mathematically, for some probability distribution p, we wish to compute

$$\Phi^* = \underset{\Phi \in \{0,1\}^N}{\arg\max}\, p(\Phi | \mathcal{D}; \mathcal{G}) \tag{4.1}$$

where, as before $\mathcal{G}$ is the predetermined set of the N discretized primitives and Φ are the existence values for each $\mathfrak{p} \in \mathcal{G}$.

4.2.1 Measurement model

Equation 4.1 does little to shed light on the maximization problem to be solved. In particular, there is no obvious probability distribution to optimize. One possible

approach at computing the desired probability distribution is to use Bayes rule to re-write the problem:

$$\Phi^* = \underset{\Phi \in \{0,1\}^N}{\arg\max} \frac{p(\mathcal{D}|\Phi;\mathcal{G})p(\Phi;\mathcal{G})}{p(\mathcal{D})}. \tag{4.2}$$

We consider each term in this expression individually.

Point-Primitive model

We first explore a possible model for $p(\mathcal{D}|\Phi;\mathcal{G})$. Each detected point in $\mathcal{D}$ has an associated ray originating through the camera and passing through that point. Additionally, any number of existing primitives in $\mathcal{P} = \{\mathfrak{p}_i \in \mathcal{G}|\Phi_i = 1\}$ may intersect this ray (see figure 4-1). We state four assumptions used to specify this model:

1. A primitive intersecting a ray generates a single point along that ray. Multiple primitives may generate multiple points. Each point generated in this manner is generated with a Gaussian distribution around the primitive-ray intersection.

2. Additional points are generated along rays at some uniform noise rate r per unit distance.

3. The point observed by the pointcloud measurement is the nearest point to the sensor generated along a ray.

4. Points are conditionally independent given a primitive set.

This last assumption states that the Gaussian distributions from which points are sampled are independent between rays. We recognize that this is not always truly

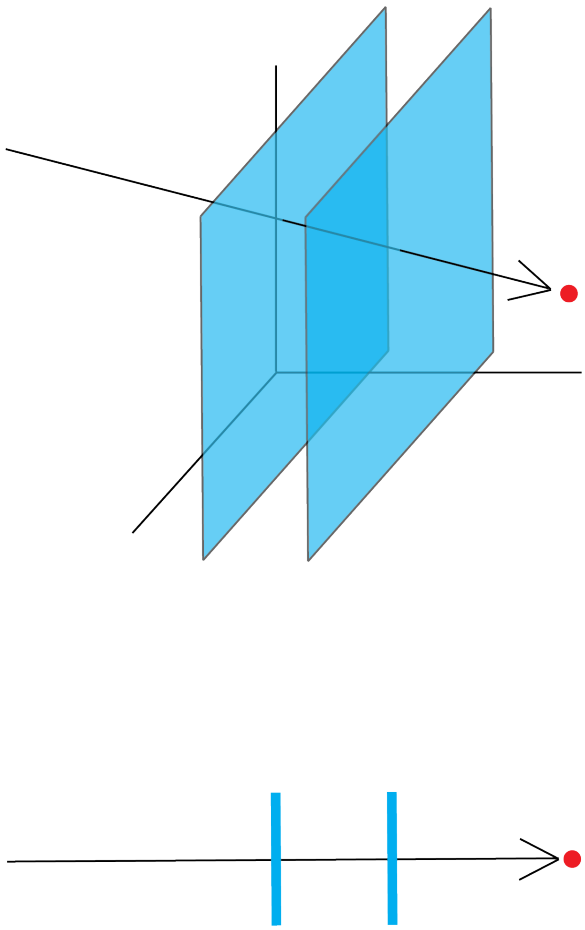


Figure 4-1: Point likelihoods are computed by projecting a ray and corresponding primitive intersections from $\mathbb{R}^3$ onto $\mathbb{R}$.

the case, as nearby rays are likely to have similar bias, but it is an assumption made to make the problem tractable. This last assumption allows us to factor

$$p(\mathcal{D}|\Phi;\mathcal{G}) = \prod_j p(y_j|\Phi;\mathcal{G}). \tag{4.3}$$

To compute any of these individual ray probabilities themselves, we first compute the distance z_j of each point y_j along its projection ray from the camera. Any primitive in $\mathfrak{p}_i \in \mathcal{P}$ that intersects this ray has a corresponding intersection distance $\mu_{j,k}$ and corresponding projected variance $\sigma^2_{j,k}$.

As an example, we first compute $p(y_j|\varnothing)$: the probability of a generated point given no existing primitive intersections along the ray to y_j. For the point y_j to be observed, it must be generated at distance z_j and there must be no points generated in front of it. As there are no primitive intersections along the ray, the only way a point can be generated is through some background noise rate r. Following the derivation for an exponential random variable, we divide z_j into n segments of length $\Delta z_j = \frac{z_j}{n}$, assume a point has a constant probability $r\Delta z_j$ of being generated in each segment, and take $n \to \infty$ to compute the probability that z_j is the observed point along the given ray.

$$p(z_j|\varnothing) = \lim_{n\to\infty} r(1 - r\Delta z_j)^n = \lim_{n\to\infty} r(1 - r\frac{z_j}{n})^n \tag{4.4}$$

$$= re^{-rz_j} \tag{4.5}$$

We next consider a scenario where there is a single primitive intersection. Using the law of total probability, we decompose the measurement function into two cases: either the measurement is noise-generated or the measurement is primitive generated.

$$p(z_j|\{\mathfrak{p}_i\}) = p(z_j|\{\mathfrak{p}_i\}, \text{noise generated})p(\text{noise generated}) \tag{4.6}$$

$$+ p(z_j|\{\mathfrak{p}_i\}, \text{generated by } \mathfrak{p}_i)p(\text{generated by } \mathfrak{p}_i) \tag{4.7}$$

The noise process has a probability r to create a noise point at z_j, and the conditional probability $p(z_j|\{\mathfrak{p}_i\}, \text{noise generated})$ is the probability that this point at z_j is the closest point on the ray to the camera.

The probability that $\mathfrak{p}_i$ does not generate a point in front of z_j is given by the inverse CDF of the normal distribution: $g(z_j; \mu_i, \sigma_i^2) = 1 - \int_{-\infty}^{z_j} N(x; \mu_i, \sigma_i^2) dx$, where μ_i and σ_i^2 are the intersection and projected variance of $\mathfrak{p}_i$ along the ray to y_j. The probability that z_j is observed, given that it is noise generated, is then the product of the probability that there are no primitive-generated occlusions $g(z_j; \mu_i, \sigma_i^2)$, the probability that there are no noise-generated occlusions e^{-rz_j}, and the probability of generation r.

Similarly, the primitive has a probability $N(z_j; \mu_i, \sigma_i)$ to generate a point at z_j, and probability e^{-rz} that there is no noise-generated point in front of it. In this case, there is no need for the inverse cumulative density function as each primitive generates only one point per intersecting ray.

Writing out the full expression, we find

$$p(z_j|\{\mathfrak{p}_i\}) = e^{-rz_j}(r \times g(z_j; \mu_i, \sigma_i^2) + N(z_j; \mu_i, \sigma_i^2)) \tag{4.8}$$

$$= e^{-rz_j} g(z_j; \mu_i, \sigma_i^2) \left(r + \frac{N(z_j; \mu_i, \sigma_i^2)}{g(z_j; \mu_i, \sigma_i^2)} \right). \tag{4.9}$$

Using the same arguments and some algebra, the measurement function involving an arbitrary number of primitives can be written:

$$p(z_j|\mathcal{P}) = e^{-rz_j} \prod_{\mathfrak{p}_i \in \mathcal{P} \text{ intersects ray}} g(z_j; \mu_i, \sigma_i^2) \times \left(r + \sum_{\mathfrak{p}_i \in \mathcal{P} \text{ intersects ray}} \frac{N(z_j; \mu_i, \sigma_i^2)}{g(z_j; \mu_i, \sigma_i^2)} \right). \tag{4.10}$$

We qualitatively examine this likelihood function in figure 4-2. The point likelihood in front of the primitive is relatively high, as the noise process might generate

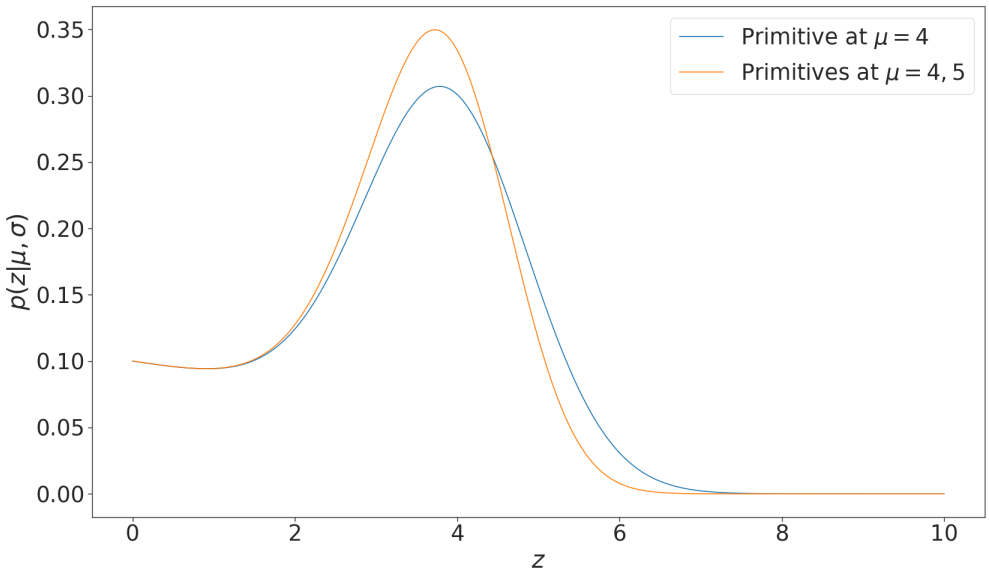


Figure 4-2: The ray-based likelihood model. One primitive intersection at $\mu = 4$, and two primitive intersections at $\mu = 4$ and 5

an occlusion. The point likelihood behind the primitive is low, as any noisy points will be occluded by primitive generated points. A strange feature of this model is that likelihoods are not always higher near where primitives exist. For example, the likelihood of a point is generated at $z = 5$ is lower when a primitive at $\mu = 5$ exists than when it does not! Although seemingly counterintuitive, this feature is a result of the occlusion calculations.

Primitive prior

We use flat circular disks as our primitives, and discretize the space such that a 1-meter radius disk in each of three orientations (face-up, face-forward, and face-left/right) occurs every 0.5 meters. We use each y_j's surface normal orientation as well as semantic class to construct the primitive prior $p(\Phi; \mathcal{G})$ over existence values. We assume a factored form $p(\Phi; \mathcal{G}) = \prod_i p(\phi_i; \mathfrak{p}_i)$. Each $p(\phi_i; \mathfrak{p}_i)$ is set in-

dividually by considering nearby points. The centers of the discretized primitives form a lattice structure, and each y_j lies inside a cell formed by this lattice structure. Each primitive has eight neighboring cells that include its center as a vertex. We set each $p(\phi_i; \mathfrak{p}_i)$ to some small constant (0.1 in practice) if there are any y_j in the neighboring cells, and set $p(\phi_i; \mathfrak{p}_i)$ to zero otherwise. To inject semantic information into the prior, we only count points that have been semantically labeled as "ground" when setting the prior for face-up disks, and only count points with a "door" or "wall" label when setting the prior for the face-forward or face-left/right disks. We use the surface normals of the "door" and "wall" points to decide whether a given point is counted for face-forward disks or face-left/right disks.

4.2.2 Combinatorial optimization and implementation details

As the evidence of the points $p(\mathcal{D})$ is intractable to compute in closed form, we use Markov Chain Monte Carlo techniques to sample from a function proportional to the posterior:

$$h(\Phi|\mathcal{D}; \mathcal{G}) \propto p(\mathcal{D}|\Phi; \mathcal{G}) p(\Phi; \mathcal{G}). \tag{4.11}$$

We use a Metropolis-Hastings sampler to sample according to equation 4.3 and equation 4.11. From now on, for notational simplicity we use the boolean vector Φ of length N to refer to a primitive set, leaving the primitive parameters implicit in the notation. All primitives being considered are assigned an entry in Φ, which is set to one to indicate the primitive exists in that scene, and set to zero otherwise.

Proposal distribution

We store only primitives with positive prior probabilities. As primitives are detected by our semantic segmentation, we check to see if that specific primitive has already been instantiated, and if it has not, it is added to a data structure that stores the most recent frame in which it was detected.

Given some scene sample Φ, we compute the next proposed primitive set according to a probabilistic transition function $Q(\Phi'|\Phi)$ by randomly flipping the existence values of the primitives in $\mathcal{G}$. More concretely, we sample a vector b of length N where the entry b_i is a Bernoulli random variable with parameter λ_i. We then set $\Phi' = \Phi$ xor b: the existence of primitives is flipped wherever $b_i = 1$. Each parameter λ_i is set such that more recently detected primitives are more likely to be flipped. By setting λ_i to decay exponentially by frame, each primitive will on expectation be flipped an equal number of times. In this manner, most computation is spent on newer primitives, but older primitives are still occasionally revisited for improvements.

Ray intersection lookups

A series of computational tricks are used to make sampling efficient. A naïve implementation might take advantage of the conditional independence between rays and compute likelihoods on a per-pointcloud, and per-ray basis, as in algorithm 1.

Evaluating equation 4.10 for a given ray depends on knowing which primitives intersect with the given ray. The naïve approach checks these intersections in a double nested loop, without taking advantage of the locality of points. We convert pointclouds to spherical coordinates centered at the pose from which they are

Algorithm 1 A naïve approach to evaluating equation 4.11

```
1: procedure ComputeIntersections(y_j, pose_t, P)
2:     intersections ← [ ]
3:     for p_i ∈ P do
4:         if p_i intersects MakeRay(pose_t, y_j) then
5:             Append p_i to intersections
6:         end if
7:     end for
8:     return intersections
9: end procedure
10: procedure Simple Probability Evaluation(D_t, pose_t, P)
11:     log likelihood ← 0
12:     for y_j ∈ D_t do
13:         intersections ← ComputeIntersections(y_j, pose_t, P)
14:         log likelihood += ComputeLogLikelihood(y_j, pose_t, intersections)
15:     end for
16:     return log likelihood
17: end procedure
```

measured to take advantage of this structure. We then store the points inside a KD tree indexed on each point's azimuthal and polar angle. Usually, polar coordinates would be unsuitable for use in a KD tree due to the discontinuity that occurs due the wrapping around of the coordinates, but our camera cannot detect points behind it. Therefore, if we set the discontinuity of the polar coordinate mapping into the KD tree behind the camera, reasonable ball queries will return the correct result. We call this polar coordinate KD tree a *RayTree*, and propose a more efficient algorithm 2. This approach avoids the double nested loop present in algorithm 1. We take advantage of the efficient ball search of the RayTree and flip the nested loop; rather than iterate over rays to compute intersections, we compute the intersections by iterating over primitives and use the results to update a table that stores each ray's intersections. This lookup table is then used to compute log likelihoods according to equation 4.10.

A final improvement can be made by recognizing that due to our proposal distri-

Algorithm 2 An approach to evaluating equation 4.11 using RayTrees

procedure RAYTREE PROBABILITY EVALUATION($\mathcal{D}_t$, pose$_t$, $\mathcal{P}$)
    RayTree $\leftarrow$ MAKERAYTREE($\mathcal{D}_t$, pose$_t$)
    Initialize RayMap
    for $\mathfrak{p}_i \in \mathcal{P}$ **do**
        $\theta \leftarrow$ COMPUTEPOLARBOUNDS($\mathfrak{p}_i$, pose$_t$)
        intersecting rays $\leftarrow$ BALLSEARCH(RayTree, θ)
        for $y_j \in$ intersecting rays **do**
            append $\mathfrak{p}_i$ to RayMap[y_j]
        end for
    end for
    log likelihood $\leftarrow$ 0
    for $y_j \in \mathcal{D}_t$ **do**
        intersections $\leftarrow$ RayMap[y_j]
        log likelihood += COMPUTELOGLIKELIHOOD(pose$_t$, y_j, intersections)
    end for
    return log likelihood
end procedure

bution, Φ' tends to be close to Φ. Therefore, it suffices to evaluate only the rays which intersect with primitives that were flipped in the proposal update: Φ xor Φ' to compute the likelihood difference. This incremental evaluation decreases the scene evaluation time by an order of magnitude and can be found in algorithm 3.

4.3 Results

We construct two toy examples to demonstrate the capabilities of the measurement model. Because of the independence assumption between rays, the likelihood of a scene is a sum of the likelihood of the rays in the scene. Our examples consist of six primitives arranged in a row; each behind the previous one. We set our camera at the origin, and the primitives are equally spaced such that the nearest one is 7 meters away, and the furthest one is 12 meters away. Points are then sampled from the surface of the primitive at 10 meters. One example consists of planar primitives, and one example consists of cylindrical primitives. A side-view diagram of

Algorithm 3 An approach to incrementally evaluating equation 4.11 incrementally

procedure RAYTREE PROBABILITY EVALUATION($\mathcal{D}_t$, pose_t, old scene score, new primitives, removed primitives)
    RayTree $\leftarrow$ MAKERAYTREE($\mathcal{D}_t$, pose_t)
    Initialize RayMap, NewRayMap, RemovedRayMap
    for $\mathfrak{p}_i \in \mathcal{P}$ **do**
        $\theta \leftarrow$ COMPUTEPOLARBOUNDS($\mathfrak{p}_i$, pose_t)
        intersecting rays $\leftarrow$ BALLSEARCH(RayTree, θ)
        for $y_j \in$ intersecting rays **do**
            append $\mathfrak{p}_i$ to RayMap[y_j]
        end for
    end for
    for $\mathfrak{p}_i \in$ new primitives **do**
        $\theta \leftarrow$ COMPUTEPOLARBOUNDS($\mathfrak{p}_i$, pose_t)
        intersecting rays $\leftarrow$ BALLSEARCH(RayTree, θ)
        for $y_j \in$ intersecting rays **do**
            append $\mathfrak{p}_i$ to NewRayMap[y_j]
        end for
    end for
    for $\mathfrak{p}_i \in$ removed primitives **do**
        $\theta \leftarrow$ COMPUTEPOLARBOUNDS($\mathfrak{p}_i$, pose_t)
        intersecting rays $\leftarrow$ BALLSEARCH(RayTree, θ)
        for $y_j \in$ intersecting rays **do**
            append $\mathfrak{p}_i$ to RemovedRayMap[y_j]
        end for
    end for
    log likelihood $\leftarrow$ old scene score
                                                ▷ Iterate only over the modified rays
    for $y_j \in$ AddedRayMap $\cup$ RemovedRayMap **do**
        intersections $\leftarrow$ RayMap[y_j]
        old score += COMPUTELOGLIKELIHOOD(pose_t, y_j, intersections)
        intersections $\leftarrow$ RayMap[y_j] + NewRayMap[y_j] - RemovedRayMap[y_j]
        new score += COMPUTELOGLIKELIHOOD(pose_t, y_j, intersections)
        log likelihood += new score - old score
    end for
    return log likelihood
end procedure

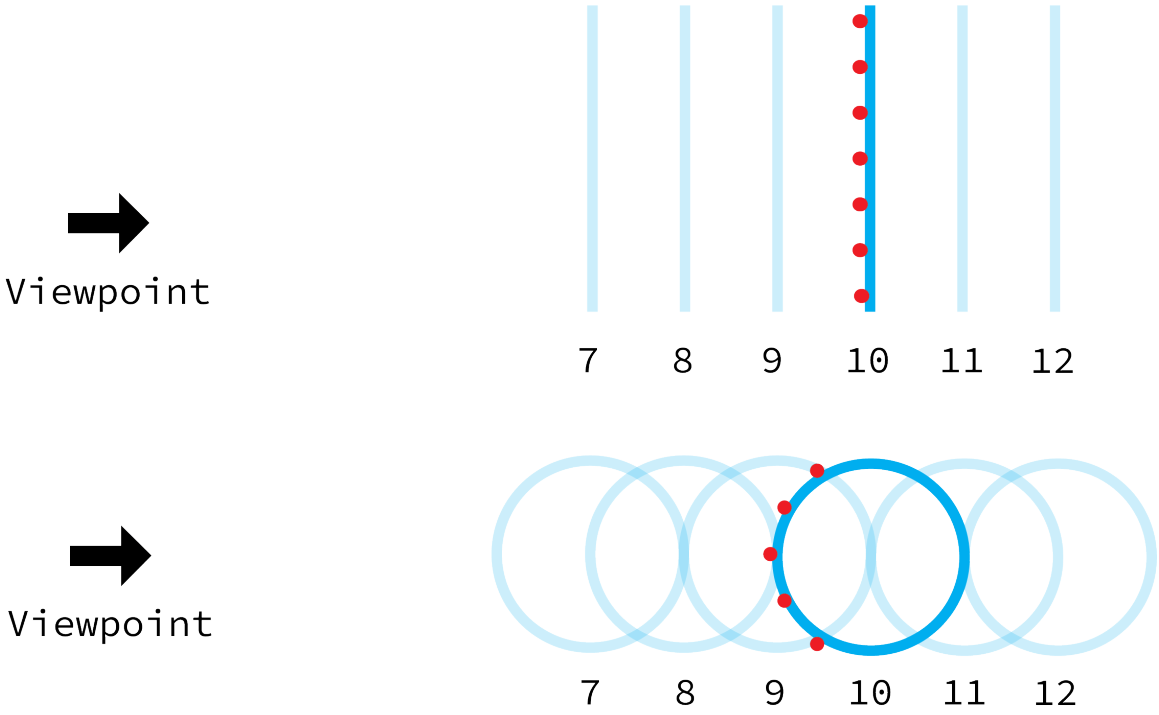


Figure 4-3: We consider a toy setup where six primitives are arranged in a row, and points are sampled from the primitive at 10 meters. On the top, we consider planar primitives, and on the bottom we consider cylindrical primitives (side view).

the setup can be seen in 4-3.

We evaluate all possible scenes, but plot the likelihood of single and double primitive scenes in the heatmap shown in figure 4-4, in which darker squares are more likely. The square on the diagonal at index i corresponds to the negative log likelihood of the single primitive at i meters, and the square on the off diagonal at position i, k corresponds to the negative log likelihood of two primitives: one at i and one at k. Unsurprisingly, the most likely configurations are those configurations whose nearest primitive is the primitive from which we sampled our pointcloud. Any configurations containing primitives in front of the sampled pointcloud appear highly unlikely.

An interesting effect of our measurement model is that scene likelihoods are largely

determined by the nearest primitive along each ray. This property is a result of our third assumption: the assumption that the point observed by a pointcloud measurement is the nearest point generated along its ray. The assumption is meant to model occlusions, and we see its effect in figure 4-4: all of the scenes with the nearest primitive at 10 meters are very close in likelihood, regardless of where the furthest primitive is. This is reasonable, as a measurement of a primitive at 10 meters gives us little information as to the existence of any primitives behind it. We also note that the two heatmaps in figure 4-4 appear visually identical. This similarity occurs because the two scenes are similar once the points and intersections are projected onto their corresponding rays; the z_j are all exactly at the primitive intersections $\mu_{i,j}$. This behavior is more general: any example where all of the points y_j are sampled from the surface of primitives in $\mathcal{G}$ are correctly detected by our likelihood function.

Ultimately, the method does not work particularly well when applied to real-world datasets. Representative results can be found in figure 4-5. We find that there seem to be extra primitives fit in certain places, and missing primitives in other places we would have expected them. These disappointing results can be attributed to two design choices: the choice of measurement model, and the sampling-based maximum likelihood estimation.

Issues with the measurement model

Our measurement model has two primary issues, one caused by the ray-based point generation and the other by the discretization of primitives. The first issue is the inability of the model to handle non-depth noise. We only model noise along each ray, providing no way to reason about points that might be noisy along a

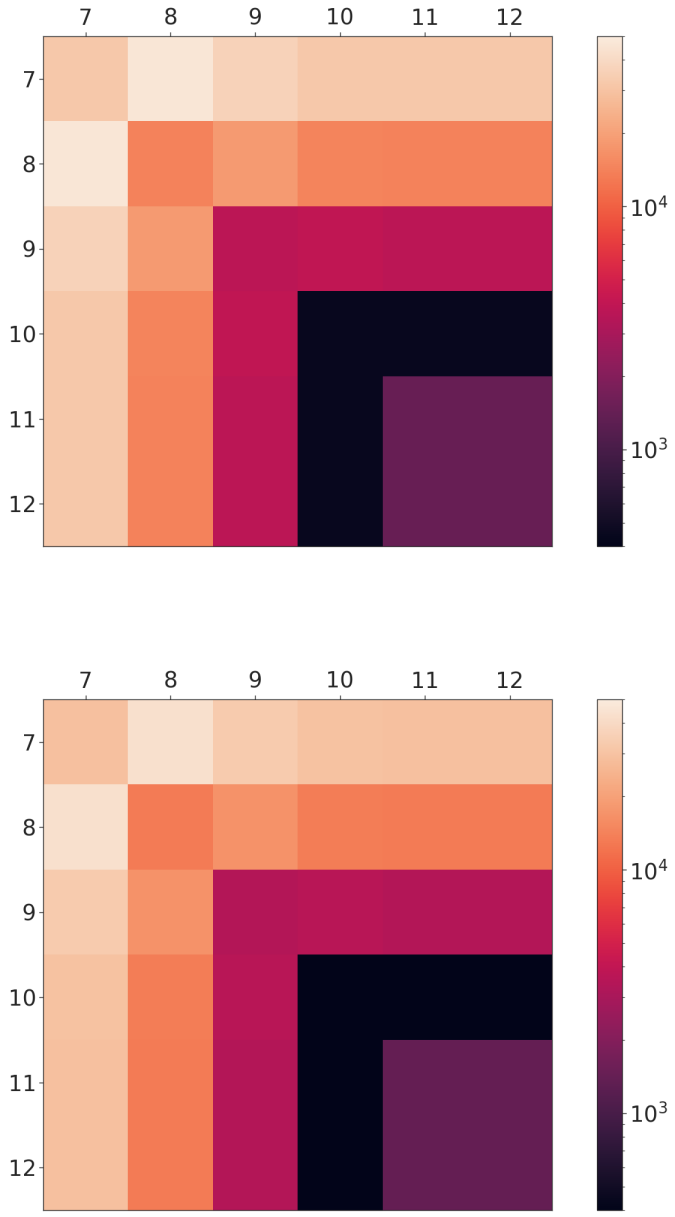
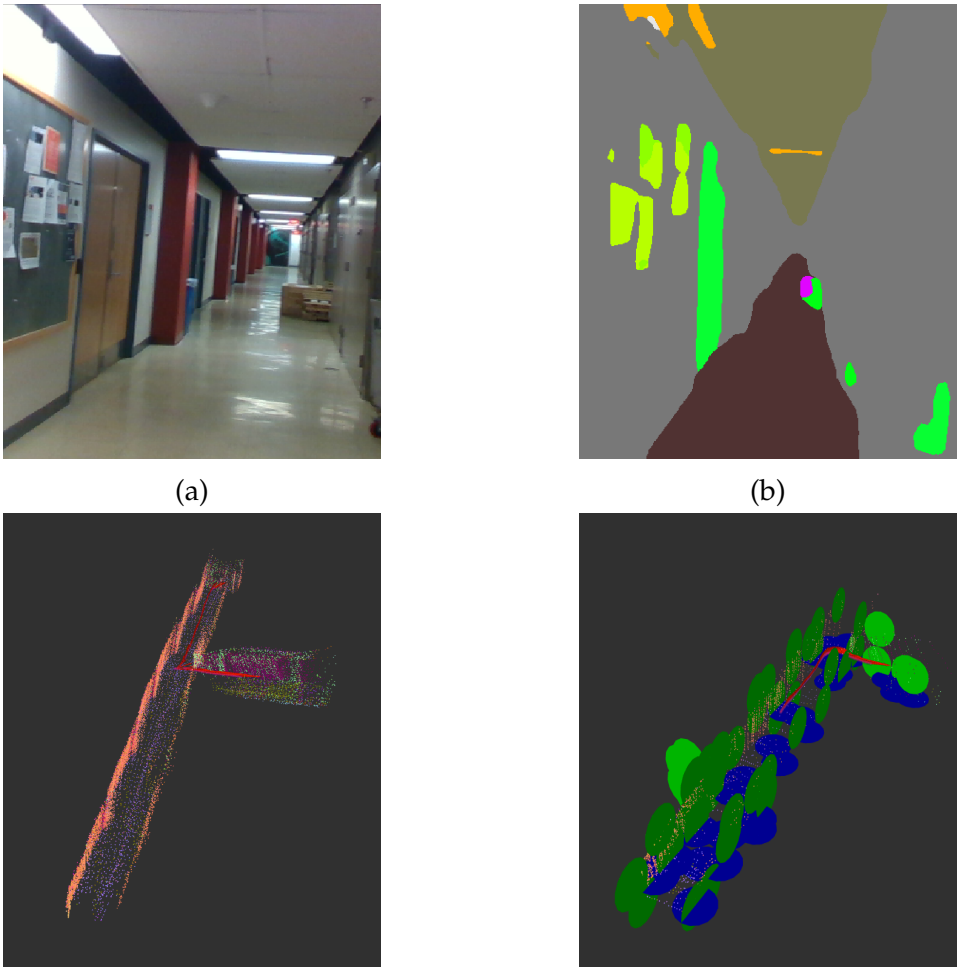


Figure 4-4: Heatmaps of the negative loglikelihood of one and two primitive combinations for the toy examples in figure 4-3. Diagonal squares correspond to single-primitives, and off-diagonal terms correspond to a two primitive combination. Darker colors are more likely.



(a) (b) (c) (d)

Figure 4-5: We show a sample input image (a) and corresponding semantic segmentation (b). We merge the pointclouds from many frames to form the full input pointcloud, together with surface normals (c). Running our algorithm results in a primitive fit (d).

different axis. The second issue is caused by the assumption that points on each ray are generated independently from each other, given the existing primitives. Because of the error introduced by the discretization, our primitive models are sensitive to point-primitive alignment. If our discretization is dense enough that the points are close enough to a possible primitive, we are able to accurately fit primitives to a scene as in figure 4-3. But if the alignment is poor, and the scene is not well represented by any of the possible discretized primitives, the error is consistent between rays. As a result of these two issues, even scenes we are able to solve exactly are poorly approximated by our maximizing subset.

We set up a toy example in figure 4-6 to demonstrate the failure of our model in capturing a simple scene if the point-primitive alignment is bad. The red points are sampled from a plane to form the input pointcloud, and two possible primitives are considered to represent this pointcloud. We evaluate our model on all four combinations of primitive existences and report the resulting likelihoods in table 4.1. Because the front primitive has many pointcloud points directly behind it, the likelihood given to any scene which includes it is zero. This is partly due to the assumption that points generated along rays are independent; while one point behind an existing primitive may be unlikely, the occurrence of many points behind the primitive make its existence in our model near-impossible, even if their existence is well-explained by epistemic error. For similar reasons, the existence of the primitive in the back has no positive evidence, and the model therefore implicitly treats the pointcloud as noise-generated.

Configuration	Loglikelihood
Empty	-0.69 (50%)
Front primitive only	-358.2 (0%)
Back primitive only	-0.69 (50%)
Both primitives	-358.2 (0%)

Table 4.1: Loglikelihoods of various primitive configurations in our toy example

Figure 4-6: A toy example to show the weaknesses of our measurement model. The red points are sampled from a plane to form an input pointcloud, and the planar primitives are the two possible primitives are considered in $\mathcal{G}$. Note that the plane used to sample the red points is not considered in $\mathcal{G}$.

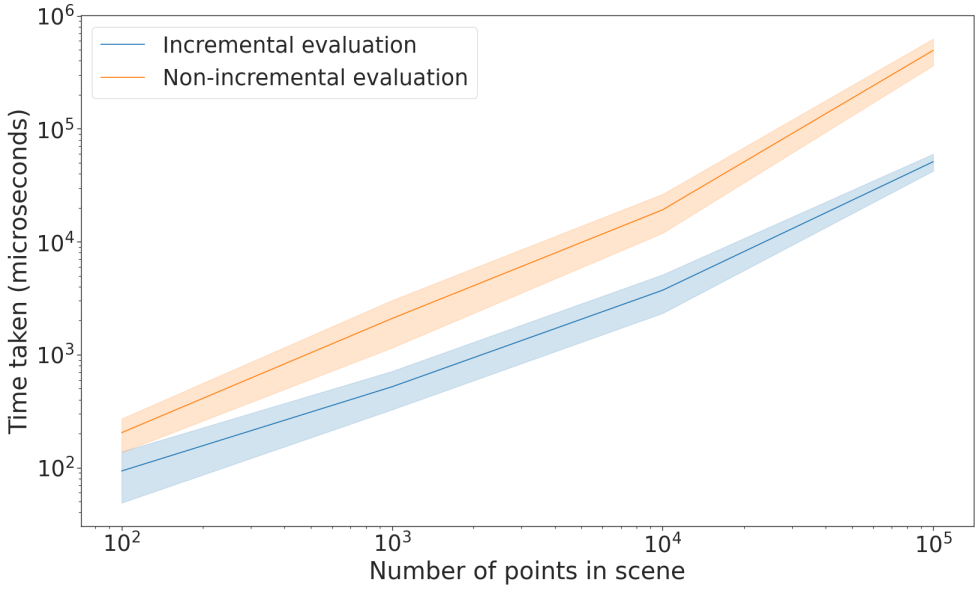


Figure 4-7: We run scene evaluation algorithms 2 and 3. The solid line is the mean of the time taken per evaluation, and the shaded regions represent times within a standard deviation of the mean. The non-incremental evaluation tends to be on average about an order of magnitude slower than the incremental evaluation.

Sampling based maximum likelihood estimation

We timed scene evaluations using both the standard and incremental algorithms found in algorithms 2 and 3. The results of the timing are found in figure 4-7. We found that the incremental algorithm is able to evaluate scenes roughly an order of magnitude faster than the non-incremental scene. The scene shown in figure 4-5 consists of about 20000 points and considers 500 possible primitives to fit to the scene. A scene evaluation and Metropolis-Hastings step takes 50 ms to compute from scratch, or 2 ms to compute incrementally on an Intel CORE i7. All things considered, we run 1000 Metropolis-Hastings sampling steps plus some appropriate burn-in steps before returning a maximum likelihood estimate. Even if the measurement model were able to accurately capture the desired primitive fitting behavior, the randomized sampling nature of the approach means it is possible that good primitive combinations are never sampled.

For these two reasons, we determine that a sampling-based approach together with our ray-based model is impractical for primitive map building. These failures, however, offer some lessons to be learned. The first is that a measurement model which better reflects good primitive fits must be developed. The second is that a pure sampling-based approach may fail to return a good map. We learn from these lessons in Chapter 5 and 6.

Chapter 5

Learning primitive existence

Some of the issues we found in Chapter 4 were due to a poor measurement model. We explained that even the global optimum of the ray-based measurement function leads to qualitatively bad scene reconstructions. In this section, we attempt to have a neural network learn better scene reconstructions. In our particular case, we wish to use a neural network to fit primitives to an input pointcloud. As described in Chapter 3 and done in Chapter 4, we once again discretize the primitive parameter space and use a neural network to infer primitive existence. We describe this process in more detail in the following sections.

A neural network is fundamentally a function approximator. In a supervised setting, the network requires training examples of the function to approximate. But these examples are hard to come by. Recall that the problem of data association to stuff has no single correct solution, and that this ambiguity is even more general than simply data association: in many cases, there may be multiple equally valid primitive reconstructions as well as solutions to the data association problem.

5.1 Training a general neural network

In its most general form, a neural network can be seen as a black-box function taking some input and resulting in some output. This black-box function has some number of internal parameters which specify a nonlinear transformation from the input to the output. Although it is difficult to set parameter values which may cause a neural network to approximate a given nonlinear transformation *a priori*, we can use two algorithms, *backpropagation* and *stochastic gradient descent* to iteratively and incrementally update the parameter values to better approximate the given nonlinear transformation.

For the purposes of this example, suppose we are given K supervised training examples $(X, y)_k$, where $X_k \in \mathbb{R}^n$ is the example input to f, and $y_k \in \mathbb{N}$ is the supervised label. In the context of image classification, X_k might be the pixel values of an image, and y_k might be a label describing the contents of the image. Our goal is to find the internal parameters γ of our neural network function $f(\cdot; \gamma)$ that minimizes an expectation $E[L(y, f(X; \gamma))]$ over our examples, where L is some loss function that specifies how close the network output value is to the true output. Because y_k and X_k are given, the minimization of L occurs over γ.

Minimizing the expected loss over our examples might be done by using gradient descent. In the case where we optimize over a small number of training examples, one can directly compute the expectation $E[L(y, f(X; \gamma))]$ over our examples, and therefore compute the gradient $\frac{dE[L]}{d\gamma}$ and optimal iterative update: $\gamma = \gamma - \eta \frac{dE[L]}{d\gamma}$ for some learning rate η. In practice, a large number of training examples makes this expectation impractical to compute, so an approximate algorithm is used. Stochastic gradient descent (SGD) approximates the gradient for this expectation by sampling the gradient for specific training examples. After sampling a training

example k, SGD applies the partial update $\gamma = \gamma - \eta \frac{L(y_k, f(X_k;\gamma))}{d\gamma}$. By iteratively computing gradients for stochastically chosen training examples, SGD optimizes the expected loss in a memory and time efficient manner.

A subtle detail we passed over in our summary of SGD was the computation of the gradient $\frac{dL}{d\gamma}$. In practice, the neural network f is defined by composing a sequence of simpler nonlinear functions $f = f_\ell \circ \cdots \circ f_1$. The gradient $\frac{dL}{d\gamma}$ can then be computed via the chain rule $\frac{dL}{d\gamma} = \frac{dL}{df}\frac{df}{d\gamma}$, where $\frac{df}{d\gamma}$ is also computed via the chain rule:

$$\frac{df}{d\gamma} = \frac{\partial f_\ell}{\partial \gamma} + \frac{\partial f_\ell}{\partial f_{\ell-1}}\frac{df_{\ell-1}}{d\gamma} \tag{5.1}$$

$$= \frac{\partial f_\ell}{\partial \gamma} + \frac{\partial f_\ell}{\partial f_{\ell-1}}\left(\frac{\partial f_{\ell-1}}{\partial \gamma} + \frac{\partial f_{\ell-1}}{\partial f_{\ell-2}}\frac{df_{\ell-2}}{d\gamma}\right) \tag{5.2}$$

$$= \cdots \tag{5.3}$$

Here, d indicates a total derivative, and ∂ indicates a partial derivative. The backpropagation algorithm provides an efficient evaluation order for this chain rule calculation to avoid the repeated computation of intermediate terms. A consequence of this training is that every f_k and L used in computing the loss function must be differentiable; if any steps are nondifferentiable, then $\frac{dL}{d\gamma}$ is not well defined, and stochastic gradient descent cannot be used to optimize the network parameters.

There are two particular challenges that we would face if we were to try to train a neural network to identify primitives in a supervised manner as described. The first challenge would be in creating a loss function in primitive-space; it is difficult to quantify what it means for two geometric primitive scenes to be similar or different. The second challenge would be the existence of training labels. Because there

is no single correct primitive fitting, training labels for any data we may want to use are difficult to find. To address both problems, we propose that self-supervised training is both the most elegant and the most practical solution. Self-supervision is the idea of training a neural network by using training labels easily computed from the input data. In the context of our problem, the self supervised loss function works as follows: first, each primitive in the network output is converted into some finite set of points in $\mathbb{R}^3$. Next, we use an existing and well-understood pointcloud-distance function, such as the chamfer distance[1] [8] given in equation 5.4, to measure the distance between the network output pointcloud, and the input pointcloud. This strategy addresses both challenges: by using an existing pointcloud distance function, there is no need to create an arbitrary primitive distance function, and by converting each primitive scene back into a pointcloud, we compare our output directly against our input data, so there is no need for any training data labels. This idea is consistent with our exposition in Chapter 3. By projecting each chosen primitive back into a pointcloud, certain superflous information differentiating equally valid reconstructions can be discarded. This projection can be thought of as computing the sufficient statistic described in Chapter 3 to make the likelihood function identifiable.

For two finite pointclouds $S_1, S_2 \subset \mathbb{R}^3$, we define the chamfer distance $d_{\text{chamfer}}(S_1, S_2)$

$$d_{\text{chamfer}}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|^2 + \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|^2 . \tag{5.4}$$

More concretely, as before, we consider a set of primitive types $\mathcal{T}$ together with a discrete and finite parameter set to form the set of discretized primitives $\mathcal{G}$. Each primitive $\mathfrak{p}_i \in \mathcal{G}$ has a corresponding finite set $\sigma_i \subset \mathbb{R}^3$ of points sampled from its

[1]Note that the chamfer distance function is not a true metric, as the triangle inequality does not hold

surface. Our neural network output is a vector Φ. The primitives in $\mathcal{G}$ are ordered and the existence of $\mathfrak{p}_i \in \mathcal{G}$ corresponds to the output ϕ_i of the network at index i. The proposed network therefore takes in a pointcloud as input and generates a vector of existence values for a predetermined set of primitives. During training, these existence values are used to project the existing primitives into a generated pointcloud: $\cup\{\sigma_i|\phi_i = 1\}$. The loss function computes the chamfer distance between the input pointcloud and the generated pointcloud.

The chamfer distance is computed by summing over point-point correspondence distances. Good fits are encouraged by the minimum terms in the distance function: an additional primitive that lowers minimum distances will likely decrease the total distance. On the other hand, adding additional existing primitives once the minimum distances are already low is unlikely to lower the minimum distances by enough to overcome the additional terms in the sum. In other words, although the chamfer distance does not explicitly enforce mutual exclusion, as multiple primitives may, in theory, be used to describe the same part of a scene, primitives are added to ensure that the points in the input pointcloud $\mathcal{D}$ have primitive points near them, while ensuring that any unneeded primitives are not added.

5.2 Probabilistic chamfer distance

In our case, the self-supervised approach is not as straightforward as described. Existence is binary: a geometric primitive either exists or it does not. While the chamfer loss described in equation 5.4 is differentiable with respect to point positions, it is not differentiable with respect to point inclusions or exclusions which

would result from the binary existence output[2]. And as mentioned in Section 5.1, if any part of the loss computation is nondifferentiable, the network cannot be trained. Approximating a discrete difference is a combinatorial problem in the number of variables, so to have any hope of training a neural network with the chamfer loss, the process must be made differentiable.

5.2.1 Smoothing existence values and an expected chamfer distance

We wish to make the chamfer loss differentiable. To this end, we first relax the domain of the function. Rather than take boolean existence values, we propose that our modified chamfer loss take a vector of floats in $[0, 1]$ as input. While using a vector of floats may seem strange for the task of primitive classification, we offer the following interpretation: a value of p_i at index i indicates that $\mathfrak{p}_i$ exists with probability p_i.

Assigning to each primitive a probability also leads to a reasonable map from primitive space to pointcloud space for use in computing the chamfer loss. Recall that each primitive $\mathfrak{p}_i$ has a corresponding set of points σ_i sampled from its surface. Rather than including $\mathfrak{p}_i$'s entire corresponding pointset σ_i when $\phi_i = 1$, each point in σ_i is included with probability p_i. To denote this random sampling, we define a random variable $Q_i \subseteq \sigma_i$ with a corresponding probability distribution $\pi_i(Q_i) = p_i^{|Q_i|}(1 - p_i)^{|\sigma_i| - |Q_i|}$.

Although smoothing the existence values seems to be a necessary first step, it is not

[2]Using a Softmax type loss similar to those used in classification networks would require labelings of the correct primitives in a scene.

sufficient, as the approach described is still non-differentiable in several places. In particular, the random variables Q_i are both discrete and random. The discreteness makes it impossible to differentiate, and the randomness makes it difficult to optimize as the gradient depends on the results of the random sampling, which may be different between function calls.

We propose to address both problems by taking an expectation of the chamfer distance over different samplings. Intuitively, it seems reasonable to claim that removing a point or adding a point is unlikely to vastly change the distance between two pointclouds, and increasing or decreasing projection probabilities does, on expectation, exactly this. We use this idea to smooth the random sampling procedure.

Recall that the definition of the chamfer distance between two point sets S_1 and S_2 is given by

$$d_{\text{chamfer}}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} \|x - y\|^2 + \sum_{y \in S_2} \min_{x \in S_1} \|x - y\|^2 . \tag{5.5}$$

We specialize the notation for our case. We use $Q = \cup_i Q_i$ to denote the set of sampled points from the entire primitive scene. We use π as the probability distribution of Q, assuming independence between the Q_i. We then set $S_1 = Q$ and S_2 becomes the input pointcloud, $\mathcal{D}$. All together, we wish to compute

$$\mathbb{E}_\pi \left[d_{\text{chamfer}}(Q, \mathcal{D}) \right] = \mathbb{E}_\pi \left[\sum_{x \in Q} \min_{y \in \mathcal{D}} \|x - y\|^2 + \sum_{y \in \mathcal{D}} \min_{x \in Q} \|x - y\|^2 \right] . \tag{5.6}$$

We refer to the loss in equation 5.6 as the *probabilistic chamfer* distance[3]. Using the

[3]There is an existing probabilistic chamfer loss in work done by Li and Lee [46]. The two losses

linearity of expectation, we break this expectation into two terms and compute them separately.

5.2.2 Probabilistic chamfer primitive loss

The first term we call the probabilistic primitive loss. To compute this term, we split the sampled points into groups based on which set they were sampled from, and once again use the linearity of expectation to compute the loss for each primitive individually.

$$\mathbb{E}_{\pi}\left[\sum_{x\in Q}\min_{y\in\mathcal{D}}\|x-y\|^2\right]=\sum_i\mathbb{E}_{\pi_i}\left[\sum_{x\in Q_i}\min_{y\in\mathcal{D}}\|x-y\|^2\right]. \tag{5.7}$$

To compute each individual primitive's loss, we expand the definition of conditional expectation and use the indicator function I.

$$\mathbb{E}_{\pi_i}\left[\sum_{x\in Q_i}\min_{y\in\mathcal{D}}\|x-y\|^2\right]=\sum_{Q_i}\pi_i(Q_i)\left(\sum_{x\in\sigma_i}I(x\in Q_i)\min_{y\in\mathcal{D}}\|x-y\|^2\right) \tag{5.8}$$

$$=\sum_{x\in\sigma_i}\min_{y\in\mathcal{D}}\|x-y\|^2\left(\sum_{Q_i}\pi(Q_i)I(x\in Q_i)\right). \tag{5.9}$$

As each point is sampled independently in π, the inside nested sum is equal to the probability that a given set $\pi(\sigma_i, p_i)$ contains the point $x \in \sigma_i$. This probability is precisely p_i. So all together, the probabilistic primitive loss can be written:

are distinct; our probabilistic chamfer distance probabilistically includes points in S_1, while their probabilistic chamfer loss forms a probability distribution based on the distances between corresponding points.

$$\mathbb{E}_\pi \left[\sum_{x \in Q} \min_{y \in \mathcal{D}} \|x - y\|^2 \right] = \sum_i p_i \sum_{x \in \sigma_i} \min_{y \in \mathcal{D}} \|x - y\|^2 . \tag{5.10}$$

As such, the primitive point loss is a linear function in the primitive probabilities.

5.3 Probabilistic chamfer point loss

The second term we call the probabilistic point loss. Once again using linearity of expectation,

$$\mathbb{E}_\pi \left[\sum_{y \in \mathcal{D}} \min_{x \in Q} \|x - y\|^2 \right] = \sum_{y \in \mathcal{D}} \mathbb{E}_\pi \left[\min_{x \in Q} \|x - y\|^2 \right] . \tag{5.11}$$

Consider this second term for a given $y \in \mathcal{D}$. We wish to compute the expected squared distance of the nearest point. Each point in σ has a corresponding squared distance d_k to y, and a corresponding probability of sampling p_k. Then the expected squared distance to the nearest point sampled is a sum of the distances, weighted by the probability that any given distance corresponds to the nearest point sampled. Without loss of generality, the squared distances are ordered such that $d_1 \leq d_2 \leq d_3 \leq \cdots$ and we can write

$$\mathbb{E}_\pi \left[\min_{x \in Q} \|x - y\|^2 \right] = d_1 p_1 + d_2 (1 - p_1) p_2 + d_3 (1 - p_1)(1 - p_2) p_3 + \cdots \tag{5.12}$$

$$= \sum_k d_k p_k \prod_{l=1}^{k-1} (1 - p_l) \tag{5.13}$$

The final sum is a sum over one such polynomial for every point in the input point-cloud, so the probabilistic point loss is a polynomial in the primitive probabilities.

$$\mathbb{E}_{\pi}\left[\sum_{y\in\mathcal{D}}\min_{x\in Q}\|x-y\|^2\right]=\sum_{y\in\mathcal{D}}\sum_{k}d_{k,y}p_{k,y}\prod_{l=1}^{k-1}(1-p_{l,y}) \tag{5.14}$$

5.4 Implementation details

We implement the neural network in PyTorch and use the IBM Horovod distributed training package. The network consists of three set abstraction layers, followed by two fully connected, batch norm, ReLU, and dropout layers. A final fully connected layer with a scaled hyperbolic tangent activation outputs a vector with values between 0 and 1.

5.4.1 Set Abstraction Layers

The backbone of the neural network is made of Pointnet [16] and Pointnet++'s [63] set abstraction layers. A set abstraction layer processes a set of points to compute a new set of fewer but higher-dimensional points containing features of the original point set. It does so in three layers: a sampling layer, a grouping layer, and an abstraction layer.

The sampling layer uses furthest point sampling to choose some subset of the input points to represent the point cloud. The grouping layer consists of a nearest neighbor or ball query around each sampled point meant to provide local context. The final abstraction layer consists of a multilayer perceptron followed by a max-pool. The multilayer perceptron can approximate arbitrary functions and the max-pool introduces desired symmetries into the representation. For more information see the cited papers.

5.5 Training and Results

We train the network on Satori, a high performance Power 9 cluster run by MIT and IBM. Each run uses PyTorch's Adam optimizer and is run for forty epochs. We train and test the network on simple synthetically generated toy examples. We consider a small set of nine possible primitives placed at disjoint locations. One such visualization can be seen in figure 5-1. To generate scenes, some subset of these nine primitives are chosen to be active, and points are randomly sampled from their surfaces, with both jitter (per-point noise) and drift (per-primitive noise).

The first experiment compares the performance of a network trained using our self supervised probabilistic chamfer loss to a network trained using a cross entropy loss. We train both networks with a dataset consisting of all $2^9 = 512$ combinations of existing and non existing primitives, and test on a similar dataset spanning all combinations of primitives but with different jitter and drifts. Our network outputs a prediction for each individual primitive whereas the supervised network classifies each primitive combination as one of 512 different classes. Thresholding the probability output of our network and comparing with the ground truth, we are able to correctly identify 98% of all primitives, and perfectly reconstruct 96% of scenes, with no extra stray primitives identified. The fully supervised cross-entropy network does similarly well, achieving an accuracy of 98% across scenes.

In the second experiment, we train the self-supervised probabilistic chamfer loss network on all of the combinations except for the single primitive scenes. We then test on these single primitive scenes which the network has never seen before. The

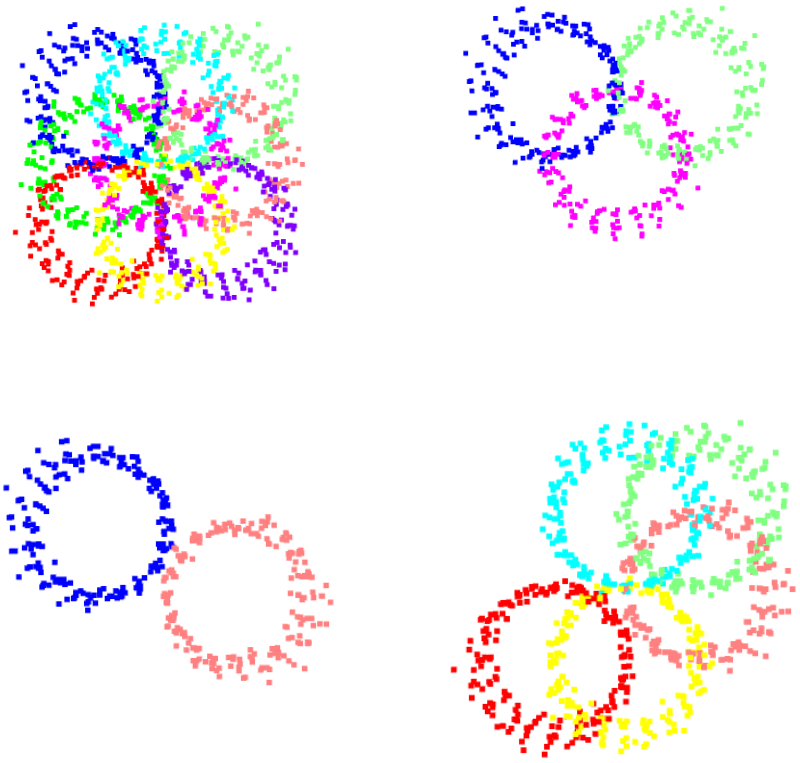


Figure 5-1: Top down view of four sample scenes used to train the neural network. Each scene is made up some combination of nine cylinders. Once the cylinders have been selected, points are sampled from their surfaces and perturbed to create the training scene.

Train	Test	Scene Acc.	Primitive Acc.
All combinations	All combinations	0.96	0.98
Multiple-primitive	Single-primitive	0.85	0.85
Single-primitive	Multiple-primitive	0.05	0.33

Table 5.1: Test accuracies for various training and test sets.

network achieves 85% accuracy on these new scenes. For the third experiment, we train the network on scenes each including a single primitive. We then test the network on pairs of primitives. The scene accuracy is no higher than 5%, with only 33% of primitives correctly detected, and an extra 43% of erroneous primitives detected. These results are summarized in table 5.1.

We also examine the effects of discretization and alignment on the neural network. Due to our discretization of the primitive space, only a finite set of primitives are considered. When this finite set of primitives aligns with the true pointcloud, we are able to achieve good reconstructions. In cases where none of the primitives considered are good fits, we achieve significantly worse reconstructions. To quantify the effect of this alignment on the network performance, we test the network on datasets with shifted alignment. Each test example is assigned a maximum offset from its primitive-aligned position, and the test example alignment is sampled from a uniform random variable between zero and this maximum offset. A maximum offset of 0 indicates that the pointcloud and primitives considered are perfectly aligned, and a maximum offset of 1 indicates that the maximum shift will align the pointcloud with a neighboring primitive. The effects of the maximum offset on the scene and primitive accuracy can be seen in figure 5-2.

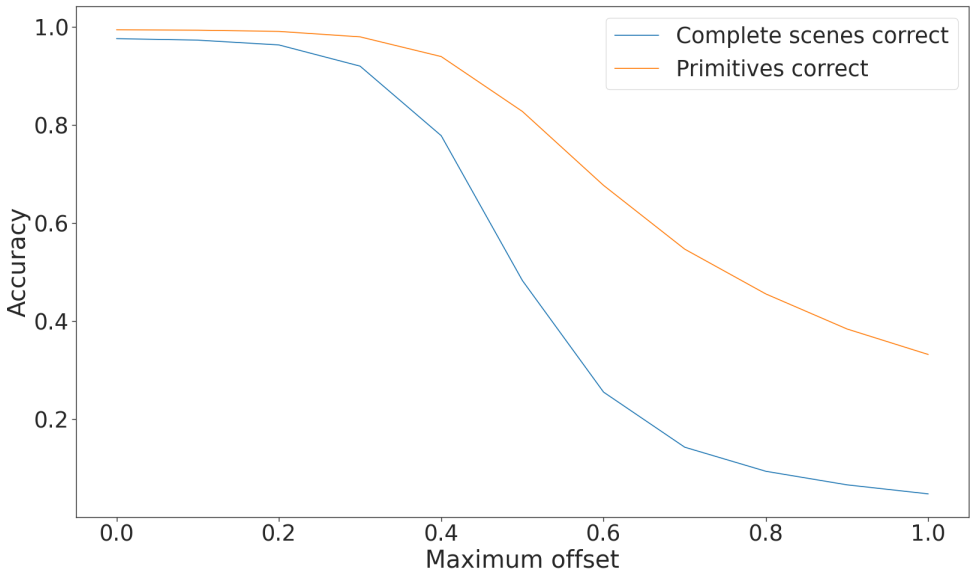


Figure 5-2: We plot the percentage of scenes and the percentage of primitives correctly reconstructed for various possible alignments. The maximum offset parameter determines the maximum difference between the true primitive-pointcloud alignment and the test primitive-pointcloud alignment.

5.6 Discussion

We find that given an equal training set, the self-supervised probabilistic chamfer loss network does approximately as well as the fully supervised network. This indicates that the self-supervised loss is at least as expressive as the softmax loss on this 9 primitive dataset. The similarity in performance is not surprising as the two networks have similar structure, except for the difference in loss functions.

While a 9 primitive dataset has 512 possible scenes, a 20 primitive dataset has over a million scenes, and even a modest 100 primitive dataset has over 10^{30} possible scenes. We show training times for networks with varying numbers of primitives in figure 5-3. We see that treating each scene as a separate class is not feasible for more complex environments. For any practical use, the network must be able to learn from simple training scenes and generalize to unseen test scenes. The second

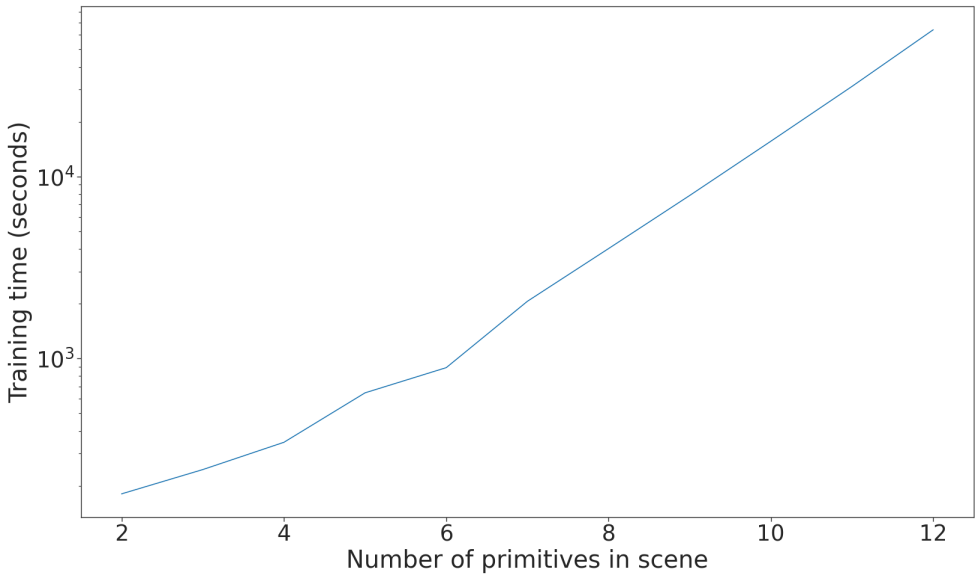


Figure 5-3: The time taken to train the neural network increases exponentially with the number of primitives considered.

experiments is the first of two experiments aimed at capturing this generalizability. In the second experiment, the network is trained on complex multi-primitive scenes, and tested on simple single-primitive scenes. Without having seen any single-primitive scenes, the network is able to choose the correct mapping between primitive location and output indices 85% of the time.

Unfortunately, we find that while the network generalizes from complex to simple scenes, it does not generalize from simple to complex scenes. We test the simplest such generalization by training on single-primitive scenes and testing on scenes of two primitives each. A scene accuracy of 5% indicates that the network is not learning the correct mappings between points in $\mathbb{R}^3$ and primitive indices, and is therefore unsuitable for use in its current form. Because of this disappointing result on a toy dataset, we refrain from testing on any larger or more complex examples and instead report this as a negative result.

Chapter 6

Inferring primitive scenes with a probabilistic chamfer distance

The probabilistic chamfer distance may be too complex to learn efficiently using our network, but our ability to analytically compute its gradient makes it well suited for iterative optimization methods. Gradient based optimizations require repeated evaluations, so we use this chapter to develop approximations to the exact probabilistic chamfer distance. Using these approximations, we are able to present an algorithm with runtime linear in the size of the input pointcloud $|\mathcal{D}|$ and linear in the number of primitives considered $|\mathcal{G}|$. This efficient evaluation allows us to optimize the probabilistic chamfer distance on scenes with thousands of primitives and tens of thousands of points.

6.1 Revisiting the probabilistic chamfer distance

6.1.1 Probabilistic primitive loss

Recall that the expression for the probabilistic primitive loss is given b y

$$\mathbb{E}_{\pi}\left[\sum_{x\in Q}\min_{y\in\mathcal{D}}\|x-y\|^2\right]=\sum_i p_i\sum_{x\in\sigma_i}\min_{y\in\mathcal{D}}\|x-y\|^2. \tag{6.1}$$

The probabilistic primitive loss is a linear function of the primitive probabilities. Furthermore, notice that the coefficients of this linear function do not depend on the probabilities themselves: they depend only on the input pointcloud $\mathcal{D}$ as well as the points corresponding to the possible primitive projections σ. This allows us to compute the coefficients once, and re-use them for subsequent optimization function calls. The algorithm to do so is given in algorithm 4.

Algorithm 4 Computing the probabilistic primitive loss coefficients

```
1:  procedure PROBABILISTIC PRIMITIVE LOSS COEFF(D, σ)
2:      KDTree ← MAKEKDTREE(D)
3:      g ← Zeros(N)                          ▷ Vector to hold coefficients
4:      for i ← 1 to N do                     ▷ Iterating over the number of primitives
5:          for x_k ∈ σ_i do
6:              Increment g_i by NEARESTDISTANCE(KDTree, x_k)^2
7:          end for
8:      end for
9:      return g
10: end procedure
```

6.1.2 Probabilistic point loss

The expression for the probabilistic point loss is given by

$$\mathbb{E}_\pi \left[\sum_{y \in \mathcal{D}} \min_{x \in Q} \|x - y\|^2 \right] = \sum_{y \in \mathcal{D}} \sum_k d_{k,y} p_{k,y} \prod_{l=1}^{k-1} (1 - p_{l,y}). \tag{6.2}$$

where the first sum is over points in $\mathcal{D}$, and the second sum is over the nearest points in σ. Note that a single p_i may correspond to multiple p_k, as p_i is the probability value assigned to the i^{th} primitive, and p_k is the probability value assigned to the primitive corresponding to the k^{th} nearest point to the pointcloud point y_j. The expression for the probabilistic point loss is also a polynomial in the p_i, whose coefficients are once again defined by only $\mathcal{D}$ and σ. As the polynomial corresponding to y_j is determined by the nearest point distances to it, our precomputation need only compute these distances as well as the nearest point ordering. We provide a possible way to compute these coefficients in algorithm 5.

Algorithm 5 Computing the exact probabilistic point loss coefficients

```
procedure SINGLE POINT COEFF(y_j, σ)
    distances ← [ ]
    for i ← 1 to N do
        for z ∈ σ_i do
            Append d(y_j, z)^2 to distances
        end for
    end for
    sorted distances, sorting indices ← SORT(distances)
    return sorted distances, sorting indices
end procedure
procedure PROBABILISTIC POINT LOSS EXACT COEFFICIENTS(D, σ)
    for y_j ∈ D do
        Accumulate SINGLE POINT COEFF(y_j, σ) in all sorted distances, all sorting indices
    end for
    return all sorted distances, all sorting indices
end procedure
```

Although the coefficients are straightforward to compute, this polynomial has

many more terms than the linear probabilistic primitive loss given in equation 5.10, so care must be taken in its evaluation. Given the squared point distances d, one could expand the expression for the probabilistic point loss to find the coefficients of the terms, but such an expansion would be slow and impractical. We instead factor the polynomial so that each p_k appears only once per y_j and exploit this factoring in a recursive expression.

This recursion is given in terms of the expecting remaining distance, ERD. An ERD_k for point y_j corresponds to the expected distance of the nearest sampled point in σ to y_j, given that the nearest k points were not sampled. Notice that the probabilistic point loss is the sum of all ERD_0 for each y_j. Seen slightly differently, for one given y_j,

$$\begin{aligned}
\text{ERD}_0 &= \sum_k d_k p_k \prod_{l=1}^{k-1}(1-p_l) && (6.3)\\
&= d_1 p_1 + d_2 p_2 (1-p_1) + d_3 p_3 (1-p_1)(1-p_2) + \cdots && (6.4)\\
&= p_1 d_1 + (1-p_1)\Big[p_2 d_2 + (1-p_2)\Big[p_3 d_3 + (1-p_3)[\cdots]\Big]\Big] && (6.5)\\
&= p_1 d_1 + (1-p_1)\text{ERD}_1 && (6.6)
\end{aligned}$$

Similarly, $\text{ERD}_{k-1} = p_k d_k + (1-p_k)\text{ERD}_k$.

6.1.3 Runtime analysis

Suppose we have $|\mathcal{D}| = J$ points in our input pointcloud, $|\mathcal{G}| = N$ primitives in our grounded primitive set, less than $|\sigma_i| \leq M$ points in each primitive projection, and we evaluate the probablistic chamfer loss and gradient t times in our optimization. The probabilistic primitive loss consists of creating a KD-Tree for the pointcloud

points and one nearest neighbor query for each primitive point. Evaluating the primitive loss and gradient is done in $O(N)$, so altogether, the probabilistic primitive loss runs in $O((J + NM) \log J + tN)$.

Computing the coefficients for the probabilistic point loss involves computing distances between J pointcloud points and NM primitive points. Each of these NM distances for each pointcloud point are then sorted. These steps run in $O(JNM(\log N + \log M))$. Evaluation involves computing $O(JNM)$ expected remaining distances, so therefore the probabilistic point loss and gradient computation runs in $O(JNM(\log N + \log M) + JNMt)$.

6.2 Approximations

Because the probabilistic primitive loss relatively is cheap to compute, it needs no approximations and instead we focus on approximations for the probabilistic point loss. We note that each polynomial created for the probabilistic point loss is a polynomial in N variables each of degree M. This suggest two possible approximations. The first reduces the degree of the polynomial by considering the effects of the expected remaining distance. The second approximation reduces the number of variables by considering the locality of each point in the pointcloud.

6.2.1 Reducing the degree of the polynomial

For each point $y_j \in \mathcal{D}$, the degree of the polynomial is based on the number of nearest neighbors considered from each primitive. By considering only the α points from each primitive nearest to y_j, the degree of the polynomial becomes α. We

build intuition that this approximation is a good approximation by considering two possibilities: either the p_i corresponding to these α points is high, or it is low. If the p_i is high, the probability that the $(\alpha+1)^{st}$ point is the nearest point to y_j is low because it is likely that one of the nearer α points is projected. And if p_i is low, then the probability that the $(\alpha+1)^{st}$ point is the nearest point is once again low because its probability of projection is low.

This truncation improves the runtime. The weights of the probabilistic point loss now include an additional $O(NM \log M)$ term to form the KD-Trees to compute the α nearest points to each y_j, but the overall construction is now $O(NM \log M + \alpha JN \log M)$ and the evaluation is now $O(\alpha JNs)$ for an overall runtime of $O(NM \log M + \alpha JN \log M + \alpha JNs)$ [1].

6.2.2 Reducing the number of variables in the polynomial

The second approximation we make is to limit the number of primitives considered for each point. By only considering the β nearest primitives to y_j, we further reduce the complexity to $O(NM \log M + \alpha\beta J \log N \log M + \alpha\beta Js)$.

We present an efficient algorithm for computing these approximate coefficients in algorithm 6.

6.3 Results and Discussion

We test our approach on three tasks. The first task is to fit geometric primitives to a pipe with multiple bends. This task is meant to simulate the sort of primitive fitting

[1] With an efficient implementation, the $O(NM \log M)$ tree construction can also be reduced by noting that each KD tree of a given primitive type is isomorphic to the others.

Algorithm 6 Computing the approximate probabilistic point loss coefficients

procedure SINGLE POINT APPROX COEFF(y_j, KDTree$_i$, PrimitiveKDTree, α, β)
    nearest primitives $\leftarrow$ NEARESTNEIGHBORS(y_j, PrimitiveKDTree, β)
    distances $\leftarrow$ [ ]
    for $i \in$ nearest primitives **do**
        nearest neighbors $\leftarrow$ NEARESTNEIGHBORS(y_j, KDTree$_i$, α)
        for $z \in$ nearest neighbors **do**
            Append $d(y_j, z)^2$ to distances
        end for
    end for
    sorted distances, sorting indices $\leftarrow$ SORT(distances)
    return sorted distances, sorting indices, nearest primitives
end procedure
procedure PROBABILISTIC POINT LOSS APPROX COEFF($\mathcal{D}, \sigma, \alpha, \beta$)
    for $i \leftarrow 1$ to N **do**
        KDTree$_i \leftarrow$ MAKEKDTREE(σ_i)
    end for
    PrimitiveKDTree $\leftarrow$ MAKEKDTREE(σ centers)
    sorted distances $\leftarrow$ [ ]
    sorting indices $\leftarrow$ [ ]
    nearest primitives $\leftarrow$ [ ]
    for $y_j \in \mathcal{D}$ **do**
        Accumulate SINGLE POINT APPROX COEFF(y_j, KDTree$_i$, PrimitiveKDTree, α, β) in sorted distances, sorting indices, nearest primitives
    end for
    return sorted distances, sorting indices, all nearest primitives
end procedure

that one might encounter while solving a manipulation problem. For our second task, we revisit a hallway domain to fit primitives to walls. This task is meant to simulate a navigation problem. Finally, we apply our approach to a registration problem. Our current implementation of the probabilistic chamfer distance is written in Python with NumPy [30] and tested on an Intel Core i7. Profiling shows that the vast majority of computation is spent doing nearest neighbor queries in the chamfer distance coefficient construction, so we estimate that an efficient C++ implementation might be between one and two orders of magnitude faster. The results from a preliminary C++ implementation is shown in figure 6-6.

6.3.1 Manipulation

We generate a pipe made of some combination of straight segments and right angle elbows. We sample points from the surface of this pipe, and these points become the input pointcloud $\mathcal{D}$ that we try to match. This problem is difficult because we do not *a priori* know the shape of the pipe or how many segments it has, and the boundaries of the segments are unclear. We find that even in the presence of pointcloud jitter, we are able to accurately detect and reconstruct the pipes. A scene such as the ones shown in figure 6-1 is solved in about 30 seconds. We run this example with $\alpha = 10$, $\beta = \infty$.

We also explore the effects of various discretizations in figure 6-2. We run 20 trials at each discretization, varying the alignment of the input pointcloud with the possible primitives in $\mathcal{G}$ for each trial. A discretization value of zero represents the base discretization necessary to fully represent the scene if the pointcloud and the existing primitives are well aligned, and discretization values deviating from 0 indicate a denser or sparser $\mathcal{G}$. In particular, for a discretization value of i, the spacing

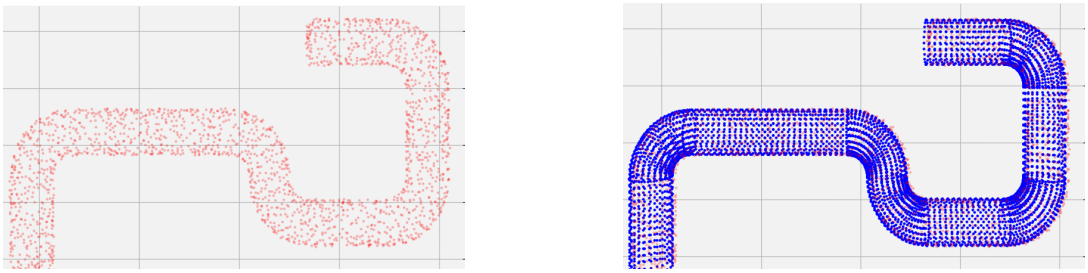


Figure 6-1: We are able to fit geometric primitives to pipes with an unknown shape and number of parts. The left column contains the input pointcloud, and the right column contains the fit primitives.

between primitives is given by 2^i times the base spacing. Unsurprisingly, denser discretizations allow for better fits. The average error for a discretization value of -2 is 275,000: just over the error of 245,000 given by hand-tuning the pointcloud-primitive alignment in figure 6-1. The second graph in figure 6-2 indicates that solve time scales linearly with the number of primitives considered as predicted by our analysis.

The effects of α on the probabilistic chamfer distance and on solve time is shown in figure 6-3. We find that, as expected, fewer points in the approximation leads to faster solve times, but also results in primitive scenes with higher error. While solve time increases relatively monotonically with α, the error introduced by the approximation seems to plateau around $\alpha = 10$.

6.3.2 Navigation

To test the navigation capabilities of a map build with the probabilistic chamfer loss, we fit square patches to a hallway pointcloud input from the Multisensorial

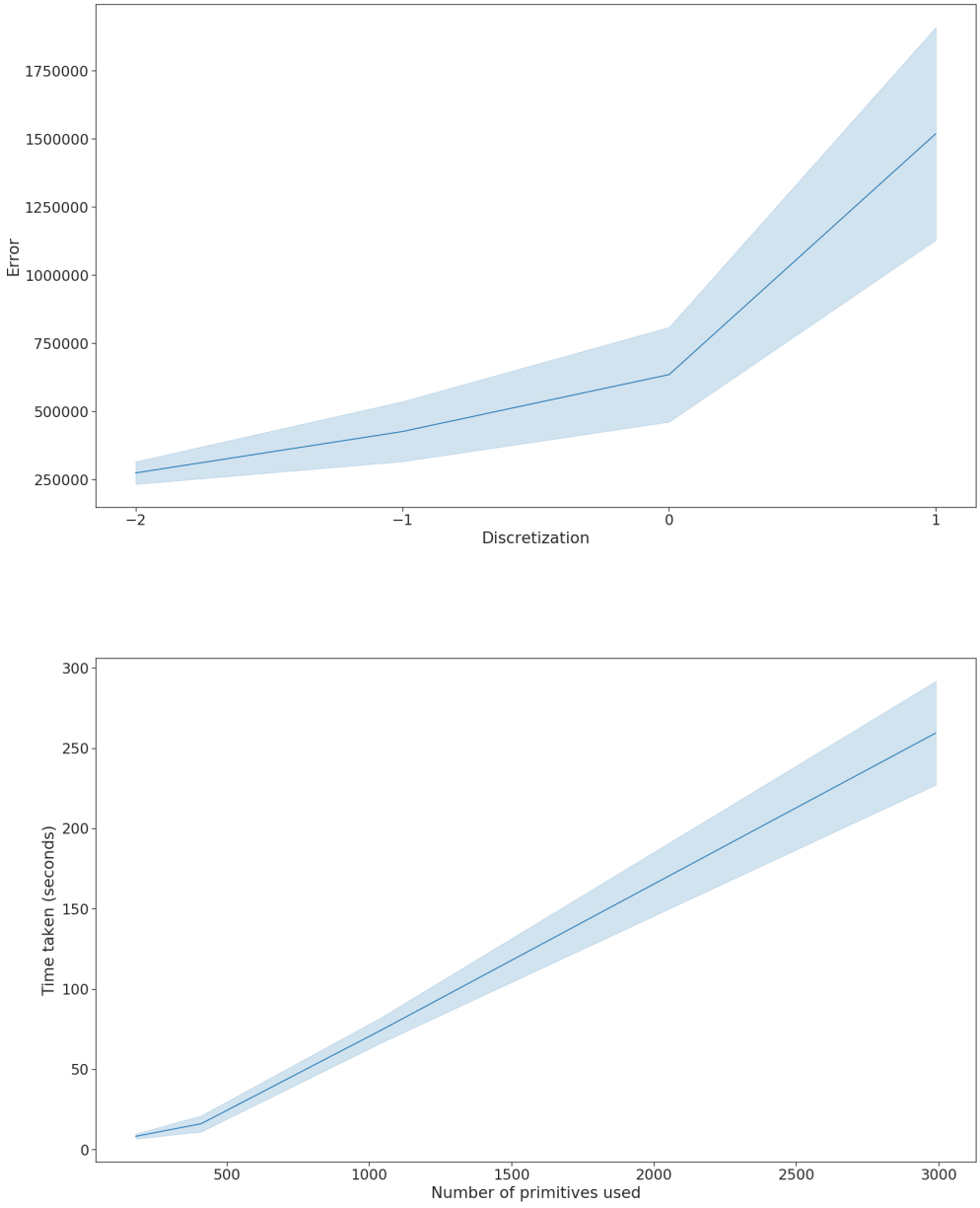


Figure 6-2: We measure the effects of discretization on both the error and the solve time. A discretization value of i means the spacing is given by 2^i times the base spacing. Solid lines represent the mean of twenty trials, while the shading represents the standard deviation.

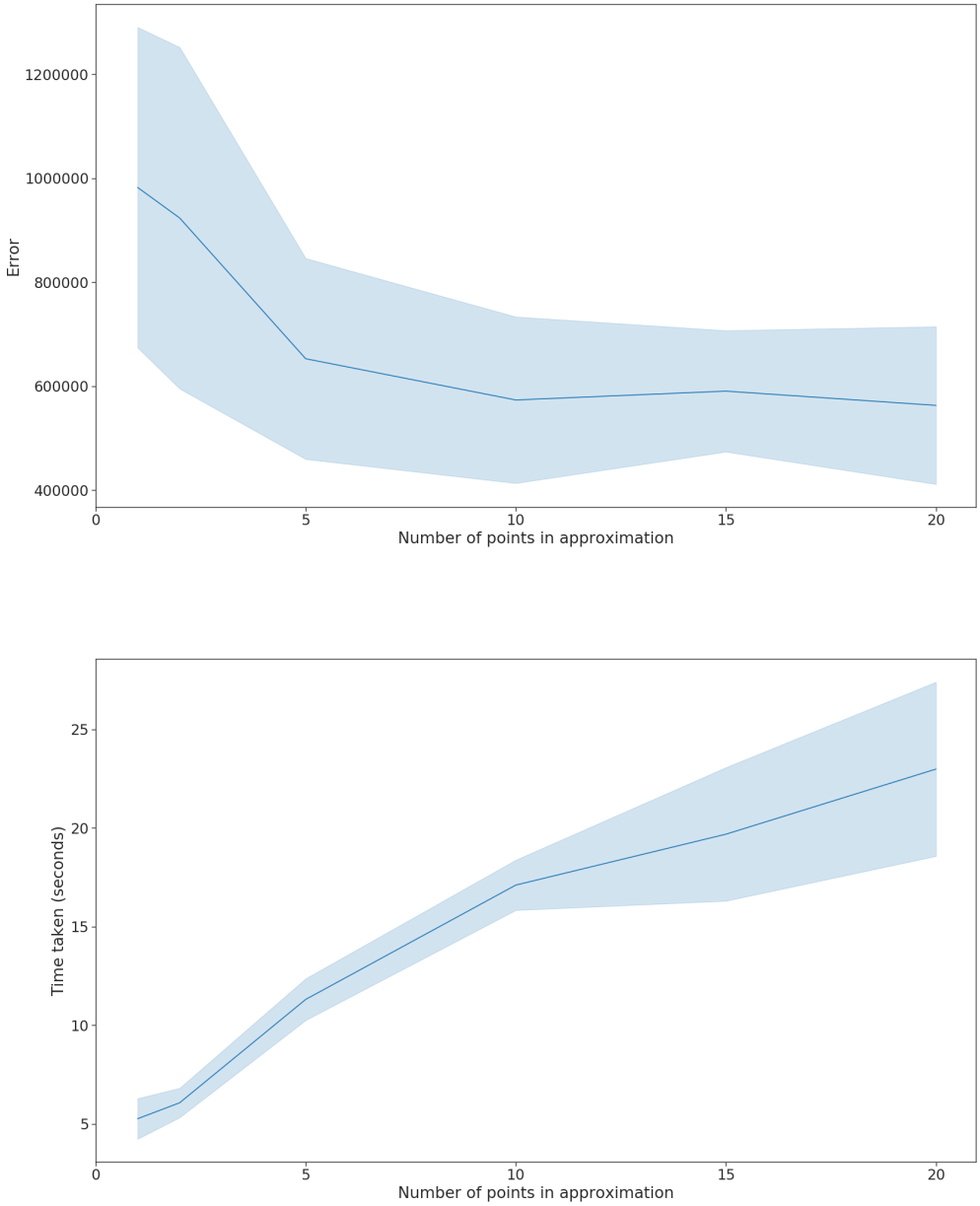


Figure 6-3: We measure the effect of α on both the error and the solve time. Solid lines represent the mean of twenty trials, while the shading represents the standard deviation.

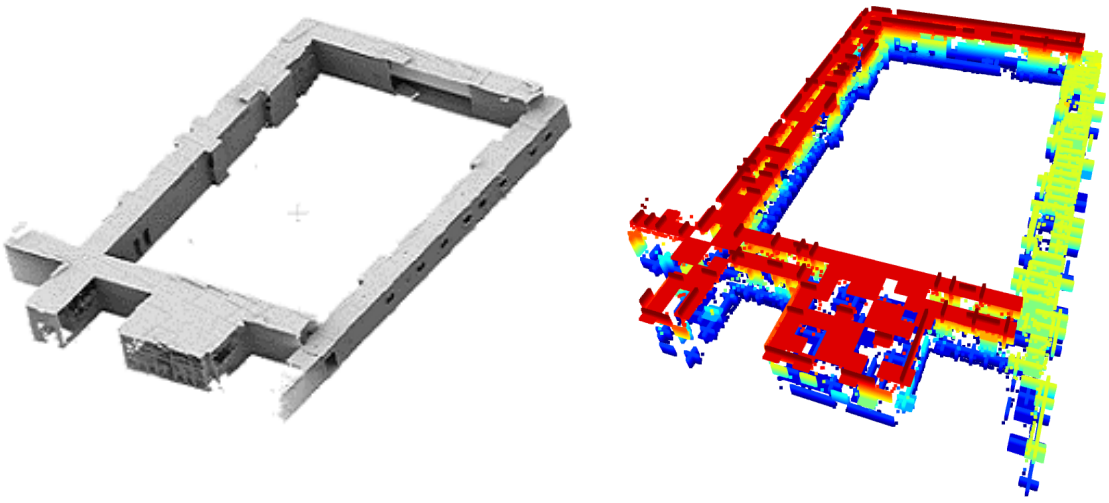


Figure 6-4: A hallway scene from the Multisensorial Indoor Mapping and Positioning Dataset [78]. The full pointcloud scene (left) compared to the initial fitted primitive scene (right).

Indoor Mapping and Positioning Dataset [78]. An image of the input pointcloud and reconstructed scene can be found in figure 6-4. Our examples are run with $\alpha = 10, \beta = 125$.

We measure the mean error from each pointcloud point to the nearest primitive, and present a histogram of the results in figure 6-5. When considering square primitives with 1 meter side lengths, the expected distance between a point in the input pointcloud and the nearest primitive is 0.22 meters, and every point in the input pointcloud has at least one primitive within a meter of it.

We note that the reconstructed figure in figure 6-4 seems to have weaknesses similar to the navigation reconstruction in Chapter 4. In this case, though, the weaknesses are caused by computational constraints. The python implementation requires aggressive sparsification of the input pointcloud to fit the required arrays in memory. Although out of scope for this thesis, the results of a preliminary C++

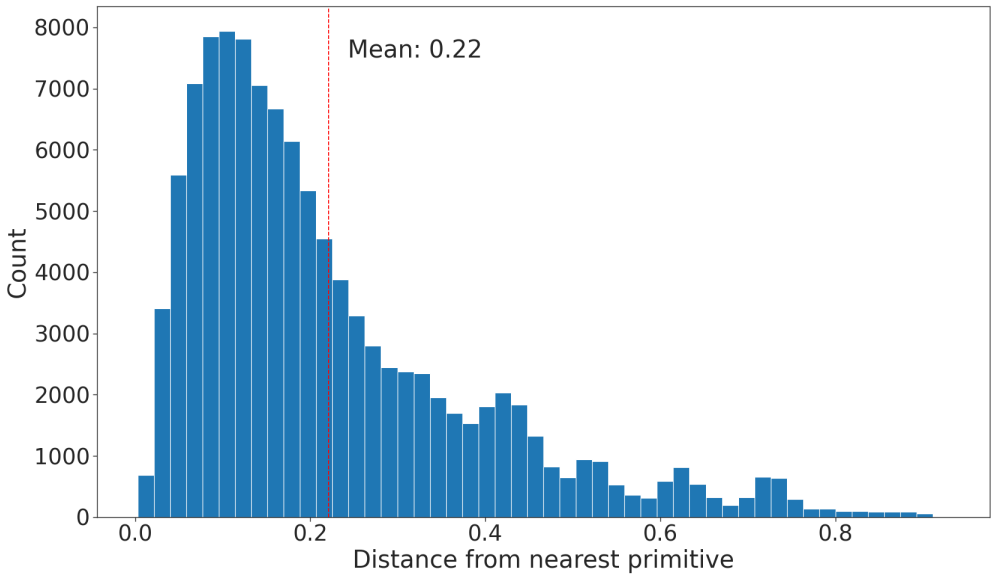


Figure 6-5: A histogram of the mean error between pointcloud points and their nearest primitives in figure 6-4.

implementation are included in figure 6-6 and figure 6-7. Further analysis will be performed in future work.

6.3.3 Registration

Pointcloud registration is a difficult problem to solve using this space discretization approach. Although there are in general fewer "true" primitives that must be found, this sparsity does not benefit our approach. Furthermore, the inherent error introduced by the discretization is dominant in the continuous pose and scale space. We include a particle-filter type approach, where particles represent parameters of various geometric primitives, and our differentiable chamfer loss acts as the measurement function, but note that it is not a practical algorithm compared to single-primitive registration methods such as ICP [3] or Teaser [84]. Nevertheless, we include results as they show the flexibility of the algorithm, run with $\alpha = 10, \beta = \infty$.

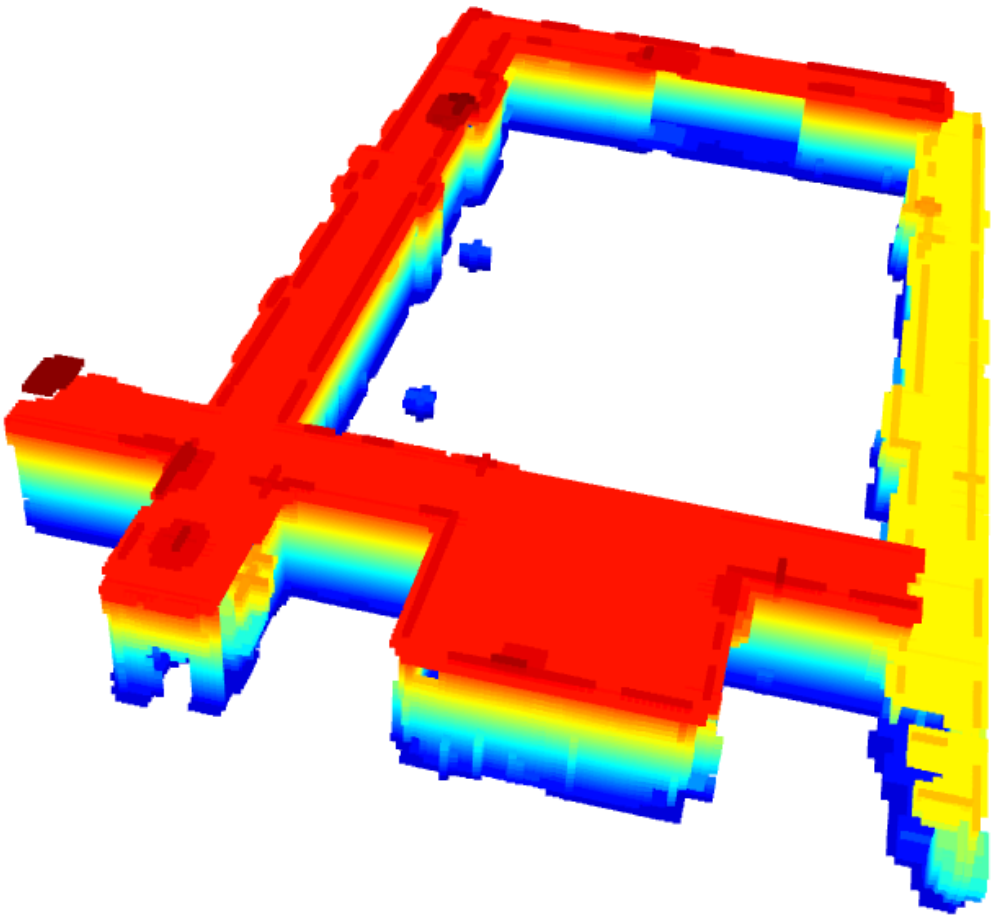


Figure 6-6: The C++ implementation results from fitting primitives to the Multi-sensorial Indoor Mapping and Positioning Dataset [78]. The increased efficiency and precise control over memory management allows us to run with an order of magnitude more points, which results in a better fitting.

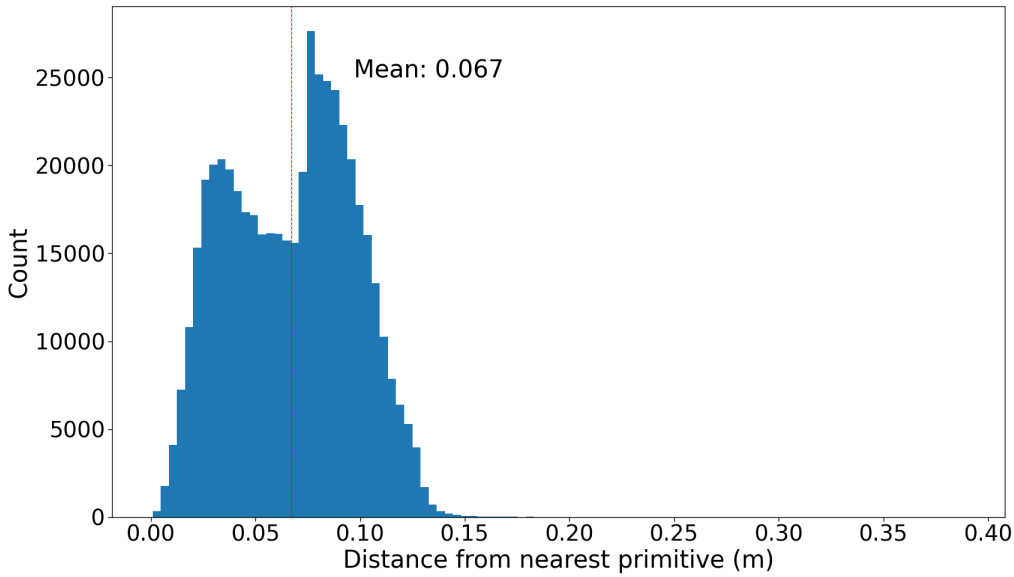


Figure 6-7: A histogram of the mean error between pointcloud points and their nearest primitives in figure 6-6.

In figure 6-8, we demonstrate the particle filter registration approach. We start with multiple possible primitives, and after a few resampling iterations converge to a qualitatively good solution. In figure 6-8 we estimate only the rotation of the primitive in the presence of 95% noise. To generate the input pointcloud, we sample 100 points from the surface of a randomly rotated true bunny, and augment the pointcloud with 1900 points uniformly sampled from a similar area. The false points are explicitly modeled by a primitive that contains points uniformly sampled from space. In figure 6-9 we show that we are able to register multiple primitives at the same time and estimate translation as well as rotation.

We find that the probabilistic chamfer loss is not particularly well suited to registration. Because parameters are never estimated directly, the only way to find the correct primitive to fit to a given set of points is to randomly sample it from pa-

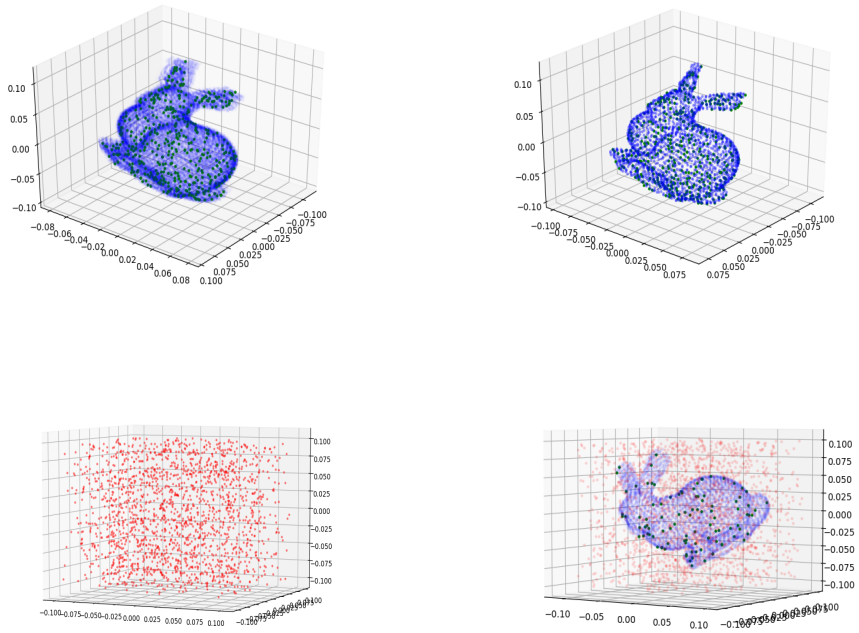


Figure 6-8: Various registration examples. More transparent points indicate a lower likelihood of that primitive (top left). After 20 iterations, we have less uncertainty over the pose of the primitive (top right). 1900 points of random noise, and 100 points sampled from the surface of a primitive (bottom left). We are able to find the rotation of the primitive in a noisy pointcloud (bottom right).

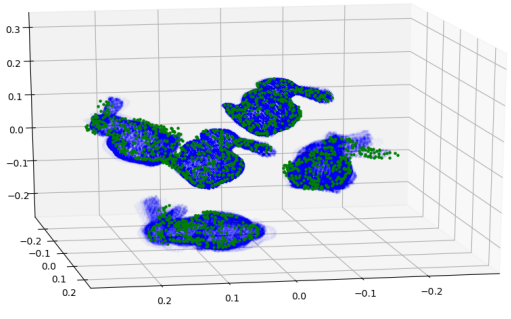


Figure 6-9: We are able to register multiple primitive in a single scene.

rameter space in the particle resampling step. The probability of sampling a pose that is sufficiently close decreases exponentially with dimensionality, and we see that we are able to much more easily sample close rotations from a 3 dimensional space than we are able to sample close poses from a 6 dimensional space.

Chapter 7

Conclusion

To conclude, we revisit our guiding question:

How can we infer the existence of geometric primitives, and is inferring existence useful when fitting primitives to non-discriminable data?

While the first part of the question has not been fully answered, it seems reasonable to say that the answer to the second part of the question is yes. In this work, we have presented three possibilities for inferring existence of geometric primitives representing *stuff*. While the sampling based approach had critical flaws, and the probabilistic chamfer distance based neural network needed certain network restructuring, we found that the direct optimization of the probabilistic chamfer distance revealed promising results.

We provided an algorithm that computes an approximation of the probabilistic chamfer distance and its gradient in linear time with the size of the input point-cloud. Fundamental to our approach of inferring existence is the idea of probabilistic existence. By relaxing what would have previously been a binary constraint,

we are able to efficiently compute gradients and turn a combinatorial optimization problem into a standard polynomial optimization problem. Testing this algorithm on various datasets we find promising results that indicate that inferring existence may be a viable way to fit geometric primitives to non-semantically meaningful data.

Further work could take various directions. One option is to implement the probabilistic chamfer distance in a compiled language to increase the scale of the problems it can be used to solve. Another option is to use the output of the probablistic chamfer distance to solve planning problems such as manipulation or navigation. Alternatively, an incremental solve might allow near-realtime performance on simple scenes such as the hallway navigation scene from Chapter 6. If this incremental optimization and near-realtime performance is achieved, it might become feasible to use primitives generated by the probabilistic chamfer distance in a factor graph for a SLAM system. Finally, we note that our approach is not more expressive than the chamfer distance itself, and in certain cases the chamfer distance might not capture behavior which is desirable to model. Future work could use a similar probabilistic approach to different pointcloud distance functions.

We initially set out to find a general purpose algorithm able to fit geometric primitives to a whole scene without a strong prior on what the scene would look like. While our current implementation is too slow to fully build large scenes, and some priors are needed to choose primitives to be considered, it sets up a new framework capable of building fully primitive scenes with weaker priors than existing approaches. We are excited to see where this work goes next.

Bibliography

[1] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic. Netvlad: Cnn architecture for weakly supervised place recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5297–5307, 2016.

[2] David Arthur and Sergei Vassilvitskii. K-means++: the advantages of careful seeding. In *In Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2007.

[3] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9(5):698–700, May 1987.

[4] N. Atanasov, M. Zhu, K. Daniilidis, and G. Pappas. Localization from semantic observations via the matrix permanent. *The International Journal of Robotics Research (IJRR)*, 35:73–99, 2015.

[5] T. Bagautdinov, F. Fleuret, and P. Fua. Probability occupancy maps for occluded depth images. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2829–2837, 2015.

[6] Y. Bar-Shalom, F. Daum, and J. Huang. The probabilistic data association filter. *IEEE Control Systems Magazine*, 29(6):82–100, 2009.

[7] Yaakov Bar-Shalom and Edison Tse. Tracking in a cluttered environment with probabilistic data association. 1975.

[8] H. G. Barrow, J. M. Tenenbaum, R. C. Bolles, and H. C. Wolf. Parametric correspondence and chamfer matching: Two new techniques for image matching. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'77, page 659–663, San Francisco, CA, USA, 1977. Morgan Kaufmann Publishers Inc.

[9] A. J. Barry and R. Tedrake. Pushbroom stereo for high-speed navigation in cluttered environments. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3046–3052, 2015.

[10] Yoshua Bengio, N. Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *ArXiv*, abs/1308.3432, 2013.

[11] S. S. Blackman. Multiple hypothesis tracking for multiple target tracking. *IEEE Aerospace and Electronic Systems Magazine*, 19(1):5–18, 2004.

[12] Sean L. Bowman, Nikolay Atanasov, Kostas Daniilidis, and George J. Pappas. Probabilistic data association for semantic slam. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1722–1729, 2017.

[13] Eric Brachmann, Alexander Krull, Sebastian Nowozin, J. Shotton, F. Michel, S. Gumhold, and C. Rother. Dsac — differentiable ransac for camera localization. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2492–2500, 2017.

[14] Alvaro Parra Bustos and Tat-Jun Chin. Guaranteed outlier removal for rotation search. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, page 2165–2173, USA, 2015. IEEE Computer Society.

[15] Trevor Campbell, Miao Liu, Brian Kulis, Jonathan P How, and Lawrence Carin. Dynamic clustering via asymptotics of the dependent dirichlet process mixture. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 449–457. Curran Associates, Inc., 2013.

[16] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, 2017.

[17] Jiawen Chen, Dennis Bautembach, and Shahram Izadi. Scalable real-time volumetric surface reconstruction. *ACM Trans. Graph.*, 32(4), July 2013.

[18] I. J. Cox and J. J. Leonard. Probabilistic data association for dynamic world modeling: a multiple hypothesis approach. In *Fifth International Conference on Advanced Robotics 'Robots in Unstructured Environments*, pages 1287–1294 vol.2, 1991.

[19] Ingemar J. Cox and John J. Leonard. Modeling a dynamic environment using a bayesian multiple hypothesis approach. *Artificial Intelligence*, 66(2):311 – 344, 1994.

[20] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[21] K. Doherty, D. Fourie, and J. Leonard. Multimodal semantic slam with probabilistic data association. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2419–2425, 2019.

[22] H. E. Elghor, D. Roussel, F. Ababsa, and E. H. Bouyakhf. Planes detection for robust localization and mapping in rgb-d slam systems. In *2015 International Conference on 3D Vision*, pages 452–459, 2015.

[23] S. M. Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, koray kavukcuoglu, and Geoffrey E Hinton. Attend, infer, repeat: Fast scene understanding with generative models. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29, pages 3225–3233. Curran Associates, Inc., 2016.

[24] D. Forsyth, J. Malik, Margaret M. Fleck, H. Greenspan, Thomas Leung, Serge J. Belongie, C. Carson, and C. Bregler. Finding pictures of objects in large collections of images. In *Object Representation in Computer Vision*, 1996.

[25] I. R. Goodman, Ronald P. Mahler, and Hung T. Nguyen. *Mathematics of Data Fusion*. Kluwer Academic Publishers, USA, 1997.

[26] Peter J. Green. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82(4):711–732, 12 1995.

[27] W. N. Greene and N. Roy. Flame: Fast lightweight mesh estimation using variational smoothing on delaunay graphs. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 4696–4704, 2017.

[28] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan C. Russell, and Mathieu Aubry. 3d-coded: 3d correspondences by deep deformation. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, pages 235–251, Cham, 2018. Springer International Publishing.

[29] Martin Habbecke and Leif Kobbelt. A surface-growing approach to multi-view stereo reconstruction. In *IN CVPR, 2007. [4] . OKUTOMI*.

[30] Charles R. Harris, K. Jarrod Millman, St'efan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fern'andez del R'10, Mark Wiebe, Pearu Peterson, Pierre G'erard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[31] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 04 1970.

[32] C. M. Hoffmann, J. E. Hopcroft, and M. J. Karasick. Robust set operations on polyhedral solids. *IEEE Computer Graphics and Applications*, 9(6):50–59, 1989.

[33] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013.

[34] N. Ikoma, T. Uchino, and H. Maeda. Tracking of feature points in image sequence by smc implementation of phd filter. In *SICE 2004 Annual Conference*, volume 2, pages 1696–1701 vol. 2, 2004.

[35] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, page 559–568, New York, NY, USA, 2011. Association for Computing Machinery.

[36] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. 2017.

[37] Adam M. Johansen, Sumeetpal S. Singh, Arnaud Doucet, and Ba ngu Vo. Convergence of the smc implementation of the phd filter cued/f-infeng/tr-517, 2005.

[38] M. Kaess. Simultaneous localization and mapping with infinite planes. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4605–4611, 2015.

[39] S. H. Khan, Xuming He, M. Bannamoun, F. Sohel, and R. Togneri. Separating objects and clutter in indoor scenes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4603–4611, 2015.

[40] K. Konolige, E. Marder-Eppstein, and B. Marthi. Navigation in hybrid metric-topological maps. In *2011 IEEE International Conference on Robotics and Automation*, pages 3041–3047, 2011.

[41] Michael Krainin, Peter Henry, Xiaofeng Ren, and Dieter Fox. Manipulator and object tracking for in-hand 3d object modeling. *The International Journal of Robotics Research*, 30(11):1311–1327, 2011.

[42] Florent Lafarge and Clément Mallet. Creating large-scale city models from 3d-point clouds: A robust approach with hybrid representation. *International Journal of Computer Vision*, 99, 08 2012.

[43] David H. Laidlaw, W. Benjamin Trumbore, and John F. Hughes. Constructive solid geometry for polyhedral objects. *SIGGRAPH Comput. Graph.*, 20(4):161–170, August 1986.

[44] C. Li, T. Simon, J. Saragih, B. Póczos, and Y. Sheikh. Lbs autoencoder: Self-supervised fitting of articulated meshes to point clouds. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11959–11968, 2019.

[45] Duanshun Li and Chen Feng. Primitive fitting using deep geometric segmentation. In Mohamed Al-Hussein, editor, *Proceedings of the 36th International Symposium on Automation and Robotics in Construction (ISARC)*, pages 780–787, Banff, Canada, May 2019. International Association for Automation and Robotics in Construction (IAARC).

[46] J. Li and G. H. Lee. Usip: Unsupervised stable interest point detection from 3d point clouds. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 361–370, 2019.

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

[48] Yangyan Li, Xiaokun Wu, Yiorgos Chrysathou, Andrei Sharf, Daniel Cohen-Or, and Niloy J. Mitra. Globfit: Consistently fitting primitives by discovering global relations. *ACM Trans. Graph.*, 30(4), July 2011.

[49] Kok-lim Low. Linear least-squares optimization for point-toplane icp surface registration. Technical report, 2004.

[50] Ronald P. S. Mahler. *Statistical Multisource-Multitarget Information Fusion*. Artech House, Inc., USA, 2007.

[51] A. Martinović, J. Knopp, H. Riemenschneider, and L. Van Gool. 3d all the way: Semantic segmentation of urban scenes from start to end in 3d. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4456–4465, 2015.

[52] Peter Meer, Doron Mintz, Azriel Rosenfeld, and Dong Yoon Kim. Robust regression methods for computer vision: A review. *Int. J. Comput. Vision*, 6(1):59–70, April 1991.

[53] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[54] H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 116–121, 1985.

[55] Manasi Muglikar, Zichao Zhang, and D. Scaramuzza. Voxel map for visual slam. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4181–4187, 2020.

[56] L. Nicholson, M. Milford, and N. Sünderhauf. Quadricslam: Dual quadrics from object detections as landmarks in object-oriented slam. *IEEE Robotics and Automation Letters*, 4(1):1–8, 2019.

[57] K. Ok, K. Liu, K. Frey, J. P. How, and N. Roy. Robust object-based slam for high-speed autonomous navigation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 669–675, 2019.

[58] Q. Pan, Gerhard Reitmayr, and Tom Drummond. Proforma: Probabilistic feature-based on-line rapid model acquisition. In *BMVC*, 2009.

[59] Daehyung Park, Michael Noseworthy, Rohan Paul, Subhro Roy, and Nicholas Roy. Inferring task goals and constraints using bayesian nonparametric inverse reinforcement learning. In Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura, editors, *3rd Annual Conference on Robot Learning, CoRL 2019, Osaka, Japan, October 30 - November 1, 2019, Proceedings*, volume 100 of *Proceedings of Machine Learning Research*, pages 1005–1014. PMLR, 2019.

[60] Despoina Paschalidou, Luc Van Gool, and Andreas Geiger. 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)*, June 2020.

[61] S. Pillai, S. Ramalingam, and J. J. Leonard. High-performance and tunable stereo reconstruction. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3188–3195, 2016.

[62] Charles R. Qi, O. Litany, Kaiming He, and L. Guibas. Deep hough voting for 3d object detection in point clouds. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 9276–9285, 2019.

[63] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 5105–5114, Red Hook, NY, USA, 2017. Curran Associates Inc.

[64] D. Reid. An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, 24(6):843–854, 1979.

[65] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, page II–1278–II–1286. JMLR.org, 2014.

[66] Antoni Rosinol, Torsten Sattler, M. Pollefeys, and L. Carlone. Incremental visual-inertial 3d mesh generation with structural regularities. *2019 International Conference on Robotics and Automation (ICRA)*, pages 8220–8226, 2019.

[67] K. Schmid, T. Tomic, F. Ruess, H. Hirschmüller, and M. Suppa. Stereo vision based indoor/outdoor navigation for flying robots. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3955–3962, 2013.

[68] R. Schnabel, R. Wahl, and R. Klein. Efficient RANSAC for Point-Cloud Shape Detection. *Computer Graphics Forum*, 2007.

[69] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, page 3528–3536, Cambridge, MA, USA, 2015. MIT Press.

[70] E. B. Sudderth, A. T. Ihler, W. T. Freeman, and A. S. Willsky. Nonparametric belief propagation. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, volume 1, pages I–I, 2003.

[71] L. Teixeira and M. Chli. Real-time mesh-based scene estimation for aerial inspection. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4863–4869, 2016.

[72] James Thewlis, Shuai Zheng, Philip Torr, and Andrea Vedaldi. Fully-trainable deep matching. pages 145.1–145.12, 01 2016.

[73] William C. Thibault and Bruce F. Naylor. Set operations on polyhedra using binary space partitioning trees. *SIGGRAPH Comput. Graph.*, 21(4):153–162, August 1987.

[74] Sebastian Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

[75] M. Tobias and A. D. Lanterman. Probability hypothesis density-based multitarget tracking with bistatic range and doppler observations. *IEE Proceedings - Radar, Sonar and Navigation*, 152(3):195–205, 2005.

[76] S. Tulsiani, H. Su, L. J. Guibas, A. A. Efros, and J. Malik. Learning shape abstractions by assembling volumetric primitives. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1466–1474, 2017.

[77] Ngo Vien and Marc Toussaint. Reasoning with uncertainties over existence of objects. pages 120–125, 01 2013.

[78] C. Wang, S. Hou, C. Wen, Z. Gong, Q. Li, X. Sun, and J. Li. Semantic line framework-based indoor building modeling using backpacked laser scanning point cloud. *ISPRS Journal of Photogrammetry and Remote Sensing*, 143:150 – 166, 2018.

[79] T. Weise, Thomas Wismer, B. Leibe, and L. Gool. In-hand scanning with online loop closure. *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 1630–1637, 2009.

[80] Thomas Whelan, Michael Kaess, Hordur Johannsson, Maurice Fallon, John J. Leonard, and John McDonald. Real-time large-scale dense rgb-d slam with volumetric fusion. *The International Journal of Robotics Research*, 34(4-5):598–626, 2015.

[81] A. Wilkowski, T. Kornuta, M. Stefańczyk, and W. Kasprzak. Efficient generation of 3d surfel maps using rgb–d sensors. *International Journal of Applied Mathematics and Computer Science*, 26:122 – 99, 2016.

[82] Wing-Kin Ma, Ba-Ngu Vo, S. S. Singh, and A. Baddeley. Tracking an unknown time-varying number of speakers using tdoa measurements: a random finite set approach. *IEEE Transactions on Signal Processing*, 54(9):3291–3304, 2006.

[83] L. L. S. Wong, L. P. Kaelbling, and T. Lozano-Perez. Not seeing is also believing: Combining object and metric spatial information. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1253–1260, 2014.

[84] H. Yang, J. Shi, and L. Carlone. TEASER: Fast and Certifiable Point Cloud Registration. *IEEE Transactions on Robotics (T-RO)*, 2020.

[85] Heng Yang and Luca Carlone. A polynomial-time solution for robust registration with extreme outlier rates. *Robotics: Science and Systems (RSS)*, 2019.

[86] S. Yang and S. Scherer. Monocular object and plane slam in structured environments. *IEEE Robotics and Automation Letters*, 4(4):3145–3152, 2019.

[87] S. Yang, Y. Song, M. Kaess, and S. Scherer. Pop-up slam: Semantic monocular plane slam for low-texture environments. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1222–1229, 2016.

[88] Li Yi, Wang Zhao, He Wang, Minhyuk Sung, and Leonidas J. Guibas. Gspn: Generative shape proposal network for 3d instance segmentation in point cloud. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[89] C. Zou, E. Yumer, J. Yang, D. Ceylan, and D. Hoiem. 3d-prnn: Generating shape primitives with recurrent neural networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 900–909, 2017.