Capturing Distributions over Worlds for Robotics with Spatial Scene Grammars

by

Gregory Izatt

B.S., California Institute of Technology (2014)
S.M., Massachusetts Institute of Technology (2017)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2022

Author .....
Department of Electrical Engineering and Computer Science
March 1, 2022

Certified by .....
Russ Tedrake
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Capturing Distributions over Worlds for Robotics with Spatial Scene Grammars

by

Gregory Izatt

Submitted to the Department of Electrical Engineering and Computer Science
on March 1, 2022, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Having a precise understanding of the *distribution over worlds* a robot will face is critical to most problems in robotics. This distribution informs mechanical and software design specifications, provides strong priors to perception, and quantifies the real-world relevance of simulation and lab testing. However, representing and quantifying this distribution is an open and difficult problem, as these worlds can vary in myriad continuous and discrete ways. This thesis is concerned with a particular class of probabilistic procedural models – *spatial scene grammars* – that are tailored to describe hybrid discrete-and-continuous distributions over environments with varying numbers, types, and spatial poses of objects. We develop a spatial scene grammar formulation that is sufficiently expressive to capture the structure of practically relevant environments, but is carefully restricted to remain amenable to various forms of probabilistic inference. We show that we can sample diverse scenes from these grammars, even under the presence of constraints on scene contents and object poses; that we can *parse* scenes with this grammar model via a novel set of mixed-integer parsing techniques to achieve detailed scene understanding and part-level outlier detection; and that we can fit unknown parameters in the model to data via an approximate expectation-maximization algorithm.

Thesis Supervisor: Russ Tedrake
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

It takes a village to write a thesis, and this one is no exception. I have so many people – and a few robots – to thank for so many kinds of support.

I owe enormous and heartfelt thanks to Russ, who has been gracious, patient, encouraging, and an excellent kind of challenging since the beginning – even (and especially) as I wandered between projects and ideas. Each of the members of my committee – Alberto and Leslie – have likewise been mentors to me across projects and contexts for years, and for that I am deeply grateful. And I'll never forget Seth, who welcomed me (when I wasn't much more than a cold-call email from an undergrad across the country) to MIT and kicked off my career in this wonderful field.[1]

My lab has been a source of constant inspiration and camaraderie – and a vital wellspring of energy and ideas to fuel the long grind inherent to a project like this one. The same goes for mentors and friends around CSAIL and MIT – I could always count on running into y'all at robotics seminars or in the Infinite and having a stimulating conversation (or mutual vent). I'm especially grateful to the members of the MIT DRC and Hyperloop teams for teaching me so much about being a good engineer and teammate, and for generally being a blast to work with.

I'm blessed with friends innumerable. Thanks, in particular, to my many amazing friends at Sidpac – SPEC, the HoH, officers, helpers, trustees, my undergrads, and all y'all at the front desk – as well as those farther flung across Boston and the wider world, for giving me amazing homes to go back to and joining me in adventures of sorts I couldn't imagine when I started. And finally, thank you to Hannah and my family – each of you being loving, supportive, and distracting at exactly the right times. I wouldn't have made it without y'all.

[1] Funnily enough, the project we planned together when we wrote my NSF GRFP proposal was *also* about distributions over environments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Having the ability to reason about, explain, and quantify the natural variation in the world is a fundamental aspect of intelligence. We humans have the ability to not just recall many of the kitchens we have experienced, but to imagine an infinite number of new ones – and of course, we quickly adapt to new kitchens as soon as we enter them by applying our knowledge of how kitchens are structured *in general*. This kind of knowledge is directly useful when designing robots: if we're designing a dishwashing robot for home kitchens, knowing the distribution of kitchens our robot will be operating in tells us exactly what kitchen layouts, dish arrangements, and object properties we will need to prepare for.

Unfortunately, for complex real-world environments like kitchens, it's not at all clear how to write down a good probabilistic model – and not necessarily obvious what the defining traits of a "good" model would even be. Some guiding practical questions include:

1. Should the environment be represented as objects, primitive geometry, images, language, or something else?

2. Should the model be able to readily sample new environments?

3. Should the model be able to explain environments, e.g. by providing quantitative density estimates or proposals over underlying structure?

Figure 1-1: Examples of scenes generated by the tools developed in this thesis – in this case, using a *spatial scene grammar* describing the arrangements of objects on a restaurant table. (See Section 2.6.4 for details.)

4. If the model has parameters, how can those be determined from data?

Depending on the application, one may have different priorities that influence model design choices. In our case, one of our primary motivations is in using this model as design and verification tool for robots: given the task of designing a dish-washing robot, we'd like to use our model to try to understand how well a prototype robot might work in diverse real kitchens. At design time, we might gather real-world data taken from real kitchens, and use that data to construct our model. During development, we would like to at least be able to draw samples from our model and test our robot in simulation – ideally, we'd be able to use structure in our model to enable more principled control synthesis, too. And finally, during deployment, we'd like to be able to check that the environments we encounter really do follow the model.

To meet these requirements, we believe we should **model environments at an object level** (for compatibility with simulation) with a **minimally complex parametric model** (for data efficiency and explain-ability). This model needs to

be able to be **generate realistic sample environments** while also being **possible to invert** (in order to explain a given scene and to fit model parameters). This is a difficult set of requirements to meet! Even if we assume that the world is composed of rigid objects with known shapes, different worlds will contain different *numbers and classes* of objects in different configurations, where the presence, types, and pose of those objects may covary. This joint continuous-and-discrete structure is the crux of this modeling problem, and trying to tackle it has encouraged the development of the tools and techniques reported in this thesis.

1.1 Scene Grammars, Trees, and Parsing

A *scene grammar* is a probabilistic procedural model that specifies a recipe for generating scenes as a list of *symbols* (or "object types") and *production rules*. Equivalently, a scene grammar specifies that all scenes have some underlying common tree-structured relationships among objects, and that scenes can be generated by sampling those trees starting from their corresponding root objects.[1] (See Figures 1-1 and 1-2 for a few visual examples.)

A scene grammar can be designed to capture a wide variety of relationships between objects in a scene, and excels at capturing hierarchical structure. For example, one could describe the generation of a room by describing how the room generates a random set of furniture within it at random poses on the floor, and each piece of furniture in turn produces a random amount of clutter within or on top of it. In this description, the room, furniture, and clutter are the object types in the grammar, and furniture-goes-in-room and clutter-goes-on-furniture are examples of production rules. In such a grammar, the existence and poses of each object depends only on the existence and pose of their parent object: this natural expression of conditional independence is a great feature of scene grammars that we will exploit throughout this thesis.

[1] This concept is closely related to formal grammars: a scene grammar is a context-free attribute grammar over a set of object types, where the attributes are poses. We explore this connection a little more in Chapter 2.

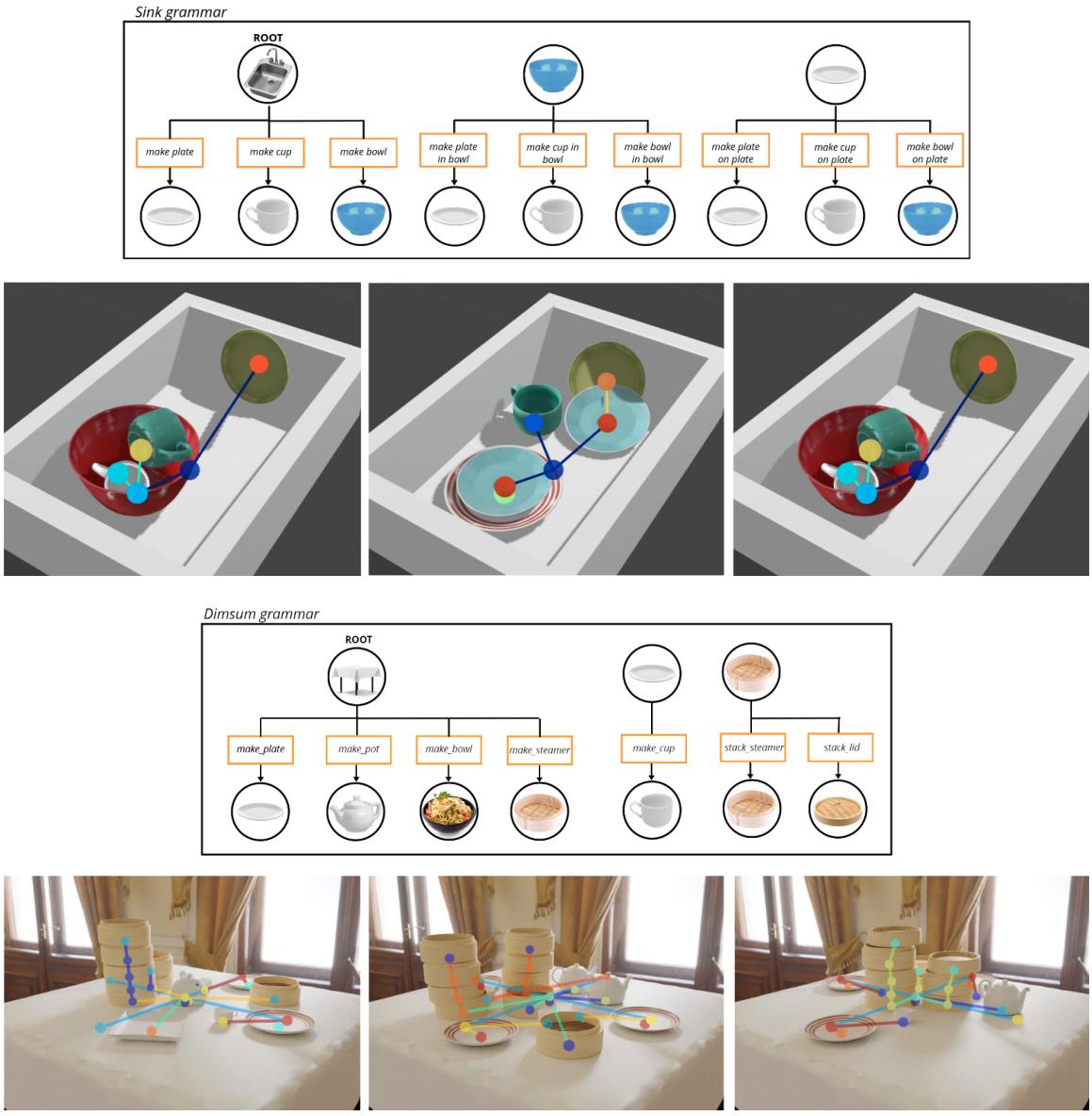


Figure 1-2: Sketches of scene grammars describing the occurrence and placement of dishware in a cluttered sink (**top**) or on a restaurant table (**bottom**), along with example scenes following each grammar. A random draw from a scene grammar activates random subsets of the available rules, which generates a scene with a random number of objects in random poses. Each example scene is annotated with a *scene tree* representing the hierarchy of relationships between objects corresponding to rules in the grammar.

A *scene tree* is a closely related annotation of a scene. Given a room with some set of furniture and clutter, a scene tree is a tree whose nodes are objects in the scene and edges are production rules from the grammar. Samples from a scene grammar are thus scene trees: a sample from the room grammar has the room as the scene tree root, connected by directed edges to each piece of furniture it contains; and, in turn, any clutter produced by a given piece of furniture will be placed in the tree under the furniture from which it was generated. In this way, scene trees are the structure by which we can relate any given scene to a corresponding grammar: they encode by which rule and under which parent each object was produced, and thus make it possible to evaluate how well the given scene is explained by the grammar.

Unfortunately, scenes are typically observed as unordered sets of objects (and that's assuming someone has already solved the perception problem for us). Given a grammar and an observed scene, inferring the best scene tree that explains the objects in the scene with the grammar is the *scene parsing* problem. Parsing scene trees from scenes can be particularly hard: depending on the details of the grammar, any scene might be described by an exponential number of different scene trees, and finding "legal" trees that follow all of the grammar rules may require carefully grouping objects in specific ways. Even worse, some grammars may even involve object types that aren't observable at all: a grammar might describe the production of a cluster of objects on a table, where the "cluster center" is itself an object in the grammar that can not be directly observed (and must instead be inferred). In these cases, scene parsing requires inferring the existence *and pose* of those hidden object types, making the problem doubly difficult.

1.2 Organization of this Thesis

The primary contribution of this thesis is a detailed formulation of a particular class of scene grammars, which we refer to as *spatial scene grammars*, along with constrained sampling, scene parsing, and parameter estimation techniques that make it possible to apply this tool to a broad range of environments and problems.

Chapter 2 provides details on the formulation of *spatial scene grammars*. This chapter refines work from [4] into a unified grammar framework that is tightly integrated with novel parsing and parameter estimation techniques presented in subsequent chapters. This chapter includes detailed remarks on some of the major design decisions and difficulties in authoring scene grammars, including the important functional differences between constituency- and dependency-style grammars and the trade-off between model expressiveness and parsing difficulty. This chapter also describes how to perform *constrained* sampling from spatial scene grammars – that is, how to use a scene grammar to produce scenes that follow additional content or pose constraints.

Chapter 3 presents two novel optimization-based formulations for scene parsing. One technique sets up a mixed-integer optimization that simultaneously decides the scene tree structure and the pose of each active node in the scene. We also present a novel integer-optimization parser that produces a set of heuristic proposals for possible poses for each node type in the grammar, and searches for the optimal way to utilize those proposals to produce a valid scene parse. We benchmark these strategies against each other in terms of their solution quality and run-time scaling with respect to grammar and scene complexity.

Chapter 4 presents an approximate expectation-maximization (EM) framework for grammar parameter estimation that generalizes the concepts from [4], along with experiments evaluating its effectiveness.

Chapter 5 discusses pilot experiments performed in the course of this thesis exploring how to close the gap between our object-focused grammars and the unlabeled images we get from our robots. This chapter also discusses a few broad ideas for future work, including possible downstream applications in robotics and alternative procedural model formulations.

1.3 Related Work

This thesis is focused on the problem of capturing *distributions over worlds* from the perspective of a roboticist. Tackling this problem requires making two major design choices in sequence:

1. What representation do we choose for a "world"?

2. How do we parameterize a distribution over that representation?

1.3.1 Choice of representation

The first of these decisions is enormously consequential: what representation of worlds should we use? This decision declares the scope of our modeling problem.

The representation of choice for much of the machine learning and computer vision community is images (or other representations close to perception: voxel grids, point clouds, lightfields, or even language). These formats are appealing, as they are both tremendously rich – the space of all images could be argued to contain (an image of) every possible world – and they correspond one-to-one with the sensors we put on our robots. The past few years have seen a wave of popular and successful deep neural generative models that can capture detailed realistic distributions over images and text (e.g. [5, 6]). Related model architectures have been shown to be able to capture distributions over more geometric (and for us, often more useful) world representations like voxel grids [7], point clouds [8], and light fields [9, 10]. The cost of weaker structural assumptions is an enormous hunger for data, and significant unresolved questions concerning model correctness and explanability.

We instead choose to describe a world as a set of objects. Most robotics simulators (and a corresponding majority of internal robotics world models) rely strongly on breaking the world down into a set of objects with known (or parameterized varying) geometry, and using that geometry as an abstraction for reasoning about how the world can be interacted with, and how the world will evolve. In this way, moving from pixels to objects vastly simplifies the modeling problem by imposing physical

structure that we know to be present and common in the world.

1.3.2 Choice of model type

The second of these decisions is to decide what modeling strategy we'll use to describe our chosen representation of worlds (or to decide whether we need a model in the first place).

Do we really need a model?

In the modern data-rich era, a tempting approach is to avoid specifying a model at all. This strategy corresponds to a dramatically non-parametric distribution modeling paradigm: instead of specifying some structured model to capture a distribution over worlds, we instead collect a sufficiently large dataset of examples to use as a proxy. A natural drawback of this approach is that, as the domain of interest grows in dimensionality, the amount of data required to ensure good coverage grows exponentially – which may limit these approaches from scaling beyond "narrow" environments for single-application robots. Still, this nonparameteric, data-driven approach is an appealing research direction: for example, [11] illustrates that, even if an analysis like adversarial example search is treated as a search across a finite set of observed examples, there's room for clever algorithms to *efficiently utilize* those examples by reasoning about how each observation is related. In our case, we wish to truly reason about *novel* environments that our robots have never seen before – preferably in a data-efficient way – which, fundamentally, demands some form of model.

Types of models

A typical modeling approach would instead be to make assumptions about the rules by which the world is assembled – whether by using highly parameterized and flexible neural approaches (as discussed above), or more classic symbolic-AI approaches. Naturally, there is a long and rich history of probabilistic scene modeling using such structured models. L-systems [12] and shape grammars [13] literally describe the

shape of objects by using rules to describe how to "grow" the object out of primitive geometry. Scene grammars instead describe the scene at the level of objects, though the amount of detail they aim to capture can vary: [14–16] all describe scenes from objects all the way down to the polygon, line segment, or pixel level, while [17–20] use objects as their terminal level of detail. One step above a scene grammar would be to instead describe a distribution over scene *graphs*, in which an object can have relationships to more than one other object. These models are more complex to sample from and do inference over, although it has been shown to be possible with MCMC methods [21]). Scene graph models are well suited to producing variations on existing content [22] and providing rich priors and structures for object-level, pixel-level, and point cloud perception [23, 24]. Even richer models – whether based on probabilistic programming [25–27], autoregressive neural models [28, 29], or hybrid procedural models with deep learning components (e.g. [28, 30]) – can in principle capture *any* distribution over *any* scenes, at the cost of significant model complexity, extremely difficult inference problems, and for deep neural models, data inefficiency.

In this work, we focus on using scene grammars as a modeling tool. We have found that scene grammars strike an appealing balance between expressiveness, interpretability, and structure. Scene grammars are founded on the strong structural assumption that the world can be assembled by following a set of simple context-free rules. We will show that we can take advantage of this simple structure to formulate novel principled optimization and inference algorithms that reason about a wide range of worlds. This simplicity naturally comes at the expensive of expressive ability – but as our results demonstrate, we can still use scene grammars to describe a range of complex robotics-relevant scenes. Scene grammars are easy to design (unlike many shape grammars) and sample from (unlike some scene graph models that only provide a joint density over scenes). It is, however, not trivial to invert or "parse" a scene using a scene grammar model; nor is it clear how best to fit the parameters of a scene grammar model to observed data. We provide novel solution strategies for these problems as contributions of this thesis.

1.3.3 Robotics applications

While large-scale applications of these tools is beyond the scope of this thesis, it is worth surveying where distributional world models fit into robotics, as it informs what properties and traits we should prioritize when we are designing our model.

Both robot policies and vision systems have been demonstrated to be trainable exclusively in sim if the sim-to-real gap is sufficiently small [31–33]. In particular, BayesSim [34] demonstrates a method for explicitly aligning the distribution of simulated rollouts to real ones (in the space of friction and mass parameters) to get better sim-to-real alignment. Many of these techniques are inspired by sim-to-real problems in reinforcement learning of dynamic tasks, and hence focus on the alignment of *dynamics* between simulation and reality. However, work more similar to ours focused on the alignment of scene *content* has also begun to accelerate. MetaSim [18] trains a neural distribution transformer to adjust scene object arrangements to make them more realistic (as measured by a distribution distance in a deep image embedding space), and experimentally demonstrates that performing that alignment quantitatively improves downstream task performance. [20] extends that system to better align the number of objects in addition to their poses. [35] corroborates that it's important to get the scene contents to be realistically distributed to achieve good transfer. While there is a growing body of realistic environments to use for training agents (e.g. [36, 37]), they can be limited in scope and diversity. The work in this thesis is motivated by a desire to generate rich data in the style of those datasets, but with the ability to customize the model to describe scenes of interest, generate arbitrarily many new environments, and reason about how new environments fit the model. These sorts of procedural scene generators are becoming increasingly popular: see [35] and [38] for two recent and powerful programmable scene generation pipelines.

Having a good model over environments is also critically useful for analyzing and improving the robustness of a given system. [39], for example, provides rate estimates for rare events by carefully sampled simulations, but requires a prior distribution

over worlds that forms the basis for those rate estimates. These world models can be powerful sanity checks and regularizers for systems at runtime. [40] shows that even just using an autoencoder as a world model can provide reliable anomaly and novelty detection abilities. Scene grammars and trees have been shown to be widely useful as a prior to improve perception or planning performance: for example, [23] uses a graph structure over body parts to ensure human part detections are at reasonable relative poses, and [41] uses learned information about typical relative placements to improve the reliability of a pick-and-place system. For a world model to be useful in those situations, it needs to not only provide new world samples, but also reason about how well hypothetical or observed worlds fit the model – which is exactly what our parsing procedure is designed to enable.

Chapter 2

Formulation of Spatial Scene Grammars

2.1 Introduction

This chapter provides a precise formulation of a class of procedural models we call *spatial scene grammars*. Spatial scene grammars are a subclass of *scene grammars*, which are themselves a class of probabilistic procedural model. Scene grammars describe distributions over sets of objects by declaring a set of probabilistic generative *rules* that can be applied to grow a scene out of parts. Spatial scene grammars focus on describing scenes composed of rigid objects from a finite set of object types with a-priori known geometry. Their random variables describe the discrete number of objects, the discrete type of each object, and the continuous pose of each object. This restriction in scope allows us to be particular about the kinds of distributions over object count, type, and pose that are supported – and to ensure that we're treating *spatial* relationships between object poses particularly carefully.

All procedural models face a fundamental trade-off between their expressive power and the ease with which they can be inverted. Stated another way: the more expressive and complex any given procedural model becomes, the harder it is to "invert" or otherwise do inference over the latent variables of that model. In the case of scene grammars, we refer to this model inversion process as *scene parsing*, by which process

we reconstruct a scene tree that explains how the grammar rules created an observed set of objects. As a core contribution of this thesis, we will show over the course of this chapter and Chapter 3 that by carefully restricting the kinds of spatial relationships described by a scene grammar to exactly those described in our spatial scene grammar formulation, we can formulate scene parsing as a tractable mixed-integer optimization problem.

2.2 Related Work

As discussed in Chapter 1, scene grammars (and the related and sometimes overlapping concepts of L-systems, shape grammars, and scene graphs) have a rich history in the fields of computer vision and computer graphics, but are just starting to appear in robotics applications. The notion of using context-free grammar (CFG) rules to build objects or scenes by parts dates back at least as far as L-systems, which combine a standard CFG over strings with a system for rendering generated strings into natural (usually branching) structures like trees [12]. This logic can be applied to building up shapes by grammar rules via shape grammars [13, 42], which have been applied to describe shapes of buildings [43–45] and general triangulated polygons [46]. These shape grammar formulations often "grow" geometry from atomic shapes or drawing rules, which lends them incredible expressive power at the cost of complexity and control difficulty. Indeed, that these shape grammars can be over-expressive and unintuitive has motivated work on guided sampling [45] and grammar inference [47–49].

We instead focus on modeling scenes by top-down design at the level of objects and their spatial relationships. A significant amount of work in this area uses grammars over scene parts either as models for scene understanding, or as regularizers for perception tasks. [14], for example, defines a simple grammar that describes a scene as being composed of a random number of spatially patterned parts, each of which can produce additional parts within or relative to itself, or terminate and produce a rectangular surface. [50], likewise, advocates for the general application of AND-OR grammars that terminate in image patches as a model for describing im-

ages in general. This line of work has evolved to include more complex grammars that separately model foreground objects with bounding boxes [15], group objects by functionality rather than spatial relationships [51], or consider additional simultaneous attribute and dependency structure to simultaneously capture constituency, relative pose, and style information [23]. These works have tended to include support for additional lateral, non-context-free relationships between objects in the scene for additional expressive power. In a separate line of work, [16] describes a similarly-structured probabilistic CFG over image parts and their 2D poses, but restricts the poses of objects to fall on a discrete grid as a prerequisite for their inference strategy. These grammars all ultimately terminate by generating image patches, lines, or other image features.

A more limited body of work – but one that shares many of the same underlying grammar tools and techniques – instead focuses more closely on capturing geometric structure. [17] learns a scene grammar from 3D datasets annotated with hierarchies that breaks down scenes constituently from rooms, to objects, to oversegmented geometric regions. [18] utilizes a CFG-structured scene grammar to generate their scenes, which they post-process with a neural distribution transformer to make more realistic; [20] demonstrates that they can use a recurrent network to improve the distribution modeling ability of this grammar. Beyond grammars, [22] reasons about spatial relationships between objects and object classes using a scene graph structure. [19] models a scene with an underlying scene tree similar in concept to our own – but their scene trees encode a specific set of *physical contact and support relationships* in a scene rather than generic grammar production rules. [19] also illustrates that they can solve the parsing problem for this scene tree representation by using a combination of importance-resampling and MCMC, in contrast to our all-at-once mixed-integer optimization approach.

Our grammar formulation is strongly inspired by the common grammar structures shared among many of these works. Our spatial scene grammar formulation is fundamentally a probabilistic CFG, as in [15–18] and others. We do not directly support lateral non-context-free relationships between objects in the grammar like

those in [15, 23], as they greatly complicate the scene parsing processes; however, we do support such arbitrary *constraints* over tree structure and poses that are handled during a conditioned sampling process. (This constraint specification is spiritually similar to the one used in [38].) We use a variation on the AND-OR-SET discrete logic from [15] [23]. While many of these works describe relative pose distributions with Normal distributions (or more – e.g. [17] captures attribute relationships with kernel density estimation), our use of Bingham distributions[1] in a scene grammar is novel to the best of our knowledge.

2.3 Formulation

A spatial scene grammar is a stochastic, attributed, parameterized context-free grammar $\{\mathcal{N}, \mathcal{R}, P_\theta^{\mathcal{N}}, P_\theta^{\mathcal{R}}, N_{root}, x_{root}\}$. This grammar is defined over node types $\mathcal{N}$.[2] Each node type $N \in \mathcal{N}$ has a continuous pose attribute $N.x \in SE(3)$[3], and associated geometry $N.geometry$ that it will produce in the final scene; this geometry can be empty if the node doesn't produce an observable artifact. The set of production rules $\mathcal{R}$ represent how a node (i.e. an instantiated symbol) can produce a child node. The rules are organized by their parent node: each node type $N \in \mathcal{N}$ has an associated list of production rules $N.\mathcal{R} = [R_0, ..., R_M]$, each of which describes the possible production of a single additional child object.

In this definition, a node can simultaneously activate multiple rules to generate multiple children. Specifically, a parent node n^p of type N activates a subset of its rules $\mathcal{R}_{active} \subset N.\mathcal{R}$ following a *discrete* distribution over n_p's rules (parameterized by θ):

$$P_\theta^N(\mathcal{R}_{active}).$$

When active, a given rule R produces a child node n^c of fixed type $R.N$ with its pose drawn from a *continuous* distribution (also parameterized by θ) that is condi-

[1] For an overview of Bingham distributions as applied to robotics, [52] is an excellent reference.

[2] These node types are equivalent to "symbols" in a formal grammar. We use "node type" instead to clarify the connection to scene trees (which are graphs over nodes of these types).

[3] We use notation $A.B$ to indicate B is a property of A.

tioned on the parent's pose

$$P_\theta^R(n^c.x|n^p.x).$$

This is expressive enough to describe, for example, a child whose pose is a distribution in the parent's frame of reference.

A draw from the grammar is a tree T of nodes, each having a concrete spatial pose. The tree is stored as a set of parent-ruleset pairs $T = \{n^p, \mathcal{R}_{active}\}$; for a given node n^p, $\mathcal{R}_{active}$ is n^p's set of active rules, which serves as the list of outgoing edges to child nodes $R.n^c$ for each rule $R \in \mathcal{R}_{active}$. A tree is drawn from the grammar by initializing a tree with a root node of type N_{root} at predetermined pose x_{root} and recursively sampling children from unexpanded nodes until no unexpanded nodes remain (see Algorithm 1).

An observed scene o is constructed by collecting the geometry-producing nodes from T. Critically, the observed scene o does not always reveal the full structure of the tree, because only those nodes that produce geometry are observable, and there may be multiple trees that produce the same observations. (For example, the centers of clusters of objects may be represented with nodes, but only the actual objects in the cluster show up in the observed scene.) Each node N of type $\mathcal{N}$ deterministically produces geometry $\mathcal{N}.geometry$ at pose $N.x$.

The joint probability of a sampled tree T is

$$P(T) = \prod_{n^p, \mathcal{R}_{active} \in T} P_\theta^{N^p}(\mathcal{R}_{active}|n^p) \prod_{R \in \mathcal{R}_{active}} P_\theta^R(R.n^c.x|n^p.x), \tag{2.1}$$

where the outer product iterates over parent nodes n^p and their actives rule sets $\mathcal{R}_{active}$ implied by their successors in the tree, where N^p indicates the node type of corresponding parent node n^p. The inner product iterates over the individual active rules R and their produced child nodes n^c associated with that parent.

The relationship between a parent node and its children is factored into the discrete choice (corresponding to P_θ^N) made by the parent to produce that child set, and the continuous choice (corresponding to P_θ^R) made to place each child at its pose relative to the parent. In the following sections, we enumerate additional structure

Algorithm 1: Forward sampling a scene from a grammar.

Input : a root node *root*
Output: a sampled scene tree

```
/* Build scene-tree top-down, starting from the supplied root
   node.                                                      */
node_queue = [root]
tree = Tree([root])
while len(node_queue) ≥ 0 do
    parent = node_queue.pop()
    /* Sample what this parent is going to produce.          */
    rules = parent.sample_rules()
    for rule in rules do
        /* Sample the pose of each child being produced.     */
        child = rule.sample_child()
        node_queue.push(child)
        tree.add_node(child)
        tree.add_edge(parent, child)
/* Return only those nodes that produce scene geometry.      */
return [node for node in tree if node.has_geometry]
```

and restrictions we can impose on these distributions to enable our ultimate goal of making posterior scene tree parsing more tractable.

2.3.1 Node types to capture common discrete relationships

The kinds of children a parent creates are chosen by sampling a set of production rules from parent node. In general, this is a draw from a joint distribution over $M = |N.\mathcal{R}|$ binary variables representing the activation of each possible rule, requiring 2^M parameters to describe. It is particularly popular in the scene grammar literature to instead describe these relationships with combinations of simple AND and OR rules in the style of AND-OR trees [15, 23]. Inspired by this approach, we define a small set of parent node types that represent more structured patterns of rule activation that can be described with vastly fewer parameters:

1. **AND** node type: The parent node provides a list of production rules, all of which are activated every time. This requires 0 parameters.

XYZ Rules

Name	Child translation expression	Log-density expression	Parameters
Fixed offset	$n^c.t = n^p.t + t_o$	0.	$t_o \in R^3$
World frame Normal	$n^c.t \sim Normal(\mu, \Sigma)$	$-0.5\delta^T\Sigma\delta - F(\Sigma)$ $\delta = n^c.t - \mu$	$\mu \in \mathbb{R}^3, \Sigma \in \mathbb{R}^3 > 0$
World frame Normal offset	$n^c.t \sim n^p.t + Normal(\mu, \Sigma)$	$-0.5\delta^T\Sigma\delta - F(\Sigma)$ $\delta = n^c.t - n^p.t - \mu$	$\mu \in \mathbb{R}^3, \Sigma \in \mathbb{R}^3 > 0$
Parent frame Normal offset	$n^c.t \sim n^p.t + n^p.r \times Normal(\mu, \Sigma)$	$-0.5\delta^T\Sigma\delta - F(\Sigma)$ $\delta = n^p.r^T(n^c.t - n^p.t - \mu)$	$\mu \in \mathbb{R}^3, \Sigma \in \mathbb{R}^3 > 0$

Rotation Rules

Name	Child rotation expression	Log-density expression	Parameters
Uniform rotation	$n^c.r \sim Uniform(SO(3))$	0.	None.
Fixed offset	$n^c.r = n^p.r \times R_o$	$-\log(\pi^2)$	$R_o \in SO(3)$
World frame Bingham distribution	$n^c.r \sim Bingham(M, Z)$	tr $ZM^Tqq^TM - F(Z)$ $qq^t \leftarrow n^c.r$ (Sec. 3.4.3)	$M \in \mathbb{R}^{4\times4}, Z \in \mathbb{R}^4$
Parent frame Bingham distribution	$n^c.r \sim n^p.r \times Bingham(M, Z)$	tr $ZM^Tqq^TM - F(Z)$ $qq^T \leftarrow (n^p.r)^T(n^c.r)$ (Sec. 3.4.3)	$M \in \mathbb{R}^{4\times4}, Z \in \mathbb{R}^4$

Table 2.1: The set of rule types supported by our grammar, along with their expression for forward sampling and log-density evaluation. Each rule type describes a parameterized, stochastic distribution on the translation and rotation of a child node n^c conditioned on the pose of its parent node n^p, in terms of each of node's translation t and rotation r components. Distinction is made between the different frames in which translational and rotational offsets are expressed, as offsets expressed in the parent's frame are bilinear in the participating pose variables and require special handling during scene parsing in Chapter 3. Terms $F(\Sigma)$ and $F(Z)$ represent normalizers that are a function of the indicated distribution parameters.

2. **OR** node type: The parent node provides a list of production rules, one of which is chosen at random to be activated. This requires $M - 1$ parameters.

3. **INDEPENDENT SET** node type: The parent node provides a list of production rules, each of which is activated randomly, independent of the other rules. This requires M parameters.

4. **REPEATING SET** node type: The parent node provides a single production rule, which is activated $k \sim Categorical(1...M_{rep})$ times. This requires M_{rep} parameters.

This set of node types is complete, in that any pattern of child activation can be implemented by chaining AND and OR nodes in an appropriate pattern.

2.3.2 Rule types to capture common continuous relationships

Each production rule R under parent n^p that is activated produces a new child node n^c with random pose x drawn from a rule-specific conditional distribution $p_\theta^R(n^c.x|n^p.x)$. We separate this spatial relationship into conditional distributions over the translational and rotational components, which can frequently be modeled as being independent from one another. The separate translation and rotation parts of the pose are referred to as $n.t$ and $n.r$ respectively.

In Table 2.1, we provide a detailed list of distinct kinds of conditional distributions over translation and rotation. We focus most strongly on cases where translation distributions can be captured using Normal distributions, and rotational distributions can be captured using Bingham distributions.[4] Further, we distinguish between different of expressing the relationships between objects: note that expressing either translational or rotational pose distributions in the parent node frame (i.e. the "Parent frame" rules) introduces bilinear relationships between the pose variables, which must be handled as special cases in the mixed-integer convex parsing procedure in Chapter 3.

This particular set of rule types has arisen from efforts to apply this grammar formulation to a wide variety of scenes; empirically, each of these rule types is useful in different circumstances, and trades off expressiveness with complexity. In our implementation, these node rule types are implemented with a class hierarchy making it easy to slot in new node types translational and rotational rule types – or, more broadly, new characterizations of $p_R(n^c.x|n^p.x)$. In practice, we employ a handful of closely related rule types to make it easier to express prismatic, planar, and revolute behavior equivalent to setting certain covariance terms to zero.

[4]Bingham distributions are an analogue of Normal distributions over the unit sphere appropriate for describing distributions over unit quaternions [53]. Bingham distributions are parameterized by an orthogonal orientation matrix $M \in \mathbb{R}^{4\times4}$ and a concentration vector $Z \in \mathbb{R}^4$. A helpful overview of this distribution type is provided in [54]. For a more detailed view of applications of this distribution type to robotics, refer to [52].

2.4 Specifying and Enforcing Constraints

While the context-free grammar structure we enforce is well-suited to describing pairwise object relationships, it is fundamentally incapable of expressing scene-wide constraints like "ensure the scene has exactly N objects" or "ensure all objects are in a non-penetrating configuration." We instead allow the user to express constraints directly so that we can consider them during the scene sampling process.

In addition to a grammar specification, a user can provide a list of constraints

$$\mathcal{C} = \{C(T) : T \to \mathbb{R}^n \geq 0\}.$$

Each constraint is a function $C(T)$ mapping a scene tree T to an n-dimensional vector, which is constrained to be (without loss of generality) strictly elementwise positive. We further distinguish between two classes of constraints that a user might specify: *pose* constraints, which are only functions of the poses of nodes in T, and *structural* constraints, which might be a function of the number of types of nodes in T. We use $\mathcal{C}(T) \geq 0$ as shorthand to refer to evaluation of the complete constraint set at once; and $\mathcal{C}^P$ and $\mathcal{C}^S$ to refer to the subsets of pose and structural constraints respectively.

Sampling scenes given this constraint set is equivalent to sampling from the conditional distribution $P(T|C(T) \geq 0)$. Unfortunately, while sampling unconditioned trees $P(T)$ is easy (see Algorithm 1), this conditional sampling problem is more difficult. Rejection might work for easy-to-satisfy constraint sets, but as the space of scene trees is naturally very high-dimensional, rejection sampling can quickly become an untenable strategy.[5] Pose constraints in particular can be very difficult to satisfy: a very common pose constraint is to enforce non-penetration of all objects, which becomes hard to satisfy when objects become sufficiently cluttered. We utilize a combination of techniques, depending on the context, to address these sampling difficulties:

[5] [38] successfully uses rejection sampling to handle similarly-posed constraints in their programmable scene generation pipeline; however, they carefully design their sampling space using domain-specific knowledge to keep their sampling complexity in check.

1. To satisfy structural constraints, we rely on rejection sampling. While we have explored guided resampling of subtrees (based on [45]) with some success, the structural constraints we experiment with here are simple enough that rejection sampling is sufficient.

2. For differentiable pose constraints of middling complexity, we use a more efficient Hamiltonian Monte-Carlo (HMC) sampling algorithm to find satisfying object configurations given a fixed tree structure.[6]

3. For very hard-to-satisfy pose constraints, we resort to a projection approach in which we sample a tree from $P(T)$ and then solve a nonlinear optimization to find the closest tree configuration that satisfies the constraints.

In practice, to produce a diverse set of trees that satisfy a constraint set, we sample a diverse population of trees that satisfy structural constraints by rejection sampling (1), and then apply technique (2) or (3) to transform those samples into ones that additionally satisfy any pose constraints.

2.4.1 HMC for sampling from pose constraints

[55] demonstrates that it's possible to use Hamiltonian Monte-Carlo [56], a relatively sample-efficient gradient-based variant of MCMC, to find object configurations that satisfy physical support and non-penetration constraints; we reproduce their approach here. Given a sampled tree $T_0 \sim P(T)$ that satisfies any structural constraints but may or may not satisfy pose constraints $\mathcal{C}^P$, we sample a sequence of trees $T_k \sim P^{HMC}(T|\mathcal{C}^P)$. Like all MCMC techniques, HMC produces a sequence of samples from a Markov process whose stationary distribution is designed to match a distribution of interest. Here, we craft a special distribution that conditions $P(T)$ to satisfy the constraints by the addition of constraint penalty terms:

$$P^{HMC}(T|\mathcal{C}^P) = P(T) * \prod_{C \in \mathcal{C}^P} \mathcal{N}[0, \epsilon](\min(C(T), 0)).$$

[6] In the case that the constraints are relatively easy to satisfy but not differentiable, one might consider a conceptually similar, but less sample-efficient gradient-free MCMC routine instead.

Here, violation of each constraint C is penalized with a Normal distribution with small variance ϵ. After a burn-in period, samples T_k generated from this process follow the constraint-conditioned distribution $P(T|\mathcal{C})$, albeit all having of the same *structure* as T_0.

In practice, we use the No-U-Turn HMC Sampler [57] as implemented in the Pyro probabilistic programming language [58] to sample from this distribution. We use a constraint penalty $\epsilon = 10^{-3}$, which has empirically proven tight enough to produce good samples.

2.4.2 Projection-based constraint resolution

Given a sampled tree $T_0 \sim P(T)$ that satisfies any structural constraints but may or may not satisfy pose constraints $\mathcal{C}^P$, we formulate a nonlinear optimization to find the closest tree with the same structure, but possibly different node poses that satisfies a constraint set

$$T^* = \text{Proj}(T_0) = \arg\min_T, ||T.X - T_0.X||_2^2$$
$$s.t. \quad \mathcal{C}^P(T) \geq 0.$$

Here, the optimization holds the structure of T fixed while searching over the poses of its nodes, penalizing the squared norm difference between the vectorized node poses against the original tree T_0.[7] In practice, we formulate and solve this optimization with Drake [59] and SNOPT [60].

Importantly, this technique introduces bias when used to produce samples satisfying the constraint set. While the population of trees T_0 will follow $P(T)$, the distribution of the population of $\text{Proj}(T_0)$ may be arbitrarily warped by the projection operation, as it will move all trees violating the constraints to the boundary of constraint satisfaction. This is in contrast to the HMC procedure, which (after

[7]This includes penalizing a difference between rotations as an elementwise distance; in our implementation, this distance is an L2 norm between quaternions. While it would be more technically correct to penalize this as an angular distance, we have not found that it unduly influences the solution quality.

sufficient mixing) will produce samples from $P^{HMC}(T)$, which approximates the intended distribution. Worse still, this nonlinear projection is generally not invertible (as it might collapse large regions of infeasible space onto single points on the boundary of feasible space in a many-to-one fashion), and so has singular Jacobian that makes accounting for this distribution transformation impossible. However, in practice, this technique is sometimes the only thing that is practically possible – and so can be used to post-process samples to follow hard-to-otherwise-capture constraints, like non-penetration in highly cluttered environments.

2.5 Scene Generation Tools

We have implemented this grammar and constraint formulation in a Python codebase, and this system is tightly integrated with a handful of additional tools that we use to produce and manipulate environments, generate geometry for use in our grammars, and produce high-quality renders of sampled scenes. We detail these tools here.

2.5.1 Spatial scene grammar codebase

The spatial scene grammar architecture we describe in this chapter; the constraint specification interface; the sampling, parsing, and parameter estimation routines; and examples are implemented in a unified Python codebase available at `https://github.com/gizatt/spatial_scene_grammars`. This codebase is tightly integrated with Pytorch [61] and Pyro [58] (for autograd, distribution types, and implementations of common inference routines), Drake [59] (for simulation, visualization, and mathematical programming abstraction), and Meshcat [62] (as a visualization tool integrated with Drake).

2.5.2 Automatic mesh generation tool

Most simulators prefer simulating *convex* collision geometry, but many objects available online (from e.g. Ignition Robotics [63] or TurboSquid [64]) have high-polygon-

count, non-convex geometry by default. To bridge this gap, we created a tool to perform automatic convex decompositions of such nonconvex meshes using Trimesh [65] and V-HACD [66], and to produce a standard SDF [67] text-file output that accumulates the resulting convex pieces into one simulate-able package. This tool is available at `https://github.com/gizatt/convex_decomp_to_sdf`.

2.5.3 Blender rendering over IPC

Decades of research in computer graphics (and decades of parallel development in computer animation, video games, etc) have fueled the creation of exceptionally powerful rendering tools that, in the right hands, can produce photorealistic images. These tools are being utilized by a growing body of work in computer vision and robotics to produce exceptionally realistic training data, whether for training object perception systems (e.g. DOPE [32]) or creating realistic depth data (e.g. Blensor [68]).

Blender [69] is a particularly popular 3D modeling and rendering tool (under an open source GPL license) that is compatible with a broad swath of popular 3D model and texture formats; includes a powerful and configurable (optionally GPU-accelerated) raytracing renderer; and, critically, is easily scriptable via a Python interface. During the course of this thesis work, we developed a tool to make it easy to use Blender to render arbitrary collections of geometry (typically extracted from a simulator) over an interprocess communication link. In this tool, a Blender process hosts a server with which a client can establish a ZMQ socket connection; this connection then allows the client to load and manipulate geometry in the Blender scene and render images to disk. This separation keeps the client from having to match the Python version and libraries expected by Blender, which can be difficult: the client could, in concept, even be written in a completely different language. In addition, we provide a Drake system block that automatically renders any geometry in a Drake simulation scene every simulation tick using this interface. This tool is available at `https://github.com/gizatt/blender_server`.

See Figures 1-1, 2-6 and 4-2 for some examples rendered using this tool.

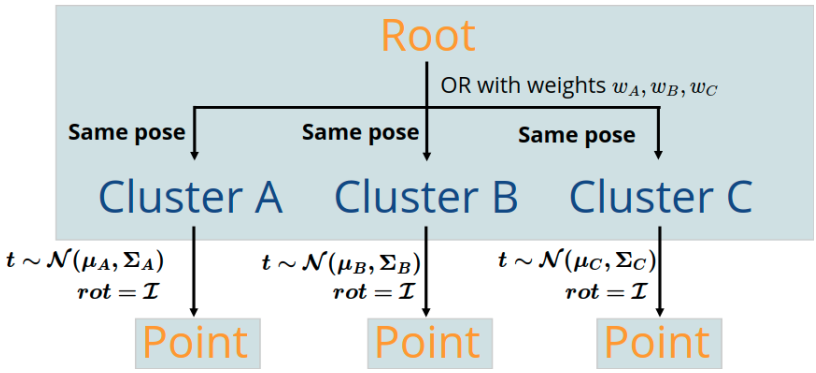


Figure 2-1: Grammar structure implementing a 3-component Gaussian mixture model (GMM) whose parameters capture the mixture weights w_A, w_B, w_C, mixture means μ_A, μ_B, μ_C, and mixture covariances $\Sigma_A, \Sigma_B, \Sigma_C$.

2.6 Example Grammars

Despite the careful limitations we impose on continuous distribution specifications, this framework can be used to capture a wide variety of scenes. This grammar framework can be viewed as a generalized, multi-level Gaussian mixture model – that is, one where the number of modes and their hierarchy is itself stochastic – with special handling dedicated to describing distributions of 3D poses $\in SE(3)$. As a result, we inherit similar expressiveness: we can combine our primitive distributions together to build arbitrarily complex distributions over poses. In this section, we detail a number of grammars that describe a diverse set of scenes. These grammars are referred to throughout this thesis.

2.6.1 Gaussian mixture model grammar

Figure 2-1 illustrates the structure of a small grammar that implements a three-mode Gaussian mixture model. This grammar illustrates an important way that our intentionally limited grammar formulation can be used to describe arbitrarily complicated spatial distributions by combining Gaussian distributions in a mixture model. We will use this grammar in Chapter 4 to compare the performance of our parameter estimation algorithm against a baseline parameter estimation algorithm (vanilla EM) that is typically applied to GMMs.

Grammar details

The GMM grammar is defined over node types

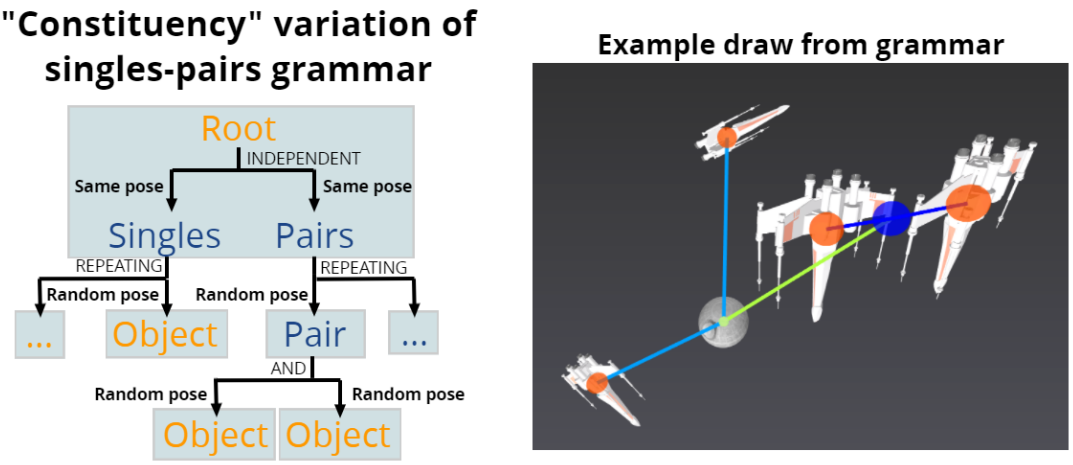
$$\mathcal{N} = \{\textsc{Root}, \textsc{Point}\}.$$

The ROOT and POINT both have associated (trivial) geometry, or otherwise have directly observable poses. ROOT can generate POINT by exercising one of three rules, each of which samples the POINT's translation from a world-frame Normal distribution, and its rotation as the identity matrix. Each of these three rules have their own parameter sets, so this grammar describes the production of a single POINT at a time as drawn from a 3-mode Gaussian mixture model.

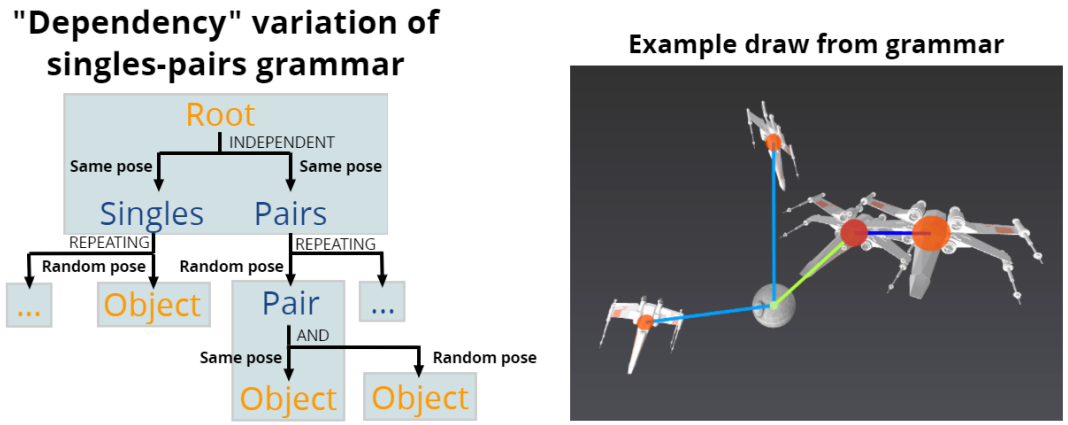
2.6.2 Singles-pairs grammar

This grammar describes a set of oriented objects that appear either independently ("singles") or in pairs with related pose ("pairs"). As physical analogue, we consider modeling that aircraft might appear in the air either on their own, or in a formation as a pair, where the two aircraft in the formation tend to be close in position and orientation.

We provide two closely-related grammars to describe this class of scenes, as illustrated in Figure 2-2. We broadly categorize the approaches one can take into "constituency" or "dependency" grammars. In the singles-pairs constituency grammar, objects in a pair are constituents of the pair itself, and the pair itself is an actual node in the scene tree. The dependency grammar variation, on the other hand, relates objects directly to each other without introducing a new "cluster center" node to represent the intermediate abstract grouping. In both cases, a pair of objects covary in both their translation and rotation, such that the pose of one member of a pair could be described with a distributions expressed in the frame of the other (using "Parent frame Normal offset" and "Parent frame Bingham distribution" rules from Table 2.1).



(a)



(b)

Figure 2-2: These two grammar variations for the singles-pairs environment describe very similar distributions over objects that can appear either on their own or in spatially correlated pairs. In each variation, the Pair node has a different spatial meaning. In the constituency variation, the objects in the pair are both randomly offset from a Pair node that has no geometry of its own. In the dependency variation, the Pair node is at the same pose as one of the objects, and the second object is randomly offset from the Pair (or, equivalently stated, is randomly offset from the first object). We will show in Chapter 3 that the constituency variation is significantly harder to parse, as it requires guessing the unknown pose of each Pair node.

Grammar details

Both singles-pairs grammar variations are defined over node types

$$\mathcal{N} = \{\text{Root}, \text{Singles}, \text{Pairs}, \text{Pair}, \text{Object}\}.$$

The Root and Object have associated geometry, or otherwise have directly observable poses.

- Root is an INDEPENDENT-SET node can chooses whether to produce each of Singles, Pairs with an independent coin flip. Each is produced at the same pose as the Root.
- Singles is a REPEATING-SET node that produces one or more Object nodes using world-frame Normal and Bingham offset rules.
- Pairs is a REPEATING-SET node that produces one or more Pair nodes using world-frame Normal and Bingham offset rules.
- Pair is an AND node that always produces two Object nodes.
  - **Constituency variation**: Pair node produces each Object using a parent-frame Normal and Bingham offset rule.
  - **Dependency variation**: Pair node produces one Object at the same pose as itself, and produces the other Object using a parent-frame Normal and Bingham offset rule. (This is equivalent to one Object being a "parent" of the other.)
- Object is a terminal node with associated geometry.

2.6.3 Cluttered sink grammar

This grammar provides a structured description of cluttered arrangements of objects in a sink. For motivation, consider the problem of verifying the performance of a robot meant to transfer objects from a sink into a dishwasher [70]. Such a robot may

be particularly sensitive to certain arrangements of objects, like objects packed into bowls or dishes tightly stacked on one another. We design a grammar that captures some of that structure, so that we might begin to quantify how often and with what spatial distributions those cases occur, and to detect those cases at runtime via scene parsing.

The high-level schematic structure of the sink grammar is illustrated in Figure 2-3, and example sinks annotated with scene trees from this grammar are illustrated in Figure 2-4. The intent of this sink grammar is to capture the way that objects might be randomly arranged within the sink, but that bowls and plates may additionally have a tendency to have objects placed within or on top of them, respectively. As a result, this grammar is based around a SINK nodes that produces a random number of intermediate OBJECT nodes. Each OBJECT node will specialize into one of the terminal node types with plate, cup, or bowl geometry; but it if specializes into a BOWL or PLATE, it may additionally produce objects inside of or on top of itself.

Grammar details

The sink grammar is defined over node types

$$\mathcal{N} = \{\text{SINK, OBJECT, BOWL, BOWLCONTENTS, PLATE, PLATECONTENTS,}$$
$$\text{TERMINALOBJECT, TERMINALBOWL, TERMINALPLATE, TERMINALCUP}\}.$$

The SINK, TERMINALBOWL, TERMINALPLATE, and TERMINALCUP types have associated geometry; all other types represent an object or group of objects of unresolved final type. (For example, a BOWL will produce both a TERMINALBOWL and BOWLCONTENTS.)

- SINK is a REPEATING-SET node that produces a random number of OBJECTS at random poses using world-frame Normal offset and Bingham rules. The sink has fixed a-priori geometry.[8]

[8] World-frame offset rules are used here as the Sink is the root node and is always produced at the same position; but parent-frame rules may be more appropriate if the Sink is to be embedded in a bigger grammar in which the rotation of the sink might vary.

- OBJECT is an OR node that specializes into a BOWL, PLATE, or TERMINALCUP at the same pose.
- BOWL is an AND node that produces a TERMINALBOWL and BOWLCONTENTS at the same pose.
- BOWLCONTENTS is a REPEATING-SET node that produces a random number of TERMINALOBJECTS[9] at random poses using parent-frame Normal and Bingham rules.
- PLATE is an AND node that produces a TERMINALPLATE and PLATECONTENTS at the same pose.
- PLATECONTENTS is a REPEATING-SET node that produces a random number of TERMINALOBJECTS at random poses on top of the plate using parent-frame Normal and Bingham rules.
- TERMINALOBJECT is an OR node that specializes into one of the terminal object types TERMINALPLATE, TERMINALBOWL, TERMINALCUP.
- The TERMINAL[*] types are terminal nodes that are associated with appropriate plate, cup, or bowl geometry.

Remarks on sampling difficulty

This grammar is at the extreme end of the spectrum of spatial clutter – samples from this grammar are virtually guaranteed to be in a deeply interpenetrating configurations. In fact, the samples this grammar produces are so cluttered that attempts to impose non-penetration constraints using each of the constrained sampling techniques from 2.4 usually fail. Naive samples from the grammar are extremely unlikely to satisfy pairwise nonpenetration constraints; HMC usually fails to adequately mix and resolve the constraint while staying within a reasonable tree configuration; and nonlinear projection frequently fails (due to numerical issues internal to the solver)

[9]Bowls and Plates produces TerminalObjects rather than Objects to remove recursion.

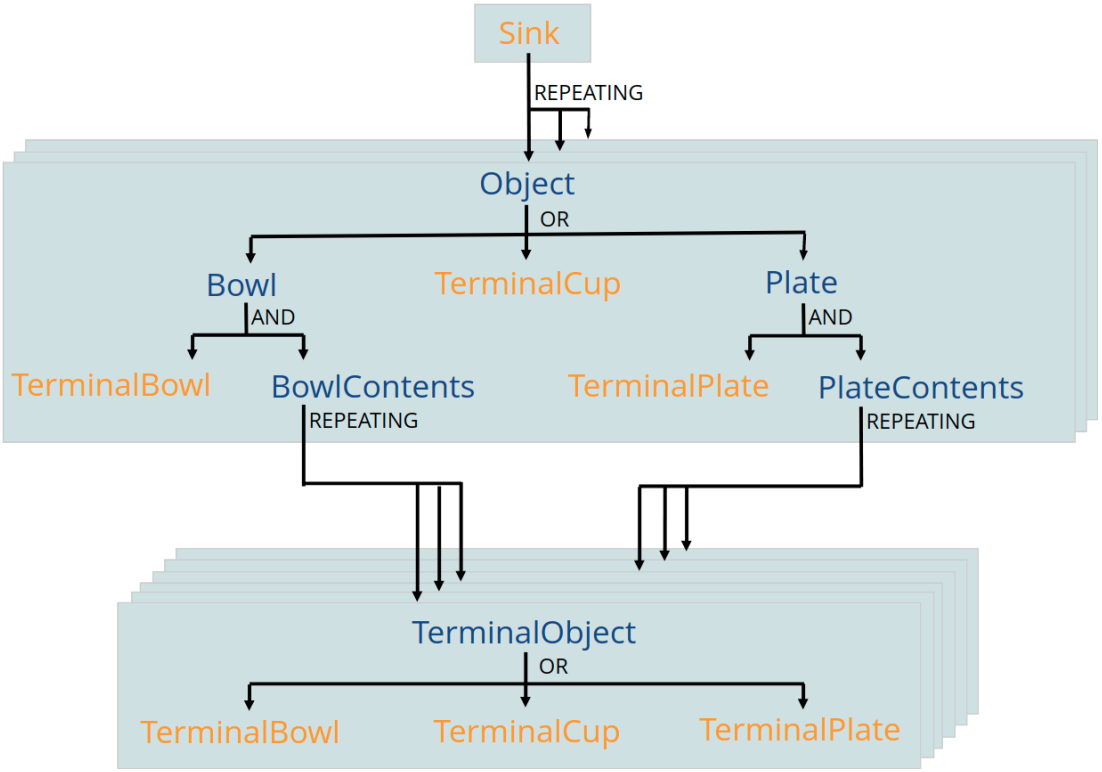


Figure 2-3: The structure of the grammar used to describe sinks. The SINK is a REPEATING-SET node that produces a random number of OBJECTS at random poses. Each object is an OR node that specializes into either a BOWL, PLATE, or TERMINALCUP. TERMINALCUPS directly produce geometry, while BOWLS and PLATES have the option of additionally producing more objects relative to themselves through BOWLCONTENTS or PLATECONTENTS intermediate nodes, each of which can produce a random number of TERMINALOBJECTS at random poses relative to themselves. TERMINALOBJECTS specialize into one of the three types of terminal scene geometry types. Equivalent sets of nodes (i.e. logical groupings of nodes with deterministically related poses) are boxed together.

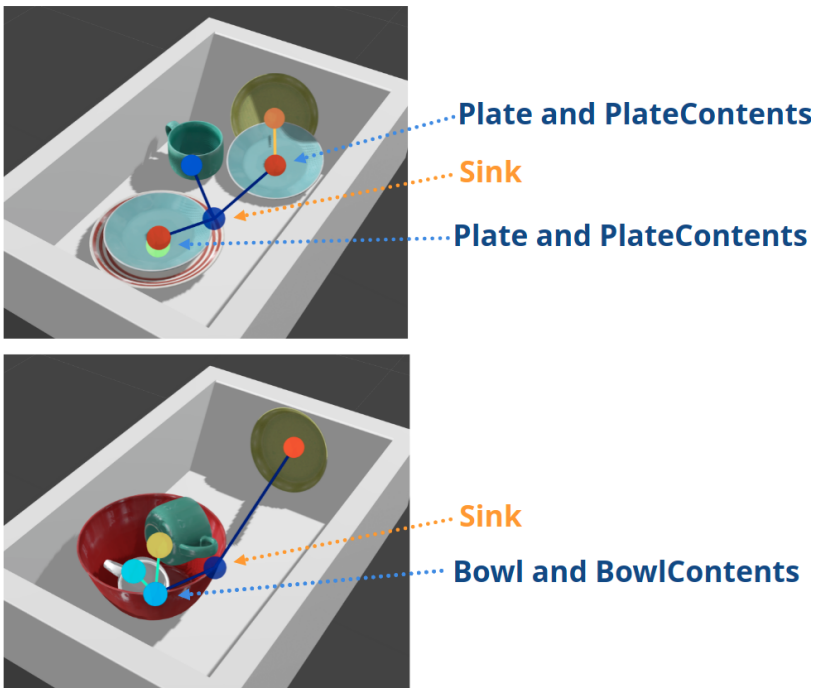


Figure 2-4: Two example scenes from our cluttered sink dataset, each annotated with scene trees indicating how they can be explained using the sink grammar.

or returns a scene configuration that bears little resemblance to the pre-projection scene, indicating significant bias in warping the object distribution. However, a grammar does not need to produce samples to be useful: we illustrate in Chapters 3 and 4 that this grammar still has useful applications in scene understanding and outlier detection.

Cluttered sink dataset

To aid in parsing and parameter estimation experiments, we constructed a dataset of sinks filled with a small set of object models using a custom virtual reality scene construction tool. We sourced 9 sink objects (3 cups, 3 plates, and 3 bowls) from the Ignition Robotics model database [63], and used a Razer Hydra [71] to teleoperate the placement of randomly generated sets of these objects into a simulated sink using Drake [59] to create a total of 68 scenes. We produced an additional dataset of 7 "outlier" scenes which were intentionally designed to violate the conventions of the inlier scenes; these are used in Chapter 4 to test whether the parameter estimation

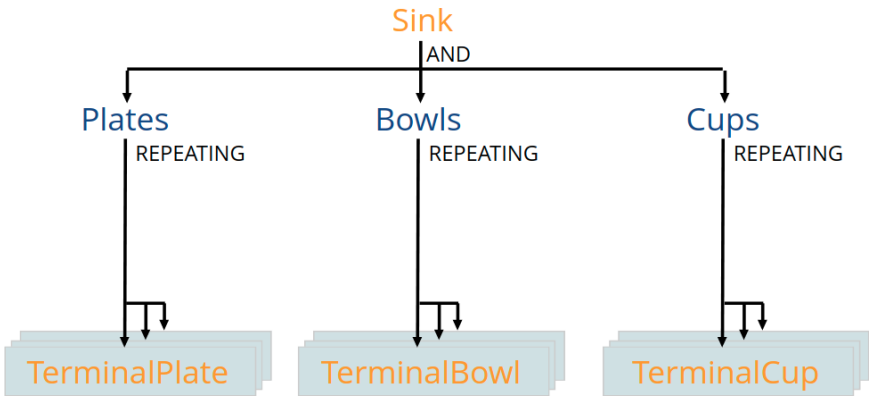


Figure 2-5: The structure of a simplified form of the sink grammar that is used as a baseline for comparison. This grammar models the distribution over appearance rate and pose independently for each object type without modeling any other relationships between objects.

process helps improve the outlier detection capability of our scene grammar tool.

Simplified grammar used as a baseline

For comparative use in Chapter 4, we define a simplified version of the Sink grammar which implements a naive approach for modeling this class of scenes. This baseline grammar (illustrated in Figure 2-5) implements the logic one might hand-code when implementing a simple scene generator for this class of scene: for each object type that could occur, we place a random number of objects of that type at independent random poses in the scene.

2.6.4 Dimsum table grammar

This grammar provides a structured description of dishware arrangements on a cluttered tabletop in a dimsum restaurant. Like the sink example, this grammar is motivated by the problem of describing structured messes typical in human environments. This grammar additionally demonstrates constraint specification and sampling under constraints.

The schematic structure of this table grammar is illustrated in Figure 2-6. The grammar is rooted at the TABLE, which uses some intermediate nodes to produce a random number of teapots and serving bowls, dishes, and trays randomly placed on

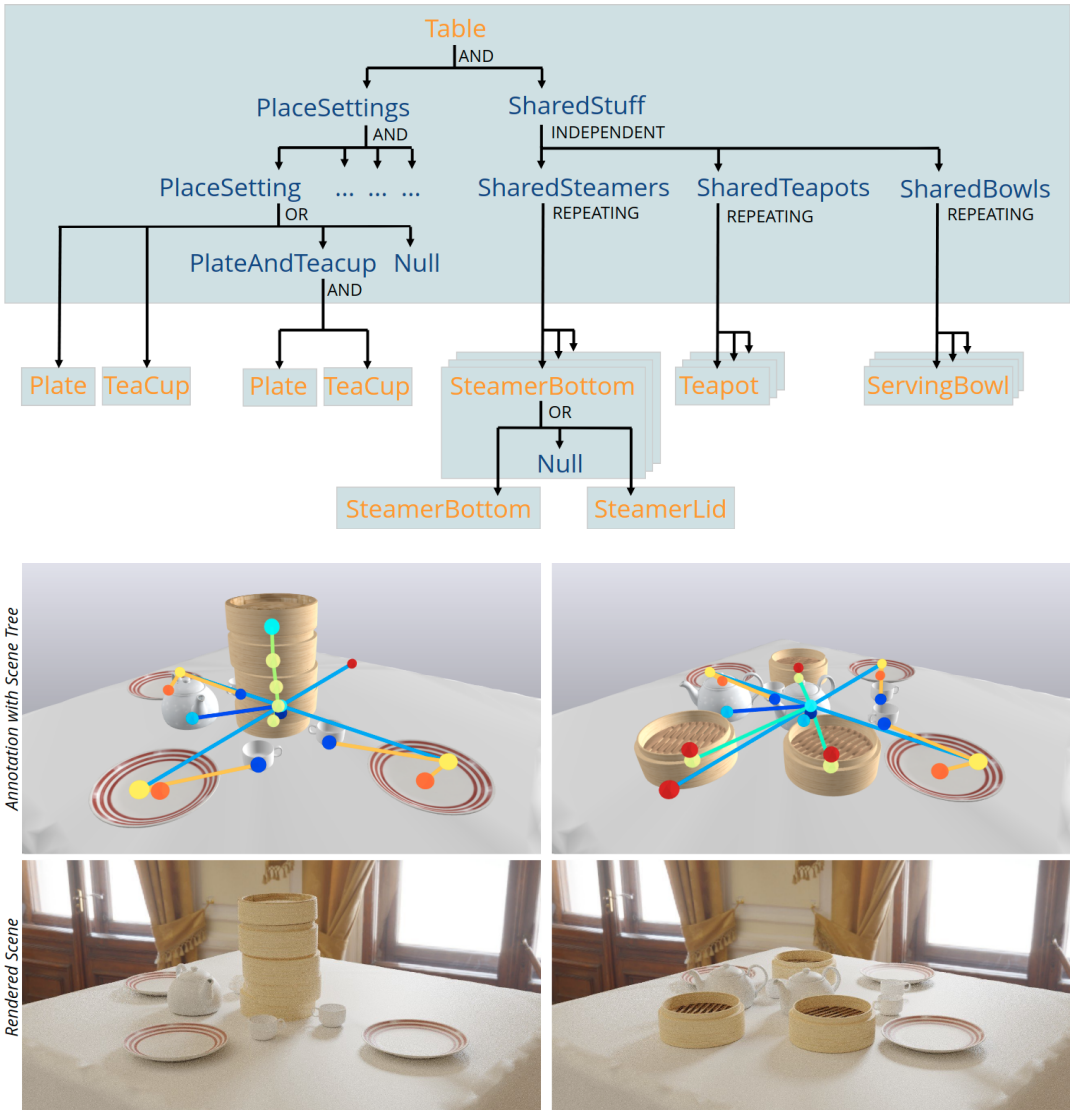


Figure 2-6: **Top**: The structure of the grammar used to describe tables at a dimsum restaurant. The TABLE produces a set of logical intermediate nodes that produce PLACESETTING nodes at the four seats of the table, as well as possible SHARED-STEAMER, SHAREDTEAPOT, and SHAREDBOWL nodes that produce a random number of their associated object type. These Shared* objects are all randomly placed in a common reference frame, while objects within a PLACESETTING are randomly placed in the frame of reference of that node. Equivalent sets of nodes (i.e. logical groupings of nodes with deterministically related poses) are boxed together. **Bottom**: Two examples drawn from this grammar annotated with their scene trees, and the same two examples rendered with our Blender-server tool (Section 2.5.3).

the table. The Table also produces four PlaceSettings, each of which is given a different deterministic pose at the four edges of the table. Each PlaceSetting can produce a Plate, a Teacup, both, or neither – with different rules capturing the different pose distributions of plates and cups occurring on their own, versus occurring together. Additionally, one of the common object types – a stackable steamer tray, SteamerBottom – is able to appear in stacks by utilizing a recursive self-reproduction rule. These stacks can terminate by the production of a SteamerLid or Null (a stack ending with nothing).

Details by node

The table grammar is defined over node types

$$\mathcal{N} = \{\text{Table, PlaceSettings, SharedStuff,}$$
$$\text{PlaceSetting, PlateAndTeacup, Plate, Teacup,}$$
$$\text{SharedSteamers, SharedTeapots, SharedBowls,}$$
$$\text{Teapot, ServingBowl, SteamerBottom, SteamerLid, Null}\}.$$

The Table, Plate, Teacup, Teapot, ServingBowl, SteamerBottom, and SteamerLid nodes have associated geometry; all other types represent an object or group of objects of unresolved final type.

- Table is an AND node producing intermediate PlateSettings and SharedStuff nodes at the same pose.

- PlateSettings is an AND node that produces four PlaceSetting nodes at four deterministic poses at the four seats at the table; each PlaceSetting is rotated with the positive x axis pointing inwards so that it provides a common reference frame for the objects within each PlaceSetting.

- PlaceSetting is an OR node that produces either a Plate, a TeaCup, a PlateAndTeacup, or Null (a dummy node that has no geometry). The

NULL and PLATEANDTEACUP nodes are produced at the same pose, while the PLATE and TEACUP nodes are produced with a parent-frame Normal and Bingham offset rule.

- PLATEANDTEACUP is an AND node that produces *both* PLATE and TEACUP nodes, each placed using a parent-frame Normal and Bingham offset rule.[10]

- SHAREDSTUFF is an INDEPENDENT-SET node that, for each of its children (see Figure 2-6, randomly chooses to produce that child independent of the others with a coin flip. It produces all children at the same pose of itself.

- SHAREDSTEAMERS, SHAREDTEAPOTS, etc. are REPEATING-SET nodes that produce a random number of their child type, placing each with a parent-frame Normal and Bingham offset rule.

- STEAMERBOTTOM is an OR node that chooses between producing NULL (nothing) at the same pose, or either a STEAMERLID or another STEAMERBOTTOM, each placed using a parent-frame Normal and Bingham offset rule. This represents a stackable steamer tray possible producing another steamer tray or lid stacked on top of itself.

- All other nodes are terminal nodes that are associated with appropriate, fixed a-priori geometry.

Constraints

In order to enforce that samples from this grammar are physically reasonable, we include two *pose* constraints on the items in the grammar:

1. **Items-on-table**: We enforce that each object's horizontal placement in the table frame is within the bounds of the table's edges, and that each object's origin is above the table surface.

[10]The parameters of the PLATE- and TEACUP-producing rules from the PLACESETTING vs from the PLATEANDTEACUP rule might capture different distributions over the placement of the Plate and Teacup if they are, or are not, co-occurring.

Figure 2-7: Four table environments sampled under the constraint that tables must satisfy the **Items-on-table**, **Items-non-penetration**, **Tall-stack**, and **Many-stacks** constraints from Section 2.6.4, using the conditional sampling techniques from Section 2.4. These constraints ensure that all objects are on the tabletop and are not colliding with each other; that the tallest stack of steamer trays is at least 4 trays high; and that there are at least 3 stacks of trays.

2. **Items-non-penetrating**: We implement non-penetration of all objects on the tabletop; but to keep evaluation of this constraint as fast as possible, we replace each object's geometry with cylindrical proxy geometry. That is, for each object, we define a "keep-out" radius, and enforce that the horizontal distance between each pair of objects is greater than the sum of each of their keep-out radii. Steamer trays within the same stack are exempt from this constraint.

We also use this grammar to demonstrate the application of an optional additional *structural* constraint:

1. **Tall-stack**: When active, this constraint enforces that the tallest stack of steamer trays is at least 4 steamers tall.

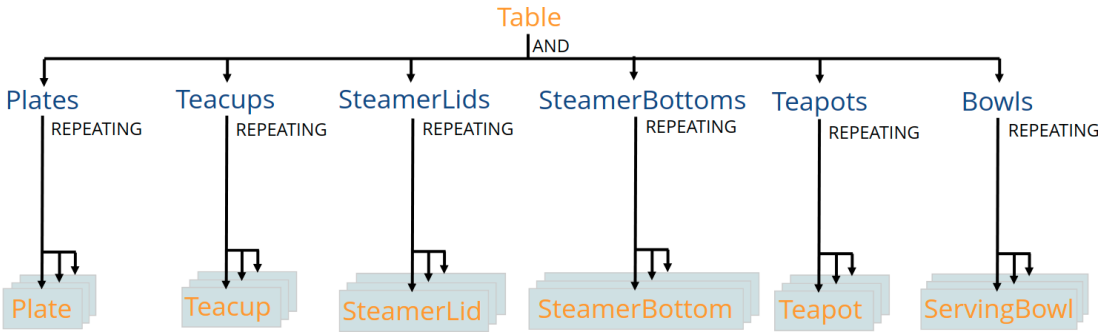


Figure 2-8: The structure of a simplified form of the table grammar that is used as a baseline for comparison. This grammar models the distribution over appearance rate and pose independently for each object type without modeling any other relationships between objects.

2. **Many-stacks**: When active, this constraint enforces that the number of distinct stacks of steamer trays is at least 3.

The scenes illustrated in Figure 2-7 were sampled under this set of structural and pose constraints using the constrained sampling methods described in Section 2.4. Scene trees were rejection-sampled from the table grammar until they satisfied the structural constraints, and then the poses of the nodes in that accepted tree were used to initialize an HMC chain to find configurations that were distributed according to the grammar rules while also satisfying the **Items-on-table** and **Items-non-penetrating** pose constraints.

Simplified grammar used as a baseline

Like the sink grammar, we define a simplified version of the Table grammar for comparative use in Chapter 4 which models the occurrence rate and pose distribution of each object type independently. This baseline grammar is illustrated in Figure 2-8.

We also define a similar constraint set to the full grammar:

1. **Items-on-table**: We enforce that each object's horizontal placement in the table frame is within the bounds of the table's edges, and that each object's origin is above the table surface.

2. **Items-non-penetrating**: We implement horizontal non-penetration of objects

on the tabletop using the same cylindrical approximation used in the full grammar version of this constraint. However, because we know stacks of steamers and lids can co-occur with similar horizontal positions, we exempt steamer-steamer, steamer-lid, and lid-lid pairs from this non-penetration constraint.

Decoration rules

While we've kept this table grammar relatively limited in order to simplify parsing and parameter estimation, it is easy to add additional rules and object types to generate even more interesting scenes with additional content and clutter. The tables illustrated in Figure 1-1 are generated by using the rules and constraints we have previously specified for the dimsum grammar, and then post-processing generated scenes by applying an additional set of *decoration* rules that:

- Sometimes place chopsticks on each plate.
- Fill each steamer tray and shared serving plate with random food models.

The complete scene after applying these rules is made physically reasonable by forward simulation.

2.7 Discussion

In this chapter, we have detailed a scene grammar formulation and shown that it can be used to describe a handful of interesting environments. We have also illustrated that we can combine a grammar with additional *structural and pose constraints* to further shape the induced distribution over worlds in ways that are not natural to express within the grammar itself. In the course of designing and implementing this grammar model, we have negotiated numerous design trade-offs and learned a handful of valuable lessons – the most important of which we remark on here.

2.7.1 Expressiveness-invertibility trade-off

In our grammar formulation, discrete and continuous randomness are carefully separated. At sampling time, a node first randomly samples which production rules will be activated to determine the set of child nodes that will be produced. Then, each rule independently samples a random pose for its corresponding child. Our set of node types makes it possible to express any discrete relationship between a parent and its active child set by appropriate composition of AND and OR rules. However, the legal relationships between a parent and child's *pose* are very restricted: in particular, the rules we enumerate in Table 2.1 only allow for unimodal Normal-like relationships.

This restriction is an intentional departure from some previous work: in [4], we allowed production rules to describe any continuous relationship between nodes to be employed by allowing rules to be written in a probabilistic programming language. While it is tempting to try to achieve as much expressiveness as possible, we have found that restricting the types of rules to only these "nice" analytic forms is worthwhile, as it makes the inverse problem of scene parsing much easier. (This is one of the primary subjects of Chapter 3.) Many complex pose relationships (e.g. objects with multimodal pose distributions relative to their parent) can be approximated by introducing more discrete structure to capture that relationship more accurately, in the same way that adding mixture components in a Gaussian mixture model can allow modeling of non-Gaussian distributions. For example, a plate on a table might be described as appearing with a roughly bimodal pose distribution – rightside-up, or upside-down – which we can capture in our grammar with two distinct production rules representing the production of a plate in either upside-down or rightside-up orientations.

2.7.2 Constituency vs dependency grammars

The singles-pairs grammar introduced above is an example of a scene that can be described by multiple grammars – and, as we will explore in Chapter 3, not all grammars are created equal! One way of categorizing grammars is to distinguish between

constituency and *dependency* grammars. These terms are borrowed from linguistics: *constituency grammars* describe the way that elements of sentence group together to form parts and phrases, with each word being a constituent of a phrase. In a scene grammar, objects may be described as constituents of some higher level grouping of objects, which itself is a node in the tree. The constituency and dependency variations of the singles-pairs grammar in Figure 2-2 highlights this difference: in the constituency variation, a pair of objects are distributed around a common "pair center" parent node, while in the dependency variation, one object is instead offset directly from the other.

A direct consequence of describing a scene with a constituency grammar is the introduction of *intermediate hidden nodes* whose poses might be difficult to recover. For example, in the singles-pairs constituency grammar, the "pose" of a pair of objects is somewhere between them, and to parse a scene, one has to decide exactly where that "pair" center is located. As discussed in Chapter 3, this transforms this problem from a purely discrete search over possible parse tree structures to a joint discrete *and* continuous search over tree structures *and* unknown poses. While we will show that we can still tackle that inference by mixed-integer optimization, it comes with a significant complexity and performance overhead. Indeed, Chapter 3 includes quantitative experiments demonstrating that if one has the choice between equivalent constituency and dependency grammars for a given class of scenes, one should almost certainly choose the *dependency* variation.[11]

2.7.3 Sampling under constraints

In this chapter, we present three techniques for resolving different kinds of constraints at sampling time: rejection sampling for resolving structural constraints; HMC for sampling trees that produces unbiased samples of trees from the original distribution that satisfy pose constraints; and a nonlinear projection procedure for handling pose

[11]From the perspective of general procedural models, dependency grammars are "constructive" models (according to the language of [72]) in which every decision in the model produces an observable artifact.

constraints that are practically too difficult to sample under the penalty method used by the HMC procedure. These strategies provide partial solutions to the constrained sampling problem, but each has its own limitations. As described in Section 2.4, the rejection sampling approach is unbiased but inefficient, and the NLP projection approach is relatively efficient but might arbitrarily warp the distribution of samples away from the distribution induced by the grammar.

HMC seems ideal in that, as penalties increase and under perfect mixing, it produces samples of node poses precisely from $P(T|C(T) \geq 0)$ *conditioned on the chosen tree structure*. However, $P(T)$ is really a distribution over both tree structures and node poses; to choose the tree structure itself, we use rejection sampling to satisfy any structural constraints, or otherwise pick a structure at random without regard to the pose constraints. However, the likelihood of a tree structure itself may be dependent on the pose constraints – and this dependence isn't captured in our procedure.

As an example, consider a grammar that produces $N \sim \text{Uniform}[0, 10]$ objects in a bin, combined with a pose constraint that all objects must fit inside the bin in a non-penetrating configuration. If the bin can only fit 5 objects, then the *constrained* distribution over scenes will have zero probability of having more than 5 objects, and might have relatively less likelihood of having 4 or 5 objects since there are fewer configurations of those objects that fit in the bin. Our existing procedure won't capture that final detail: it will instead produce scenes with a uniform random number of objects and try to make each scene independently satisfy the constraints. While we can easily reject those scenes in which pose constraint satisfaction is determined to be impossible (due to obviously poor final solutions), we're likely to *over-represent* scenes with many objects. A complete solution to resolve this bias could be assembled by synthesizing techniques from [45] and [21]. In particular, it should be possible to utilize a reversible-jump MCMC approach to construct a random walk over scene trees by combining random resampling of subtrees (to search over discrete tree structures) with HMC over node poses (to find likely node poses given a fixed tree structure).

Chapter 3

Scene Parsing

3.1 Introduction

Given a scene grammar and an observed scene, one might ask a fundamental question: "How would this grammar have generated this scene?" *Scene parsing* seeks the answer to this question. Given an observed set of objects o as a set of objects with fully observed poses and geometry, one can "parse" the scene under a grammar by finding a scene tree from that grammar that explains all of the objects. We frame scene parsing as a posterior inference problem: the scene grammar is a procedural model providing $p(o, T)$, and scene parsing is inverting that model by sampling sampling trees likely under the posterior $p(T|o)$.

This inverse problem can be very hard. The grammar may be capable of producing an incredible number of unique trees – related to the fact that there may be exponentially many ways of grouping or relating observed objects together – but only a few of those tree structures will be legal under the grammar. Even worse, many grammars involve additional unobservable node types representing abstract object groupings and cluster centers, each of which needs to be correctly proposed and associated with its constituent objects in the parse tree. Worse still, each of these latent intermediate nodes may have an associated pose that must be simultaneously estimated and optimized.

Perhaps as a consequence of this fundamental difficulty, greedy and heuristic pars-

ing strategies are common in the scene parsing literature. Unfortunately, these procedures typically require problem or grammar-specific heuristics, and can easily yield suboptimal or completely infeasible parses. In this chapter, we present an approach for principled all-at-once parsing by utilizing integer programming to perform optimization over the space of all parse trees while maintaining usable performance for practically-sized scenes and grammars. Beyond direct use as parsing techniques, we hope that the strategies used to formulate these parsing optimizations are inspiring to readers as strategies for performing principled, exhaustive exploration over different scene structures, with potential applications in areas like formal control synthesis and verification.

3.1.1 The MAP parsing problem

Given a grammar and an observed node set, we frame the maximum-a-posterior (MAP) parsing problem as an optimization that searches the space of all parse legal trees from the grammar to find the one that optimally explains the scene. We present and compare two variations on this approach.

- **Section 3.4: Pose-optimizing MICP Parsing** We show that we can write a mixed-integer *convex* program (MICP) that simultaneously searches over the structure and node poses of a scene tree from the grammar while optimizing a log-likelihood score. As a MICP, this problem can be solved to global optimality relatively efficiently with off-the-shelf solvers.

- **Section 3.5: Proposal-based IP Parsing** While the MICP approach is *complete* (in the sense that the approach can parse scenes from any grammar using rules from Chapter 2), it scales relatively poorly as grammar complexity increases. We present a second approach that takes a middle ground between MICP parsing and more classic heuristic approaches: for any intermediate nodes whose presence and pose is uncertain, we *heuristically* generate a population of candidate poses for those nodes, and then use a similarly-structured integer program (IP) to find the best way to glue those candidate nodes together into

an optimal parse tree. This gluing optimization is guaranteed to recover the best parse tree given the proposal set, so the suboptimality of the resulting parse tree decreases as more (or more accurate) intermediate node proposals are generated. In comparison to the MICP approach, this technique is applicable to a wider variety of scene grammars and has better scaling behavior, at the cost of requiring heuristic proposal generation.

Both of these approaches ultimately find an optimal parse tree by solving an MICP, which can be solved efficiently by a number of commercial solvers. This class of mathematical programs has several appealing properties. MICP solvers provide certificates of either global optimality or infeasibility (by way of a branch-and-bound optimization process), meaning that we can be certain that the parse tree we produce is either the best one, or that the set of observed objects we've encountered can't be explained by our grammar at all. Even further, the optimization process makes it possible to extract not just the globally optimal solution, but the *top N optimal solutions*. Depending on the grammar and parsing formulation, this sometimes means that we can try to collect multiple, *diverse* parse trees that provide alternative explanations for a scene. These diverse solutions are useful for evaluating parsing uncertainty or ambiguity, and prove useful in performing parameter estimation in Chapter 4.

In Section 3.7, we provide a quantitative comparison and analysis of the performance of these parsing techniques on a handful of grammars. In particular, we illustrate the significant difference the grammar formulation can make on the effectiveness of each parsing technique: given output-equivalent *dependency* and *constituency* grammars for a scene, we show that the dependency grammar is significantly easier to parse in terms of both runtime scaling and the accuracy and diversity of parsing results.

3.2 Related Work

Unfortunately, common dynamic programming and chart-based parsing techniques from formal grammars are hard to apply to scene grammars, as scene grammars typ-

ically have continuous inherited attributes *and* produce unordered outputs. Despite that difficulty, scene grammar parsing has attracted significant attention as a tool for scene understanding. Heuristic parsing methods that combine bottom-up and top-down structure proposals with greedy parse tree construction have historically been successful [4, 14, 73], but can be inconsistent for grammars where intermediate structure is hard to guess. Reversible-jump MCMC (RJMCMC) provides an appealing framework with which to search the space of varying-size parse trees [19,21,45,51,74], but this technique only works when one can engineer an effective jump proposal. To sidestep these difficulties, we focus on a novel all-at-once mixed-integer optimization parser which is *guaranteed* to find the optimal parse tree.

Scene parsing can also be viewed as a special class of inverse procedural modeling (IPM) as it is applied to more general procedural models. For example, one instantiation of IPM is vision-as-inverse-graphics, which is concerned with reproducing an observed image by figuring out the scene geometry [26,27] or even graphics code [75,76] that produced it. These techniques rely on combinations of vision subsystems (often conveniently trained on data generated by the model), sampling-based inference (e.g. MCMC), and specialized code and synthesis algorithms. Vision subsystems are typically seen as proposal engines that produce samples of "reasonable" explanations for an observed scene. Each of these pieces maps to an existing idea about scene parsing. Vision systems are an analog to intermediate node proposal systems that provide guesses of good intermediate structure for a parse tree. Sampling-based inference (i.e. performing MCMC over parse trees) is the direct analog of the RJMCMC techniques of [19, 45]. And the code synthesis approach in [76] utilizes a SAT-based search to find parse trees to describe a given object with constructive solid geometry, which is very similar in spirit to our integer programming approach.

3.3 Common Setup

Given a grammar with parameters θ and an observed set of objects o, finding the maximum a posterior (MAP) tree

$$\arg\max_{T} p_\theta(T|o)$$

is a general approach for "parsing" the scene, in the sense that such a MAP tree would be the best possible explanation for the scene using the grammar. Being able to find this optimal scene tree would be broadly useful, but this posterior inference problem can be extremely hard. The best scene tree T for a given scene may be ambiguous; may require proposing new hidden structure; and generally requires searching over diverse paths for explaining the scene.

Because the set of observed objects is unordered and attributed, common approaches to parsing from formal grammars are not helpful. We instead approach this parsing problem by enumerating the space of all parse trees for our grammar with a set of binary variables, and writing a cost and constraint set to extract the MAP parse tree as an integer program.

We can rewrite our objective

$$\arg\max_{T} P_\theta(T|o) = \arg\max_{T} \log(P(o|T)P_\theta(T))$$

by taking a log and applying Bayes' rule. In this optimization, we might search over all trees T, where $p(o|T)$ is an indicator function that the observed nodes in T precisely match observed objects o and those trees that match the observed nodes are ranked by their tree score $P_\theta(T)$.

3.3.1 MAP parse tree selection on the supertree

To facilitate the search over the space of all parse trees in the grammar, we construct a "supertree" ST from the grammar, which is constructed in such a way that any

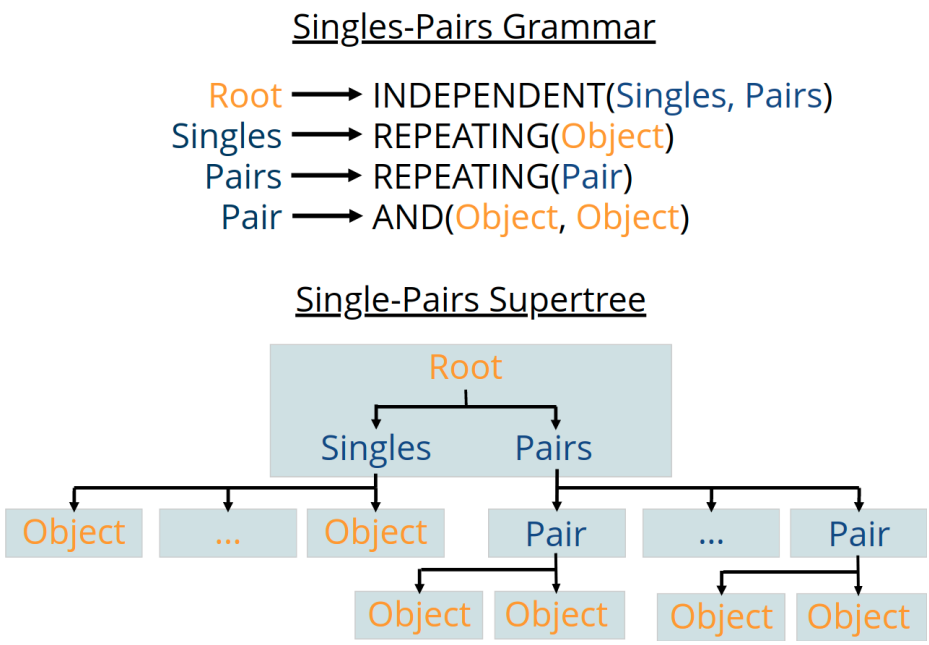


Figure 3-1: An abbreviated form of the singles-pairs grammar, and an illustration of its corresponding supertree. Any possible scene from the grammar occurs as a subtree of this supertree.

tree T is a subgraph of ST[1]. This supertree is constructed by initializing a tree at N_{root}, and for each node of type N, attaching one child for every rule R in $N.\mathcal{R}$ of type $R.N$ that would be produced by that rule. (See Figure 3-1 for an example.) If the grammar is recursive, then we bound the depth of the supertree to keep it finite.

This supertree ST proves to be a convenient data structure for parameterizing the space of all scene trees in the grammar, which we will utilize in both parsing strategies.

3.3.2 Supertree simplification via "equivalent sets"

A repercussion of the restriction on node types in Section 2.3.1 is that representing complex logical relationships often requires chaining the primitive AND-OR-SET types together, leading to a bloat of node types. (The sink grammar (Figure 2-3) is an extreme example, requiring many intermediate nodes to implement logic like

[1]This approach is similar in spirit to [16], which forms an enormous factor graph over every possible (discretized) placement of every possible node type. In our case, we instead capture node poses with continuous attributes, leading to a much smaller tree over all generations.

"a bowl produces geometry, but might also produce some number of objects inside of it".) However, these intermediate almost always have trivial pose relationships to their parents or children: we might distinguish between a Bowl, the TerminalBowl that implements its geometry, and its abstract BowlContents that decides whether the bowl contains anything, but they all have the same pose.

In general, we frequently encounter supertrees that have with sets of intermediate nodes whose poses are constant offsets from one another. To take advantage of this property, we define a partition of a supertree into "equivalent sets" of nodes, such that the pose of each node in an equivalent set are deterministically related. We can detect these equivalent sets with a simple graph algorithm: collect the nodes in the supertree, connect those parent and child pairs whose corresponding production rules are fixed offset rules, and find connected subgraphs of the resulting graph to find equivalent sets. By construction, the pose of the nodes any node in an equivalent set can be recovered from the pose of any other node in the equivalent set. Equivalent sets are illustrated in the grammar specification Figures (2-1, 2-2, and 2-3) as blue boxes drawn around sets of nodes.

Finding these equivalent sets corresponds to simplifying the grammar and supertree to only those spatial relationships that provide critical information for parsing. In both parsing strategies, we consider whether each *node* in the supertree is active, but only consider the contribution of poses at the level of equivalent sets.

3.4 Pose-optimizing MICP MAP Parsing

Our first optimization-based approach to solving the parsing problem is to parameterize the space of all possible parse trees under the grammar with a number of binary variables that choose the structure of the parse tree, and an additional set of continuous variables that control the poses of the nodes in the resulting parse tree. We construct an optimization over these mixed binary and continuous variables whose solution is the optimal parse for the scene. We use the supertree as our guide for allocating these decision variables: if we allocate a binary activation variable and

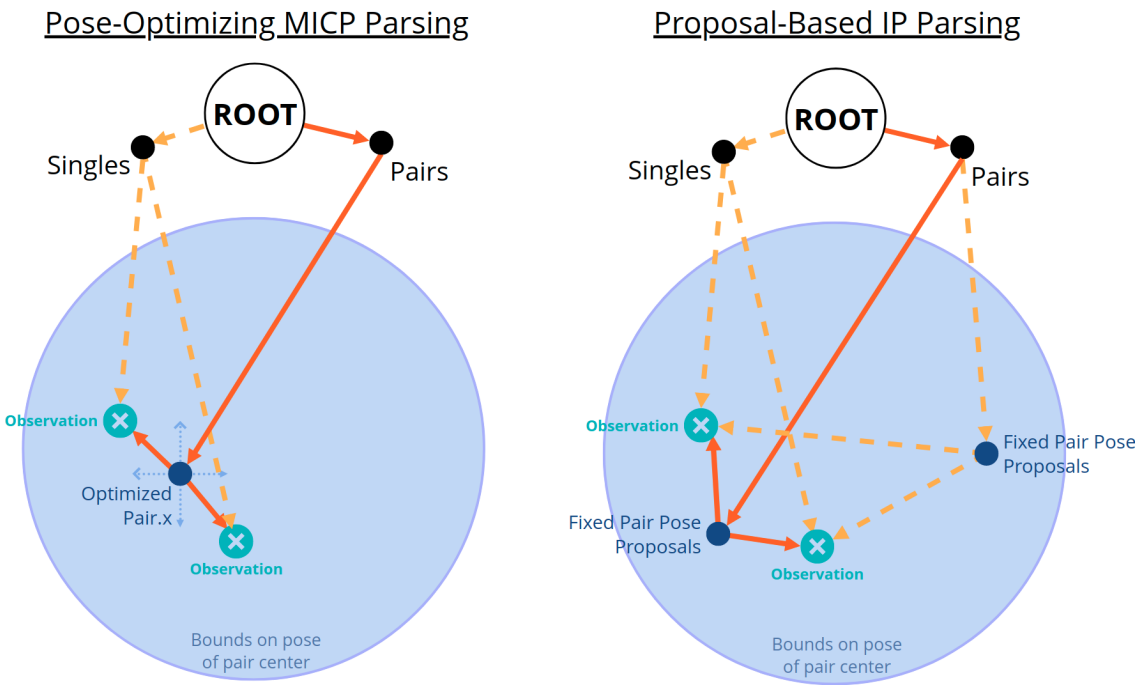


Figure 3-2: Pictorial depiction of the decision-making process formulated in each parsing method when parsing observations (represented in 2D as teal crosses) from the singles-pairs grammar. **MICP parsing** chooses a parse tree for the scene by selecting a subtree of the supertree that spans the set of observed nodes. (One such subtree has its edges highlighted in red in each figure.) Subtrees are weighted by their total tree log-probability, which is influenced by the number of types of active children of each active node, and the relative poses of active parent-child pairs. In MICP parsing, the pose of the intermediate Pair node is simultaneously optimized with continuous variables. **IP parsing** instead samples a set of proposed poses for each intermediate node type and adds an edge for every pair of possible parent-child poses.

continuous pose variable for each node in the supertree, we can formulate a MICP to find settings of those variables to optimize $P(T)$ while matching the observations (i.e. enforcing $P(o|T) = 1$).

Ideally, we would frame a *convex* MIP (i.e. MICP), which we would attempt to solve relatively efficiently to global optimality with a branch-and-bound approach. Unfortunately, the implementation of $P(T)$ might require enforcing some number of nonconvex costs and constraints; certain continuous relationships between nodes lead to nonconvex costs, and even simply constraining the optimized poses to lie $\in SE(3)$ is itself a nontrivial nonconvex constraint. As a result, a significant amount of this section is devoted to describing how to get rid of those nonconvexities by a combination of reparameterization and convex approximation.

3.4.1 Problem setup

Assume a grammar with supertree ST and an observed node set o. For each node n in ST, we add a binary variable $n.a$ indicating whether is active, and a continuous pose variable $n.x \in SE(3) = (n.t \in \mathbb{R}^3, n.r \in \mathbb{R}^{3\times3})$ representing the pose of that node if it is active. Any tree in the grammar (up to the recursion bounds used to generate the supertree) corresponds to a pattern of these node activation variables and a particular choice of node poses; and given these variables, we can compute $P(T)$ and $P(o|T)$.

3.4.2 Implementation of $P(o|T)$

To constrain that we only consider trees that match our observed set, for each node $\hat{n} \in o$, we add binary variables $b_{n,\hat{n}}$ for each $n \in ST$ of matching node type. A given $b_{n,\hat{n}}$ being active indicates that we're explaining the existence of observed node $\hat{n}$ with

node n in the our parse tree. We impose constraints

$$\begin{aligned} \forall \hat{n} \in o \qquad & \sum_{\text{same-type } n \in ST} b_{n,\hat{n}} = 1 \\ \forall n \in ST \qquad & \sum_{\text{same-type } \hat{n} \in o} b_{n,\hat{n}} \leq n.a \end{aligned}$$

which enforce that all observed nodes are explained exactly once, no latent node is used to explain more than one observation, and a latent node that can produce geometry is active if and only if it explains an observation.

When a correspondence $b_{n,\hat{n}}$ is active, we require that the latent node pose precisely matches the observed node pose. We implement this with a Big-M formulation:

$$\begin{aligned} \forall \{n, \hat{n}\} \text{ of matching type } : \\ |\hat{n}.t - n.t| &\leq M_t * (1. - b_{n,\hat{n}}) \\ |\hat{n}.r - n.r| &\leq M_R * (1. - b_{n,\hat{n}}) \end{aligned}$$

which constrains a node translation and rotation to be elementwise equal to its corresponding observed node, but free to vary if not corresponded. To choose the tightest possible M_R and M_t, choose $M_R = 2$, since elements of the rotation are bounded in $[-1, 1]$, and set M_t to be the largest expected translational distance (in *any* of the x-y-z axes separately) between nodes of this type in the grammar. Given these choices for M_R and M_t, these constraints will be functionally deactivated and hence not influence the optimal solution when their corresponding binary variables are inactive.

3.4.3 Implementation of $P_\theta(T)$

Implementing $P_\theta(T)$ requires constraining these variables to only encode legal trees, and encoding the objective term $P_\theta(T)$ of those trees.

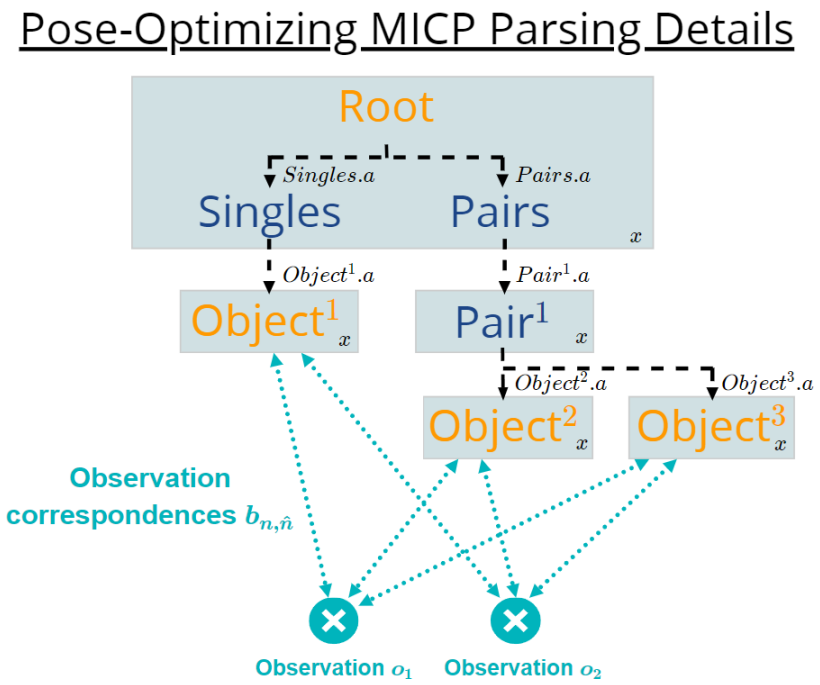


Figure 3-3: Depiction of the most important decision variables in the MICP parsing formulation, as they relate to parsing a two-observation scene from the singles-pairs grammar. Binary activation variables $n.a$ select whether each node in the supertree participates in the parse, and continuous pose variables x determine the pose of the nodes in each equivalent set (where each equivalent set is depicted with a shaded box). Possible correspondences of the observed objects to the various places they can be produced in the supertree are depicted in cyan, and are selected between with binary correspondence variables $b_{n,\hat{n}}$.

$x \in SE(3)$

We parameterize poses with translation vector and rotation matrix components $n.x = (n.t \in \mathbb{R}^3, n.r \in \mathbb{R}^{3\times3})$. We impose the constraint

$$n.r \in SO(3): \quad n.r^T n.r = \mathbb{1}, \quad det(n.r) = +1$$

with the mixed-integer convex outer approximation employed by [77]. These constraints allocate binary variables to partition the range of each element of $n.r$, and impose piecewise convex outer approximations (i.e. McCormick envelopes, [78]) on their bilinear products as they appear in the $SO(3)$ orthogonality constraints. As this is an expensive constraint, we are careful to only apply it where necessary: any node that produces geometry does not need this constraint applied, as that node's rotation will either be constrained to be equal to a matching-type observed node's rotation, or will not enter into the objective. (As discussed in Sections 3.3.2 and 3.4.3, in practice, we can actually avoid expressing this constraint in an even wider set of circumstances.)

Tree structure constraints

For each parent node n^p and its set of children n_i^c in ST, we enforce

$$n_i^c.a \geq n^p.a, \forall i$$
$$n_{root}.a = 1$$

which constrains that the children can only be active if the parent is active, and that the root of the supertree is always active.

Depending on the type of n^p, n^p being active may imply different constraints on the activation of the children, which we translate into constraints:

1. **AND Node**: $n_i^c.a = n^p.a, \forall i$: if the parent is active, all children are active.

2. **OR Node**: $\sum_i n_i^c.a = n^p.a$: if the parent is active, exactly one child is active.

3. **INDEPENDENT SET**: No constraints.

4. **REPEATING SET**:

   (a) $\sum_i n_i^c.a \geq n^p.a$: if the parent is active, at least one child is active.

   (b) $n_i^c.a \geq n_{i+1}^c.a$: assuming that this ordering of the children in the child set is consistent, enforce that the children must activate in this order.

   (c) $n_i^c.t[0] \geq n_{i+1}^c.t[0]$: Under the consistent ordering of children, the x-component of the children translations must increase. This constraint is not necessary, but breaks a symmetry resulting from children of this rule being exchangeable, decreasing the number of equivalent solutions the MIP solver must sort through.[2]

$\log P_\theta(T)$ **objective**

Starting from Equation 2.1, $\log P_\theta(T)$ expands to a sum of densities:

$$\log P_\theta(T) =$$
$$\sum_{\{n^p, \mathcal{R}_{active}\} \in T} \left[n^p.a \times \log P_\theta^N(\mathcal{R}_{active}|n^p) + \sum_{R, n^c \in \mathcal{R}_{active}} n^c.a \times \log P_\theta^R(n^c.x|n^p.x) \right]$$

where multiplication by $n.a$ indicates that we only include densities for those nodes and rules active in the tree under consideration. To implement this in an MI context, we use convex reformulations of these terms.

For the discrete density terms $n^p.a \times \log P_\theta^N(\mathcal{R}_{active}|n^p)$, the implementation depends on the node type:

1. **AND Node**: $\log(P_\theta^N)$ is always 0.

2. **OR Node**: $\log(P_\theta^N) = \sum_i^{|N.\mathcal{R}|} \log(\theta^N[i]) * n_i^c.a$, where θ^N is the set of parameters controlling the Categorical probability of each rule activation for this node type.

[2] In practice, we've found that these symmetry-breaking constraints significantly improve runtime; without them the branch-and-bound search process must sift through a set of equivalent solutions that grows exponentially in the number of unbroken symmetries. Breaking these symmetries is also vital to decreasing redundancy in the set of top-N solutions (see Section 3.6.2).

3. **INDEPENDENT SET**:

$$\log(P_\theta^N) = \sum_i^{|N.\mathcal{R}|} \log(\theta^N[i]) * n_i^c.a,$$

where θ^N is the set of parameters controlling Bernoulli probability of each rule activation for this node type.

4. **REPEATING SET**:

$$\log(P_\theta^N) = \sum_i^{|N.\mathcal{R}|} (\log(\theta^N[i]) - \log(\theta^N[i-1])) \times n_i^c.a,$$

where $\theta^N[i]$ is the parameter controlling the Categorical probability of exactly i children being active, with $\log(\theta^N[0]) = 0$. Since $n_i^c.a \implies n_{i-1}^c$, this objective term will equal $\theta^N[i]$ if i children are active, or 0 if none are active (which indicates that the parent and all of its children are inactive and should not enter the objective).

For the continuous density terms

$$n^c.a \times (\log P_\theta^R(n^c.x|n^p.x),$$

we need a way to deactivate the objective term $\log P_\theta^R$ when n^c is inactive. To avoid bilinear relationships between the activation variable and the node pose, we implement this deactivation with a slack formulation that adds variable $n^c.x_{slack}$ for each node:

$$|n^c.x_{slack} - n^c.x| \leq M(1. - n^c.a)$$

We can now express this cost term as

$$(1. - n^c.a) \times C + \log P_\theta^R(n^c.x_{slack}|n^p.x).$$

This term is designed such that when $n^c.a$ is deactivated, the objective takes 0 value. C is chosen to be the maximum value that can achieved by the rule probability

$\log P_\theta^R(n^c.x_{slack}|n^p.x)$ when $n^c.x_{slack}$ is unconstrained; for our rule types, this is the same for any setting of $n^p.x$ and is easily computable from the log-density equations in Table 2.1. M is chosen to be sufficiently large to not restrict the pose when the child node is inactive: it can be chosen to be the maximum expected x, y, or z coordinate any object of that type is reasonably expected to attain. If the child is active, then the constant term falls away, the slack pose is constrained equal to the node pose, and the objective is the appropriate rule log probability. If the child is inactive, then the child pose is unconstrained, and so will be chosen to maximize $\log P_\theta^R$, which cancels with the constant term to result in zero overall density. This functionally removes this rule from $P_\theta(T)$ when the child is inactive, even if other constraints have been placed on $n^c.x$.

The exact functional form of $\log P_\theta^R(n^c.x_{slack}|n^p.x)$ depends on the rule type, as listed in Table 2.1. For most rule types, the rule log-density can be directly implemented as a quadratic objective, but there are two special cases:

1. Bingham distribution log-densities are linear in the quaternion outer product qq^T; we represent the 10 unique terms in this outer product with intermediate variables. These outer product terms have linear relationships to elements of the rotation matrix according to the standard quaternion-to-rotation-matrix conversion formula, which we impose as linear constraints. See Section 3.4.3 for more details.

2. Parent-frame Normal- and Bingham-distributed offsets involve a bilinear relationship with the parent node rotation $n^p.R$. When required, these bilinear relationships are approximated with piecewise McCormick Envelopes [78] that reuse the binary variables involved in constraining the relevant rotations $\in SO(3)$.

Details of Bingham Distribution reparameterization

Given a 3×3 matrix of decision variables $r \in SO(3)$, we wish to penalize with cost corresponding to the Bingham log-density

$$ZM^T qq^T M.$$

Here, q is a quaternion corresponding to rotation r, and Z, M are Bingham distribution parameters matrices. Naively encoding this cost by adding decision variables for q leads to a nonconvex parameterization, since the conversion from r to q is not linear.

Instead, we introduce 10 decision variables for the 10 unique bilinear terms in the outer product qq^T:

$$qq^T = \begin{pmatrix} q_0^2 & q_0q_1 & q_0q_2 & q_0q_3 \\ q_0q_1 & q_1^2 & q_1q_2 & q_1q_3 \\ q_0q_2 & q_1q_2 & q_2^2 & q_2q_3 \\ q_0q_3 & q_1q_3 & q_2q_3 & q_3q_3 \end{pmatrix}$$

And enforce each of these constraints:

$$q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1$$

$$r = \begin{pmatrix} 1 - (q_2^2 + q_3^2) & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & 1 - 2(q_1^2 + q_3^2) & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & 1 - 2(q_1^2 + q_2^2) \end{pmatrix}$$

to ensure the (implied) quaternion is a unit quaternion that corresponds with the desired rotation. The matrix qq^T assembled out of these decision variables now enters linearly into the expression for the Bingham log-density.

Short-circuiting in observed cases

The piecewise convex approximations to the $SO(3)$ constraints and bilinear relationships in the parent-frame rule cases are extremely undesireable: as illustrated in the

results in this chapter, having even a handful of these constraints in the parsing MICP can increase problem size and runtime by an order of magnitude. We can avoid these approximations by taking advantage of the fact that many nodes in the supertree have associated geometry, and thus those nodes will either be corresponded to observed nodes or be inactive. This means that their pose can be described taking the pose of one of the observed nodes of matching type, rather than being directly optimized with a decision variable. In certain cases, this lets us rewrite non-convex cost terms in convex ways by taking advantage of our correspondence variables.

For each equivalent set of nodes in the supertree, we define that that equivalent set is *observable* if and only if any node in the equivalent set being active implies at least one observable node in the equivalent set is active. If an equivalent set is observable under this definition, then the pose of each node n in the equivalent set will take the value of one of a finite list of observed poses $n.\mathcal{X}^o$. This list of poses is generated by, for every node n^o in the equivalent set that has geometry, accumulating $\hat{n}^o.x$ from all matching-type observed nodes $\hat{n}^o \in o$.

As an immediate consequence, any node in an observable equivalent set does not need its rotation constrained to be in $SO(3)$, as we know that if the node is active, its rotation will be constrained to be exactly equal to an observed rotation. But further, in the cases of the the bilinear relationships in parent-frame translation and rotation rules, we can rewrite the costs to avoid those bilinear relationships. Examining the Parent-frame Normal offset case, there is a bilinear relationship between node pose variables in the parent-frame offset term $\delta = n^p.r^T(n^c.t - n^p.t - \mu)$. If the parent node is in an observable equivalent set, then we can replace δ with a new intermediate variable δ_{slack}, which is constrained such that

$$\begin{aligned}&\forall (r^o, t^o) \in n^p.\mathcal{X}^o : \\ &\quad |\delta_{slack} - (r^o)^T(n^c.t - t^o - \mu)| \leq M * (1. - b_{n,\hat{n}}).\end{aligned}$$

Here, we use the binary correspondence variable for observed node $\hat{n}$ to force δ_{slack} to be equal to a (linear expression of) that observed node's pose (r^o, t^o). This re-

moves the bilinear relationship (and its corresponding loose piecewise outer convex approximation), in exchange for a list of linear constraints with Big-M activations. A very similar trick can be applied to handle the bilinear term $(n^p.r)^T(n^c.r)$ in the Parent-frame Bingham distribution case.

The combination of constraint short-circuiting and equivalent set formation is critical to scaling this system. Without either, the kitchen-sink grammar in Figure 2-3 would require hundreds of intermediate node poses, each requiring $SO(3)$ constraints; this leads to MICPs with excessive solve times even using powerful commercial solvers. By applying these simplifications to the MICP, the same scenes can be parsed in seconds because we have removed many or all of the nonconvex constraints.

3.5 Proposal-based IP Parsing

The previous MICP parsing approach is solving two problems in parallel by simultaneously searching for the discrete parse tree structure and the continuous poses of the active nodes in the selected parse tree. Most of the complexity of the MICP approach comes from handling nonconvexities in the tree log-probability that includes nonconvex functions in node poses. However, we observe that if we assume we already have a library of reasonable candidate node poses for every node type, we wouldn't have to directly optimize over node poses, and could instead build an integer program (IP) that searches over all possible ways of connecting those nodes together into a reasonable parse tree.[3]

This parsing approach relies on two components:

1. A proposal engine that produces reasonable guesses for intermediate node poses. We provide top-down and bottom-down heuristics for this purpose in Section 3.5.1.

2. An optimization formulation that searches for an optimal way to connect together the set of observed nodes with intermediate nodes at those proposed

[3]This observation motivates the short-circuiting trick used to simplify the MICP in Section 3.4.3; this approach applies that trick to non-observed cases as well.

poses to create a parse tree that's valid under the grammar.

3.5.1 Proposal generation

Given a grammar with a set nodes types $\mathcal{N}$ and an observed set of nodes o, we seek to generate candidate intermediate (unobserved) node poses that are likely to appear in scene trees that produce o. Specifically, we generate a mapping $\mathcal{X}^{prop} = \{N \to \{x_i^N\}\}$ from *unobserved* node type N to a list of proposed poses for that node type x_i^N.

Here, we describe two independent mechanisms for generating these proposals. The techniques we describe here are heuristic methods designed to work well for the grammars we use as examples in this thesis; they demonstrate common concepts we think are useful for heuristic proposal generation, but may not be well-suited to all grammars.

Each method produces a *list* of proposed intermediate nodes: to generate a mapping from node types to candidate poses, we take the set of unique proposed poses for each node type in the candidate sets generated.

Top-down sampling

A straightforward method for generating candidate intermediate nodes is to forward-sample the grammar. Given a set of observed nodes, we know that each of those observed nodes as well as the grammar root will be present in any scene tree. As described in Algorithm 2, we can produce intermediate nodes by sampling children from each rule under each of those guaranteed-to-be-present nodes. We continue to produce candidate intermediate nodes in a recursive manner by forward-sampling from the sampled children until we sample an observed node type (or nothing); these can be thrown out, as there's no value in proposing new observed nodes that we didn't observed in the current environment.

Algorithm 2: Algorithm for forward sampling candidate intermediate nodes given a grammar and observed node set.

```
Input  : the root node root
Input  : an observed node set observeds
Output: a dict with node type keys and pose set values
/* Forward sample from each node we know to be present in the
   scene and collect resulting unobserved nodes.                */
output = {}
for origin_node in (observeds + [root]) do
    /* Forward-sample children from this node, and from any
       produced children, etc.                                  */
    expand_queue = [origin_node]
    while len(expand_queue) > 0 do
        parent = node_queue.pop()
        /* For *every* rule, sample a child.                    */
        for rule in parent.rules do
            /* Sample the pose of each child being produced.    */
            child = rule.sample_child()
            /* If this child is an unobserved node type, then
               collect its pose as a candidate, and try to keep
               producing top-down proposals under it.           */
            if !child.has_geometry then
                node_queue.push(child)
                output[type(child)].push(child.x)
return output
```

Bottom-up rule inversion

Depending on the grammar, it may be hard or impossible to randomly sample an intermediate node that reasonably explains an observation. For example, in a grammar describing clustering behavior of objects, an intermediate unobserved node might be used to represent the center of a cluster – but in high dimensional spaces, randomly sampled cluster centers are unlikely to remotely align with the real cluster centers of the data. Instead, we prefer to propose these cluster centers *bottom-up* to ensure they align with observed objects.

We frame this bottom-up proposal generation as, given a child node n^c and candidate parent node n^p related by a rule with density $P^R(n^c.x|n^p.x)$, finding the maximum likelihood $n^p.x$. While we could conceivably draw samples from the posterior

$P^R(n^p.x, n^c.x)$ to get even greater coverage, we believe that because this sampling would necessarily draw from a high dimensional space, it would be unlikely to sample configurations that provide any more information that these MAP estimates.

Fortunately, for the rules used in this thesis (listed in Table 2.1), this maximum-likelihood problem is easy to perform in closed-form:

- Fixed offset rules have a single valid parent translation or rotation at which the parent and child pose are related by that fixed offset.
- Normal- or Bingham-distributed offset rules have a maximum-likelihood solution at which the child is placed according to the mode of the Normal (or Bingham) distribution. We choose the parent rotation first, and then use it to choose the optimal parent translation.
- For Uniform rotation rules, we pick an arbitrary parent rotation (i.e. $R = \mathcal{I}^{3\times3}$) as the maximum-likelihood solution, since all parent rotations are equally likely.

In the case where additional non-trivially-invertible rules are included (e.g. the arbitrary probabilistic-programming-based rules used in [4]), performing this inversion is not so easy. Learned inverse functions would be a good fit for solving the inverse proposal problem in that general case, as arbitrarily many samples of the forward rule can be acquired at very low cost.

As in top-down proposal sampling, we apply this procedure in a recursive bottom-up manner: each observed node produces candidate parents, which in turn produce additional candidate parents until an observed or root node type is proposed. To prevent cycles, we limit the recursion depth of this procedure.

3.5.2 Formulation

Assume a grammar with supertree ST and node types $\mathcal{N}$; an observed node set o; and a set of mappings from unobserved node types N to possible node poses $\mathcal{X} = \{N \to \{x_i^N\}\}$, generated by the heuristic intermediate node proposal algorithms in Section 3.5.1.

As in the MICP approach, we use the supertree and its component equivalent sets of nodes as a mechanism for selecting the active parse tree. For each node in the supertree, we allocate a binary activation variables $n.a$ that decides whether that node will participate in the parse tree. We then build a finite set of possible node poses for each equivalent set of nodes in the supertree, and choose among them with binary variables. For each equivalent set S from ST, we build a set of poses $S.\mathcal{X}$ representing the complete list of sampled poses that any node in that equivalent set could take. If the equivalent set contains observed node types, then this list of poses is collected from all observed nodes of like type. Otherwise, this list of poses is collected from the intermediate node pose proposals $\mathcal{X}$ of all constituent nodes in the equivalent set. Each equivalent set is allocated a set of binary variables $S.B = \{0,1\}^{|S.\mathcal{X}|}$ that decides which of its possible poses this node will use.

Figure 3-4 illustrates how we can use relationships between these binary variables to choose the structure of a parse tree. For every parent and child node pair $\langle n^p, n^c \rangle$ in ST whose connection crosses a boundary between equivalent sets (i.e. $n^p \in S^p$, $n^c \in S^c$), we'll consider every possible *pair* of poses $\langle n^p.x, n^c.x \rangle \in S^p.\mathcal{X} \times S^c.\mathcal{X}$ as a weighted edge, where the weight is a constant calculated from $P^R_\theta(n^c.x|n^p.x)$.[4] By inspecting node activation variables $n.a$ and equivalent set pose selection variables $S.C$ across the tree, we can determine which edges are active and hence the tree log-probability. By applying appropriate constraints on these variables, we can ensure only legal scene trees are selected.

3.5.3 Implementing $P_\theta(T)$

We need a number of constraints to enforce that we only allow legal parse trees, and a number of costs to implement $P_\theta(T)$. To create legal parse trees, we constrain:

- For every pair of nodes n^p, n^c in ST, $n^p.a \geq n^c.a$; i.e., a child is only active if the parent is active.

[4] We don't need to worry about pose relationships within equivalent sets since all nodes in the equivalent set have deterministically related pose.

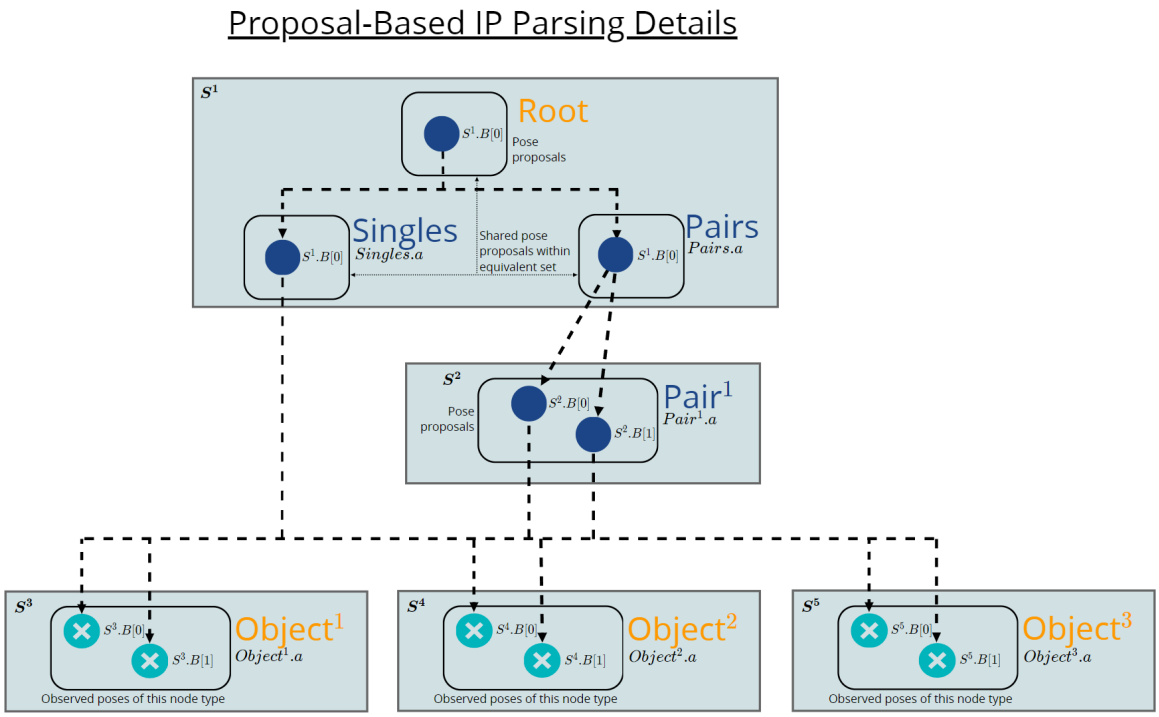


Figure 3-4: Depiction of the most important decision variables in the IP parsing formulation, as they relate to parsing a two-observation scene from the singles-pairs grammar. Binary activation variables $n.a$ select whether each node in the supertree participates in the parse. Within each equivalent set (indicated with shaded boxes, i.e. S^1, S^2, etc), a candidate node poses is selected from a set of proposals (shown in a rounded box) using binary selection variables $S^i.B$. For each parent/child pair in the supertree, each combination of choices of parent and child pose from respective their proposal sets induces a different cost based on the relative pose relationship of those nodes. Appropriate constraints and costs on these binary variables ensures that only legal parse trees that explain the observed nodes are selected by the parsing procedure.

- For every equivalent set S and its constituent nodes n_j, constrain

$$\sum_i S.B[i] \leq \sum_j n_j$$
$$\sum_i S.B[i] \leq 1$$
$$\sum_i n_j.a \leq \sum_i S.B[i] \, \forall j$$

  which enforces that the equivalent set has exactly one selected pose if and only if any of its constituent nodes are active (and chooses no pose and has no active constituent nodes otherwise).

We additionally constraint $n^{root}.a = 1$ to force the root to always be present, and enforce that the root's equivalent set selects the pose x^{root} as is required by the grammar.

For the part of $P(T)$ corresponding to the log probabilities of the active rule set under each node, we use the same cost terms from Section 3.4.3. These terms calculate the log-density of each discrete production rule $P_\theta^{N^p}(\mathcal{R}_{active}|n^p)$ for each parent node n^p with active child rules $\mathcal{R}_{active}$, which is a linear function of the activation variables of n^p and the activation variables of n^p's direct children in ST.

For the part of $P_\theta(T)$ corresponding to the continuous pose relationship log-probabilities, we build cost terms by calculating edge selection variables by pre-calculated edge scores. First, observe that only parent-child node pairs that are in *different* equivalent sets will have pose relationships that contribute to $\log P_\theta(T)$; parent-child pairs within an equivalent set have deterministically related poses that occur with log-probability 0 (i.e. probability 1). For each parent-child node pair $\langle n^p, n^c \rangle$ in ST whose connection crosses a boundary between equivalent sets (i.e. $n^p \in S^p$, $n^c \in S^c$), create indices i and j that index the proposed poses in $S^p.\mathcal{X}$ and $S^c.\mathcal{X}$. We consider every possible *pair* of poses $\langle n^p.x, n^c.x \rangle \in S^p.\mathcal{X} \times S^c.\mathcal{X}$ as set of weighted edges $E^{n^p \to n^c}$; these weights are calculated from $P_\theta^R(n^c.x|n^p.x)$ for each constant value of the parent and child pose. We allocate binary variables $b^{n^p \to n^c} \in \{0,1\}^{|S^p.\mathcal{X}| \times |S^c.\mathcal{X}}$ such that $b^{n^p \to n^c}[i,j]$ activates the edge between selected

pose i and selected pose j. We apply these constraints:

- $\sum_k b_k^{n^p \to n^c} \leq 1$, which enforces that at most one edge is active.
- $\sum^k b_k^{n^p \to n^c} = n^c.a$, which enforces that the child has to have an active incoming edge in order to be active.
- $b^{n^p \to n^c}[i,j] \leq \frac{S^p.B[i] + S^p.B[j]}{2}$, which only allows edge activation when the appropriate parent and child poses have been selected.

Given all of this setup, we can now add an objective terms

$$P_\theta^R\left(S^p.\mathcal{X}[i] \middle| S^c.\mathcal{X}[j]\right) * b[i,j]$$

representing the cost contribution of the pose relationship represented by edge when that edge is present.

3.5.4 Enforcing $P(o|T) = 1$

For each observed node $\hat{n} \in o$ of type N, we scan over all equivalent sets containing a node of type N and collect those binary variables representing the selection of $\hat{n}$'s pose as the pose for that equivalent set; we refer to this set of correspondence variables as $\hat{n}.b_{corr}$. We enforce $\sum \hat{n}.b_{corr} = 1$, which, in combination with previous constraints, enforces that each observed node is explained by the parse tree exactly once.

3.6 Additional Features Common to Both Methods

3.6.1 Nonlinear optimization post-processing

In some cases, the formulation of either of these parsing techniques requires some level of approximation. In the MICP parsing case, the constraints to keep the rotation matrix decision variables $\in SO(3)$ and the McCormick Envelope approximations used to encode parent-frame translation and rotation rule probabilities are approximate. As a result, the solution to the MICP may not be precisely feasible or optimal under

the true rotation constraints[5] and objective. In the proposal-based parsing case, the proposed poses of intermediate nodes may themselves not be optimal, so the optimal solution to the optimization may not be an optimal parse.

For that reason, in both cases, we take the MICP solution and use it to seed a nonlinear program (NLP) of the same objective. The NLP fixes the parse tree structure and optimizes the poses of any intermediate nodes with respect to the full nonlinear form of $P(T)$. This NLP is handed to a commercial solver and seeded with the MICP optimal solution; solutions are typically found in tens of milliseconds.

3.6.2 Multiple solutions

Because both of these parses are framed as a mixed-integer convex optimization, both forms can be solved to certified global optimality by a number of off-the-shelf solvers by an efficient (though still exponential in the worst-case) branch-and-bound algorithm [79]. As a bonus, the same procedure can be used to extract the global top N best integer solutions (paired with their associated optimal continuous variable setting) as ranked by the objective. When the only binary variables in the program correspond to distinct parse tree structures, these multiple solutions each describe a different parse tree structure for the scene. Practically speaking, this means that we can get diverse explanations of the scene from the MIP solver at the cost of more runtime. We explore some of these top N solutions (including when they are and are not informative solutions) in Section 3.7.1.

3.7 Experiments

We analyze the performance of these parsing methods on grammars describing two classes of scenes:

- The **singles-pairs grammar** describes sets of oriented objects that can appear

[5] [77] provides a general analysis of the tightness of the $SO(3)$ constraints. In our case, using 2 binary variables per half-axis and linear interval binning, we see mean errors from orthogonality $mean(R^T R - I)$ on the order of $0.05 - 0.1$ for unobserved nodes during example parses.

on their own or in covarying pairs, and is described in detail in Chapter 2.6.2. We distinguish between two variations of this grammar: a *constituency* variation that specifies that a pair of objects are both positioned relative to a latent "Center" location, and a *dependency* grammar that specifies that one object in the pair is instead offset relative to the other object.

- The significantly more complex **sink grammar** describes the contents of a cluttered sink in which plates, bowls, and cups appear; objects can appear either independently, on top of plates and inside of bowls. This grammar is described in detail in Chapter 2.6.3.

3.7.1 Case studies on singles-pairs grammar variations

In order to demonstrate that our parsing procedure recovers qualitatively reasonable parse trees, we draw random samples of scenes from the singles-pairs grammar and attempt to recover the ground truth scene tree with both parsing methods. We illustrate the ground truth scene tree along with the raw parse output and NLP-postprocessed parse tree from both methods and grammar variations in Figure 3-5.

The MAP parse tree is ultimately recovered by both techniques under both grammar variations: in both cases, both techniques recover a ground truth parse tree structure that groups two nearby objects into one pair. Note, however, that in parsing the constituency grammar in particular, both techniques recover suboptimal parse trees before NLP postprocessing. The MICP parse tree has the correct structure, but a suboptimal pair location. This suboptimality is due to looseness in the piecewise outer approximation of the bilinear relationship required to express the cost relationship between the pair location and its constituent object locations. This suboptimality could be tightened by decreasing the size of each McCormick envelope, which could be achieved by partitioning the space of translations more finely (i.e. with additional binary variables, which will come with an increase in runtime). Also note that this particular grammar is a *worst case* for the MICP parsing technique: it involves parent-frame Normal translation offset and parent-frame Bingham rotation

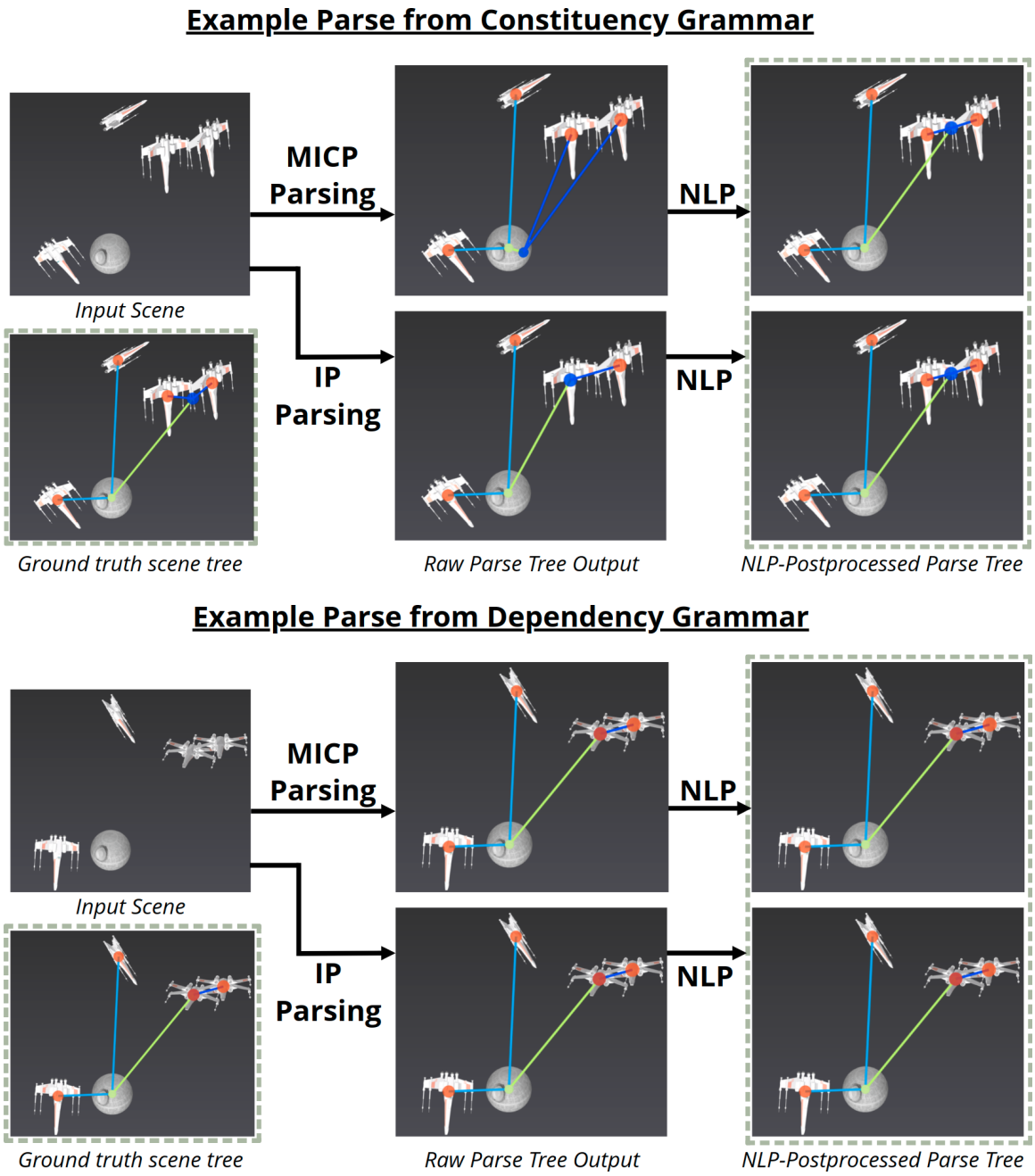


Figure 3-5: Example scenes drawn from the singles-pairs constituency and dependency grammar variations, as parsed by both the MICP (top row in each subfigure) and IP (bottom row in each subfigure) techniques. Parses before and after the NLP-based post-processing step are illustrated. The ground truth scene tree for this scene is illustrated next to the input scene for comparison.

offset rules, both of which involve bilinear relationships involving the parent and child poses in their log-likelihood evaluation. Similar grammars written using world-frame Normal translational offset rules (which can describe similar translational offsets, but without the ability for those offsets to vary with the parent rotation) do not face this issue. As illustrated in Figure 3-6, even in this difficult case, the MICP technique recovers the right parse tree structure in 95% of cases, allowing the NLP post-processing to easily recover the optimal solution and resolve the suboptimality.

The IP parsing also recovers a suboptimal pair location: it uses the pose of one of the two objects in the pair as its proposal for the pose of the latent pair (a result of a bottom-up pose proposal rule as described in Section 3.5.1). Since the recovered tree structure is correct, NLP post-processing is able to quickly and easily improve on these suboptimal solutions to find MAP-optimal parse trees.

Impact on parsing accuracy and runtime scaling

In order to probe both the accuracy and typical runtime of each parsing method, we can sample large numbers of scenes (along with their ground truth scene trees) from each of the singles-pairs grammar variations and parse each one with each method, and examine relevant statistics across that population of parsing trials. We measure the parsing accuracy by comparing the total log-probability of the recovered parse tree against the total log-probability of the ground truth tree; we consider a success to be when the parse tree meets or exceeds the ground truth tree probability. We can additionally vary the grammar *complexity*, which we measure in terms of the maximum number of objects the grammar can produce, by changing the maximum number of pairs and single objects the Singles and Pairs nodes are allowed to sample. Figure 3-6 illustrates both a measurement of parsing accuracy of each method on each grammar variation, and the distributions over runtime we observe from each parsing method on randomly sampled scenes as we vary the grammar complexity.

We see that both parsing methods succeed 100% of the time on the dependency grammar, but occasionally fail on the constituency grammar.[6] When parsing the

[6]Note the stringent definition of failure here; a perfectly reasonable parse tree that's slightly

Accuracy (relative to ground truth) of parsing methods

		Method MICP	IP
Grammar Variation	**Constituency**	15%/**95%**	56%/**100%**
	Dependency	100%/**100%**	100%/**100%**

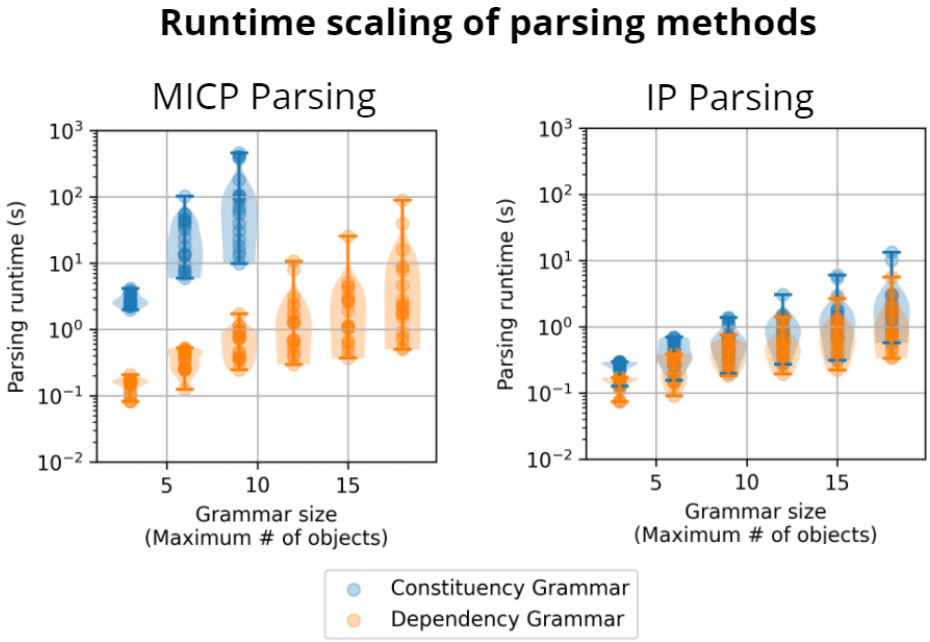


Figure 3-6: **(Top)** Assessment of parsing accuracy of both methods on the singles-pairs grammar. Across N random samples from the corresponding grammar ($N = 60$ for the constituency grammar under the MICP method; $N = 120$ otherwise), we indicate the percent of parses that match or beat the log-likelihood score of the ground truth tree for the sampled scene before (upper number) and after (lower, bolded number) performing NLP-based post-processing. **(Bottom)** Comparison of runtime of both parsing methods on randomly sampled ($N = 20$ per column) scenes from the singles-pairs grammar, as the complexity of the grammar is increased by increasing the maximum number of objects that the grammar can produce. Note the log scale on the y axis.

constituency singles-pairs grammar, the IP parser finds suboptimal trees 56% of the time, but after the NLP post-processing step, all of those results are improved to match or beat the ground truth parse trees. This indicates that the IP parser is selecting reasonable parse tree *structures*, but not finding the optimal latent poses for any Pair nodes. This makes sense, as the IP parser must choose the latent Pair poses from a set of proposals, none of which may be optimal. The MICP parser shows similar behavior: it rarely returns a perfect parse tree directly from the MICP optimization, but after NLP post-processing its performance rising to near perfect. As noted previously, this is a worst-case grammar for the MICP parser, as parsing it requires approximation of bilinear terms between the latent Pair translation variables and its own rotation. The MICP-optimized Pair translation is thus an approximation of the optimal Pair translation, which these results indicate is sufficiently good to lead the MICP to recover the optimal parse tree structure in 95% of cases, and leads the MICP astray in the other 5%.

Unfortunately, all methods show roughly exponential worst-case empirical time complexity as grammar size, which corresponds to underlying optimization size, is increased. The worst of this scaling behavior is seen during MICP parsing of the constituency grammar, which requires solving by far the hardest parsing problem (i.e. inferring unknown latent node poses); parsing such scenes can take tens to hundreds of seconds each, for scenes with handfuls of objects. IP parsing of the same scenes proves much faster and easier, and while still increasing in time roughly exponentially with grammar complexity, the rate of increase is modest and allows rapid parsing of much larger scenes.

For reference, for the MICP parser, the number of integer and continuous variables scales approximately as

$$O\Big((\#\ \textit{of supertree nodes})\Big).$$

worse than ground truth is still counted as a failure for the purpose of this experiment.

For the IP parser, the number of integer variables scales approximately as

$$O\Big((\#\ \textit{of supertree nodes}) \times (\#\ \textit{of pose proposals per node})^2\Big).$$

For this example, the number of supertree nodes grows linearly with the number of nodes that the grammar produces. (This relationship is not true in general for all grammars.) In the IP parsing case, the number of pose proposals per node also grows linearly in the number of observed nodes, as each observed node is used as a pose proposal.

Impact on finding multiple solutions

While the MIP solver can be asked to find the top N solutions to both forms of the parsing problem, the quality of those top N solutions can vary, as illustrated in Figure 3-7. The solver considers solutions to be different if their *integer solutions* are different, but depending on the parsing formulation and grammar, different integer solutions might correspond to the same functional parse tree. In the case of the dependency grammar, the only integer variables in the optimization correspond to choices that determine parse tree structure, so the top N solutions all correspond to *structurally* different parse trees both before and after the NLP pass. As a caveat, these parse trees are guaranteed to be different *modulo any unbroken symmetries in the grammar*: e.g., an AND node that activates two identical rules can lead to two "different" parses that activates either one of the rules to produce a functionally identical, but technically different, parse tree. These symmetries can be broken by enhancement to the parsing formulations.

Unfortunately, both formulations of the parsing problem for the constituency grammar muddy this story. Because of the binary variables involved in the $SO(3)$ constraints and piecewise McCormick approximations of bilinear cost terms, different integer solutions to the MICP parsing formulation are no longer guaranteed to correspond to unique parse trees: for the scene illustrated in Figure 3-7, the top 10 unique

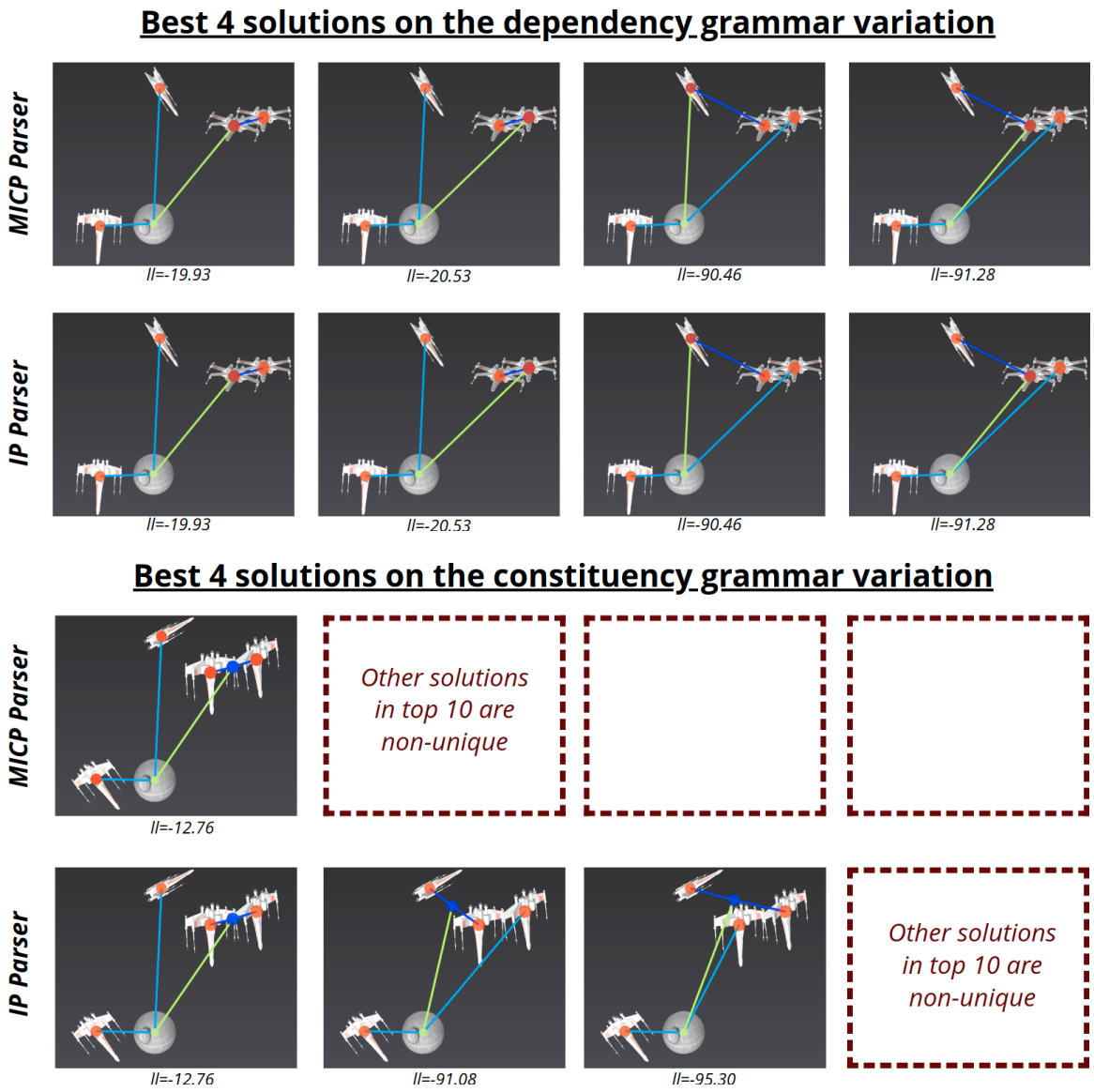


Figure 3-7: The best 4 parses for example scenes drawn from the dependency (**top**) and constituency (**bottom**) variations of the singles-pairs grammar. Each parsing method produces 10 *integer-unique* solutions, from which the best 4 unique parse trees are extracted. Integer solutions do not necessarily correspond to unique parse trees, depending on the grammar and formulation details, so 4 unique solutions are not always available. Each scene is annotated with its total tree log-probability ("ll"). The optimal parse tree from each technique matches or beats the ground-truth parse tree that was used to generate the scene.

integer solutions all correspond to functionally identical parse trees.[7]

The situation is better, but not perfect, for the IP parsing formulation: each intermediate unobserved node with unknown pose is associated with a list of binary variables corresponding to the choice of which pose (from a finite set of proposals) that node will take. Different assignments of these variables do not necessary correspond to unique tree *structures*, but they should correspond to unique scene trees with differently placed nodes as long as the set of proposed node poses are unique. For the scene illustrated in Figure 3-7, each truly unique scene tree corresponds to 4 different integer solutions, so we get a total of 3 unique scene trees in the top 10 integer solutions (corresponding to solution #1, #5, and #9). 2 of those integer solutions correspond to different pose proposals that converge to the same optimal tree, and 2 of them correspond to an unbroken symmetry in the AND node logic used to implement the Pairs node.

3.7.2 Case studies and comparison on sink grammar

In order to demonstrate that our parsing procedure can be applied to a slightly more complex and realistic class of scenes, we show that these techniques can be applied to parse the dataset of physically realistic cluttered sink scenes we assembled to correspond with a sink grammar (both are detailed in Section 2.6.3). This sink grammar captures some of the typical structure present in cluttered sinks full of objects: that bowls, plates, and cups can appear independently at random poses within the sink, but that objects also tend to be placed inside of bowls and on top of plates. Capturing these patterns and allowing for the production of 9 unique dish models (3 bowls, 3 cups, and 3 plates) leads to a grammar with 19 node types and

[7] We suspect the reason why *so many* of the top solutions are redundant is an unfortunate combination of the objective and piecewise convex $SO(3)$ constraint approximation. Rotation matrix decision variables enter the objective linearly as part of the Bingham distribution log-density, so at an optimal solution they'll always be driven to constraint boundaries and corners. These corners are the transition points between the convex regions of the piecewise-convex constraints used to construct $SO(3)$ constraints and piecewise McCormick envelopes; at these points, multiple different binary variable settings corresponding to the activation of one of the intersecting regions are feasible. The different elements within the rotation matrix are each subject to this problem, meaning there could be an exponential number of equally-valid integer solutions all corresponding to a globally optimal cost and corresponding parse tree.

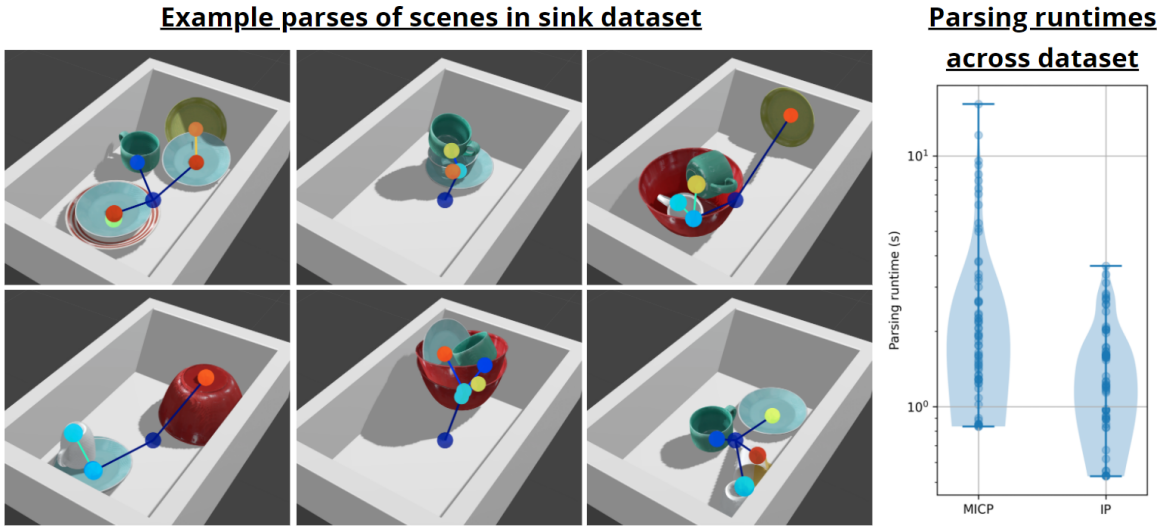


Figure 3-8: **Left** Example MAP parses from 6 scenes from the sink dataset. Both parsing methods return the same MAP parse tree for all scenes in the dataset. **Right** Plots visualizing parsing process runtime across the 68 scenes in the dataset. Each point is a run of the corresponding parsing process asked to produce the top 10 scenes. The parsing methods returned the same MAP parse tree for all scenes; timing differences can be primarily attributed to differing setup times and minor formulation differences.

produces a supertree with 517 nodes.

Figure 3-8 illustrates that our parsing techniques recover qualitatively reasonable parse trees with reasonable runtime. Because the sink grammar is structured as a dependency grammar, both techniques agree on the set of top N MAP parse trees and solve in similar time.[8] Figure 3-9 further illustrates that our parsing techniques recover diverse and meaningful parse tree structures for these scenes, each of which hypothesizes a different underlying hierarchy for the scene.

[8]The variation in runtime observed in Figure 3-8 is likely *partially* due to differences in setup time. These optimizations are set up using the Python bindings for Drake [59] before being passed to Gurobi. The size of the supertree for this grammar leads to a large number of decision variables and constraints to be allocated and manipulated, which can be particularly slow in Python. The MICP, with its extra continuous pose variables, is more complex to set up for this grammar. However, the handful of outliers over 10 seconds under the MICP method appear to be due to the solver.

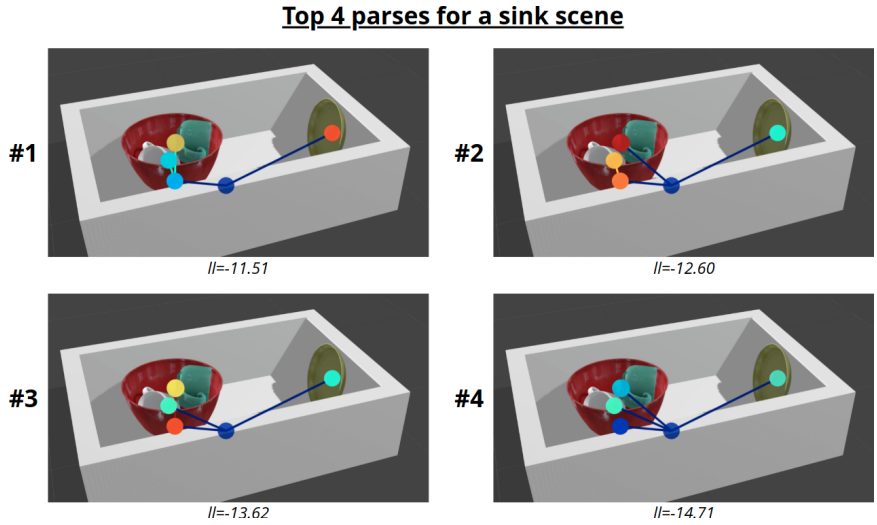


Figure 3-9: The best 4 parses for an example scene from the sink dataset. Because the sink grammar is constructed as a dependency grammar with no unobserved equivalent sets, both MICP and IP parsing techniques produce the same solution set in similar time.

3.8 Discussion

In this chapter, we have demonstrated that we can use a *supertree* of a scene grammar as a scaffold for performing a mixed-integer optimization over the space of all scene trees in the grammar; and that two variations on a mixed-integer convex parsing approach can be successfully applied to multiple example scenes and grammars. Our results broadly indicate the success of both parsing methods, but also illustrate some important weaknesses to keep in mind when deploying these methods in practice. Here, we remark on some of the principle scaling concerns facing these methods, along with ideas on how to address them; as well as other ideas on extensions to more complex parsing problems.

3.8.1 Scaling concerns and alternative optimization approaches

Constituency vs dependency grammars

Like any mixed-integer convex optimization, our parsing procedure faces worst-case exponential time complexity, which is unfortunately corroborated by Figure 3-6. The worst-case grammar – the singles-pairs constituency grammar, which has many un-

observed intermediate nodes whose poses are involved in nonconvex relationships – scales particularly poorly beyond scenes with a handful of objects. As noted previous, the most direct resolution for this problem is often to reformulate the grammar itself: many scenes can be equivalently stated with a *dependency* grammar that avoids the introduction of unnecessary intermediate nodes. As is clear in Figure 3-6, simply making this switch can improve parsing runtime by orders of magnitude. The core difference between the dependency and constituency variation of, for example, the singles-pair grammar is that parsing a scene with the dependency variation does not require inferring the pose of the unobserved "Pair" nodes. (To see this, note that in Figure 2-2, each equivalent set (drawn as a shaded box) contains an observed node.) Moving from a *constituency* to a *dependency* formulation has significant impacts on the parsing process:

- The MICP parsing formulation becomes significantly easier to solve: because all equivalent sets are observed by definition of a dependency grammar, all pose decision variables will be exactly equal to the pose of one of the observed nodes, and no costly $SO(3)$ approximation constraints need to be applied. Indeed, for dependency grammars, the MICP and IP parsing procedures are practically identical if the observation short-circuiting trick is applied (Section 3.4.3).

- The IP parsing formulation requires no proposals (as, by definition of a dependency grammar, there are no intermediate node poses that require proposing), and so the IP parsing formulation will always be able to find the optimal parse (as long as the supertree recursion depth limit has not cut off vital tree structure to explain the scene).

- NLP post-processing is unnecessary for both techniques, as there are no continuously varying uncertain pose variables that may have been poorly proposed or loosely approximated.

Note, however, that if the grammar does not contain rules that produce bilinear relationships – if, for example, only *world-frame* translation offset and rotation dis-

tribution rules are used – then the MICP formulation becomes vastly easier to solve, as most of the difficult nonconvexities will not be present.

Supertree complexity scaling

The supertree will eventually, itself, become a barrier to scaling, as it will grow exponentially with the complexity of the grammar. Reformulations of this problem might focus on removing or limiting the exponential growth of the supertree by counting how often each node type can possibly occur in a draw from the grammar. In the same vein, the supertree interpretation of the parsing problem may itself lead to parsing techniques that do not require explicitly constructing the entire tree at once, but instead rely on lazy evaluation of supertree structure during a branch-and-bound-style solution exploration.

Alternative optimization approach: bilinear alternation

While we focused on formulating the parsing problem, including its nonconvexities, as a single optimization problem, it's possible that we could instead approach [some of] the nonconvexities through an alternation strategy. The worst of the nonconvexities occur in bilinear relationships from the Parent-frame Normal offset and Parent-frame Bingham offset rules from Table 2.1. The former involves bilinear relationships $n^p.r^T n^p.t$ and $n^p.r^T n^c.t$, and the latter involves the bilinear relationship $n^p.r^T n^c.r$. One could form a binary partition of these pose decision variables and alternate between optimizing one set of poses and node activations while keeping the other set fixed in a bilinear alternation approach. Using this approach would remove the loose McCormick envelope approximations to these bilinear terms – at the possible expense of convergence and optimality guarantees.

Alternative optimization approach: SDP relaxations of $SO(3)$

An increasingly popular approach to optimization over $SO(3)$ is to construct convex semi-definite programs (SDPs) that, in practice, form tight relaxations of underlying optimization problems of interest. Given a rotation parameterized by the 9 elements

of its rotation matrix, [80] provides a spectrahedral (i.e. semi-definite) constraint that constraints those elements to lie within the convex hull of $SO(3)$. The formulation of this constraint is functionally very similar to our construction of the quaternion outer product in Section 3.4.3. In many contexts, this constraint is tight – for example, when the optimization objective over those rotation matrix elements is linear, we know the optimal solution will (modulo some corner cases) lie on the constraint surface which corresponds to $SO(3)$. TEASER [81] takes advantage of this property by utilizing a similarly-structured tight SDP relaxation as part of a point cloud registration algorithm. In our context, we may be able to utilize this SDP relaxation in place of the mixed-integer $SO(3)$ approximation we currently use, with hope that we see the same tightness results as [81].

A major barrier to applying this technique is that node rotation matrices still enter into two kinds of bilinear relationships, as described in Section 3.8.1. Many approaches to solving the point cloud alignment problem (including ICP and TEASER) rely on separately solving for the translation and rotation – but, to the best of our knowledge, these tricks can't be applied in this case due to tight coupling of the rotation and translation variables in the objective. There is a chance that, upon careful inspection of the way the translation and rotation decision variables enter the objective (and interact, in particular, with the Normal and Bingham distribution concentration and covariance matrices), one may uncover useful symmetries that might enable semi-definite approximations of these remaining relationships.

3.8.2 Future directions

The role of physical feasibility in scene parsing

As we have discussed in Section 2.4, physical constraints have a strong influence on the layout of objects in a scene, but can be difficult to cleanly integrate into an inference framework. Our framing of the scene parsing problem carefully sidesteps this issue by assuming we've already been handed a scene that satisfies all of the physical constraints that we expect. As a result, we don't need to reason about these

constraints at all during parsing, as we fix the location of all geometry that might participate in those constraints.

However, if one were to introduce a more complex observation model – even just the introduction of noise in the object pose observation process – these constraints become relevant to parsing and need to be satisfied. Consider, for example, incorporating non-penetration as a canonical and critical physical constraint. If the parsing process requires inferring the *true* locations of objects in addition to the existence, poses, and relationships between any latent nodes, then the set of inferred object poses must be constrained to be non-penetrating. One way this constraint could be addressed is at the NLP-postprocessing layer, as the non-penetration constraint is easily expressed as a nonlinear function of all object poses and geometries.

Possible applications to hybrid control and verification

Finally, we hope that the details of these parsing techniques have value to the reader beyond methods for recovering MAP parse trees. Control synthesis and verification has turned its eye towards practical robot manipulation systems, but currently has limited tools for reasoning about the behavior of a robot facing wildly varying *numbers* of objects. These parsing formulations demonstrate that it's possible to perform principled optimization over the space of *varying-size scenes*, which could be turned towards, for example, mixed-integer adversarial example search as formal verification strategy.

Chapter 4

Parameter Estimation

4.1 Introduction

A scene grammar provides a probabilistic procedural model over environments $p_\theta(o, T)$, with a set of parameters θ that control the shape of the output distribution. These parameters control how often each node produces each possible set of children, and the shape of the relative pose distribution for each child conditioned on its parent. Despite these parameters having intuitive meanings – like the relative occurrence of one object types versus another, or the typical relative position of a child object in the frame of reference of its parent – finding appropriate settings of these parameters can still be arduous and difficult in sufficiently complex grammars. Further, one might desire to optimize or do inference over these parameters to find the best possible parameter setting to capture the distribution one has observed in a dataset of observed environments. To perform that inference, we need to tackle the problem of *parameter estimation* for scene grammars.

4.2 Related Work

A subset of the scene grammar (and broader procedural scene modeling) literature is concerned with this parameter estimation problem. A significant amount of previous work that utilizes scene grammars performs some amount of parameter estimation

(or even grammar inference) in order to generate the grammars that they then use for parsing. However, many of those works assume access to labels of the structure captured by the grammar: for example, [17] assumes access to consistently labeled scene hierarchies, and [51] directly collects statistics about object occurrences and geometry from readily-available data. [82] optimizes grammar parameters with a gradient update (similar in spirit to the variational inference strategy we use in [4]), but likewise does not need to solve as complicated a parsing problem to perform that update. Unlike these approaches, we aim to optimize grammar parameters *without any parse tree labels*. This is a tricky process, as the parse tree provides the link between the observed scene and the grammar parameters; the missing parse trees are instead latent variables that must be inferred. We utilize our scene parsing procedure to recover the most likely parse trees, which we leverage in an approximate expectation-maximization (EM) framework to perform parameter estimation.

A fundamentally different tactic from our own is to instead align the parameters by producing many samples from the model, and tweaking the parameters to make that output distribution "align" with a target distribution according to a metric. This frequentist alignment method is used in MetaSim [18] to control the output distribution of a neural distribution transformer, and by the follow-up work in MetaSim2 [20] to directly adjust parameters of a recurrent neural variation of a scene grammar. This approach is appealing in that it bypasses the difficult parsing and inverse modeling requirement, but requires engineering a distribution comparison metric to drive the alignment: [18] and [20] use two-sample tests in a deep embedding space computed from rendered images for this purpose. BayesSim [34] uses another distinct tactic to recover (posteriors over) dynamics parameters in simulator models: they use simulated data train an inverse function that predicts latent parameters from observed rollouts, and apply that inverse function to real rollouts to estimate the distribution of latent parameters in reality. This approach bypasses any computation of the simulator likelihood function (which is typically impossible to get from a black-box simulator), but requires formulation of a proposal engine that can predict latent model parameters from observed outputs – a problem that is particularly difficult in

our case, where latent model parameters consist of entire scene tree structures.

4.3 An Approximate EM Approach

Assume that we are handed a grammar model of observed scenes $P_\theta(o)$ with parameters θ, an initial guess for the parameters θ^0, and that we're supplied with a set of fully observed environments $\mathcal{O} = \{o\}$. Each observed environment is a collection of rigid bodies with known poses and geometries matching those produced by the grammar, assumed to be in the language of the grammar. We wish to find the grammar parameters that maximize the [log]-evidence of the data

$$\theta^* = \arg\max_\theta p_\theta(\mathcal{O}) = \arg\max_\theta \log p_\theta(\mathcal{O}).$$

We can tackle this by applying the standard expectation maximization (EM) algorithm [83]. Evaluating the expected value of the log-likelihood of an observed scene o under the distribution of parse trees induced by our current parameter values θ^k as

$$Q(\theta|\theta^k) = \mathbb{E}_{T\sim p_{\theta^k}(T|o)}\left[\log p_{\theta^{k+1}}(o,T)\right],$$

we choose θ^{k+1} to maximize these expectations independently across each datapoint

$$\theta^{k+1} = \arg\max_\theta \sum_{o\in\mathcal{O}} Q(\theta|\theta^k).$$

While we cannot analytically write down the expectation as in classic EM, we can instead approximate it by enumerating over the set of M most probably parse trees T recovered by our parsing procedure from Chapter 3. This allows us to form an alternating procedure:

1. For each observation and current parameter guess θ_k, collect the M most likely trees $T_i \sim p_{\theta^k}(T|o)$ as the top M best solutions from our MICP parsing procedure, for a user-specified M.

2. Determine the relative weights of each tree by

$$p_{\theta^k}(T|o) \approx \frac{p_{\theta^k}(T_i, o)}{\sum_{T_i} p_{\theta^k}(T_i, o)}$$

.

3. Update

$$\theta^{k+1} = \arg\max_{\theta} \sum_{o \in \mathcal{O}} \left[\sum_{i}^{M} \log p_{\theta}(T_i, o) p_{\theta^k}(T_i|o) \right]$$

(1) and (2) calculate the expectation term $\mathbb{E}_{T \sim p_\theta(T|o)}$ by approximate enumeration: we use the MAP parsing procedure to gather the set of trees that have the (approximately) highest posterior likelihood, and assume that this set is the set of all trees that can explain the scene with nonzero probability, so that we can calculate their relative posterior likelihoods for use as weights in (3).

(3) calculates an update for each parameter based on the population of parse trees for each observation. For our grammar formulation, this can be done in closed form separately for each parameter. For each node type, the relative rule occurrence rates can be determined by counting how often each rule occurs under that node type in the set of all parse trees for all observations. For each rule type, the parameters of the rule's underlying distribution (i.e. Normal or Bingham) can be fit to the set of all parent/child pairs using that rule type in the set of all parse trees for all observations.[1] However, if additional rule types are added for which this fitting cannot be done in closed form, it can instead be done by gradient descent, forming a variational-EM approach (see [4]).

4.3.1 Measuring distances between sets of scenes

In order to measure the success of the parameter fitting process, we require a metric with which to compare two populations of scenes. As scenes can contain varying types and numbers of objects, there is no unfortunately no standard or obvious way

[1] We use the Bingham distribution implementation from [84], which fits Bingham distribution parameters to an observed set of samples by the method of moments.

to perform this comparison – even writing a distance metric between two *individual* scenes is nontrivial. We propose two metrics here, which we use to evaluate our experiments.

1. **Dataset log-evidence under the model:** We can estimate the total evidence under the model of the set of scenes in the target dataset. We could estimate this value by marginalization:

$$p_\theta(\mathcal{O}) = \sum_{o \in \mathcal{O}} \int_T p_\theta(o, T)$$

   in which we sum over scenes o from the dataset $\mathcal{O}$; for each scene, we enumerate over parse trees T that might explain the scene. In practice, we approximate this sum by instead summing over the top M parse trees as gathered by the parsing procedure, which should be the primary contributors to the sum over all possible parse trees.

2. **Two-sample testing at an object level with MMD:** We can produce a population of samples $\mathcal{O}^* = \{o^* \sim p_\theta(o, T)\}$ by randomly drawing scenes from the grammar, and compute a distribution distance measurement $D(\mathcal{O}, \mathcal{O}^*)$. In order to make it easier to apply statistical tools, we first break these populations up by object: we take $D(\mathcal{O}, \mathcal{O}^*) = \frac{1}{K} \sum_N^K D_N(\mathcal{O}_N, \mathcal{O}_N^*)$ for each observed node type N and corresponding set of individual observations of node type N O_N and O_N^*. Note that this throws away information about which objects co-occurred: we are now only inspecting the distribution of poses of each class of object across each dataset.

   To evaluate D_N, we utilize the Maximum Mean Discrepancy (MMD) statistic [85] which measures the distance between two distributions $p(\mathcal{O}_N)$ and $p(\mathcal{O}_N^*)$ by comparing the average discrepancy of the moments of embeddings of samples from both distributions. If this embedding is to a reproducing kernel Hilbert space (RKHS) with kernel $\mathcal{K}$, then it can be shown that this metric goes to zero if and only if *all* moments are matched (i.e. the distributions are iden-

tical). In this case, the MMD can be estimated through the kernel trick by evaluating the the provides a pairwise distance between samples $\mathcal{K}(x_1, x_2)$ for all sample pairs $\langle x_1, x_2 \rangle$. Standard implementations of MMD utilize a standard RBF kernel; however, since our samples are objects in $SE(3)$, we need to handle them carefully. For a pair of objects $o_1 \in O_N$, $o_2 \in O_N^*$ with poses $\langle o_1 = (t_1, R_1), o_2 = (t_2, R_2) \rangle$, we use

$$k(o_1, o_2) = \exp\left(-\alpha\big(||t_1 - t_2||_2^2 + \beta \times \mathit{angle}(R_1^T R_2)\big)\right)$$

with α determining the scale of the kernel, β allowing different weightings of the translation and rotation components, and *angle* converting the differential rotation matrix to the angular distance (in radians) between the two rotations. This kernel reaches its maximum when the poses are identical, and falls off as a Gaussian otherwise. This function is symmetric and positive definite, and thus corresponds to a RKHS by the Moore-Aronszajn theorem [86], thereby fulfilling the property in the MMD that we seek.

4.4 Experiments

4.4.1 Comparison with EM for GMMs

Grammars of the type described in this work could be viewed as a vastly more complex form of mixture modeling in which the mixtures can be stacked hierarchically. The EM parameter fitting algorithm used is commonly applied to standard mixture models: in a [e.g. Gaussian] mixture model with $N^{modes} = 3$ components, the Expectation step corresponds to computing correspondence probabilities of each datapoint to each of the N^{modes} components, and the Maximization step is performed by enumerating over those correspondences to find new parameter estimates. For comparison, consider a grammar that implements the same logic: in the GMM grammar (Figure 2-1), each observation is a single point that can be explained by one of three production rules corresponding to each of the 3 mixture components. If we run

our approximate EM algorithm and ensure that the parsing procedure always returns the top N^{modes} parse trees, then the parsing procedure will enumerate all possible assignments of an observed point to mixture modes. Our parameter maximization step will, in turn, consider over all possible assignments of each observed point, and thus precisely reproduce that *exact* EM procedure for GMMs. Stated differently, in this case of GMM fitting, our *approximate* EM procedure reduces to *exact* EM, as we know the procedure enumerates over all possible parse trees for each observation.

To illustrate that this reduction occurs, we perform an experiment in which we ran our EM algorithm alongside a baseline vanilla EM algorithm [1]. In each trial, we randomly sample an $M = 3$ component ground truth GMM and sample 100 observed points $\in R^3$, which are passed to each EM algorithm in order to recover the GMM parameters. Each EM algorithm is seeded from the same random initial condition and run for 5 iterations (which was typically enough for convergence). We use the element-wise Earth mover's distance between the ground truth model and estimated model as the score. The correlation between the scores of each method are illustrated in Figure 4-1; the methods performed nearly identically, modulo noise due to minor numerical differences in the handling of small mixture weights.

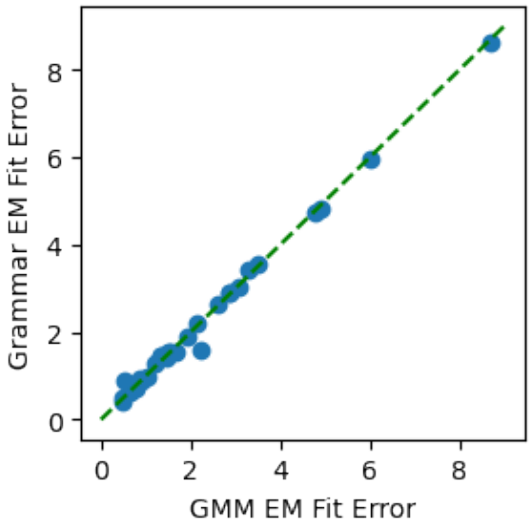


Figure 4-1: Correlation between fitting errors of a baseline GMM EM implementation [1] and our approximate EM algorithm on 30 randomly-initialized GMM parameter estimation trials. Each dot is a trial. Each score is computed as the Earth mover's distance between the ground truth and corresponding estimated model. The close correlation in fitting error empirically supports that our method reduces to vanilla EM in this case.

In the case of GMM parameter estimation, our technique is certainly overkill, and we take a massive performance hit due to the significant overhead of setting up and solving many mixed-integer optimization. (Running the baseline GMM fit on this toy

example is nearly instantaneous, while running our parameter fitting for a dataset of 100 points routinely takes 3 minutes, or an average throughput of 100ms per MICP solve.) Our method pays off in larger-scale examples where trivial EM algorithms are not possible to write.

4.4.2 Case study on the dimsum table grammar

Our table grammar (Section 2.6.4) is an example of a grammar whose parameters have clear physical meaning, but whose precise values are may not be immediately obvious a priori. Exactly how many plates, bowls, and shared serving trays appear on a typical table? Does that number vary by restaurant, or by time? How tightly clustered do those objects tend to be on the table surface? These parameter estimation tools provide an avenue for tackling these questions.

Parameter estimation

To create a target dataset of tables, we set our grammar parameters to known reasonable values that produce a qualitatively-reasonable dataset, and sample 100 scenes. To ensure that our scenes are physically reasonable, after randomly sampling a tree structure and initial set of poses, we use the HMC conditioned sampling approach from 2.4 to generate a chain of samples that satisfy the *Items-on-table* and *Items-non-penetrating* constraints (detailed in 2.6.4) that ensure all items are within the bounds of the tabletop and are a sufficient distance from each other. We inspect the trees produced by the chain and accept the latest tree in the chain that satisfies the constraints as our reasonable sample. Finally, we simulate the objects in the tree forwards in time to a statically stable configuration to place them on the tabletop.[2] Exemplar trees from this dataset are illustrated in Figure 4-2. These scenes contain between 7 and 19 objects each, and the supertree for this grammar contains 186

[2]While this static-stability constraint could, in principle, be framed as a constraint, it is both very stiff and highly dependent on potentially complex object geometry, and so is difficult to address in practice. In this scene, objects that follow the grammar and satisfy the non-penetration and on-table constraints are very close to static stability already: forward simulation typically only moves objects a few centimeters downwards to be precisely in contact with a support surface.

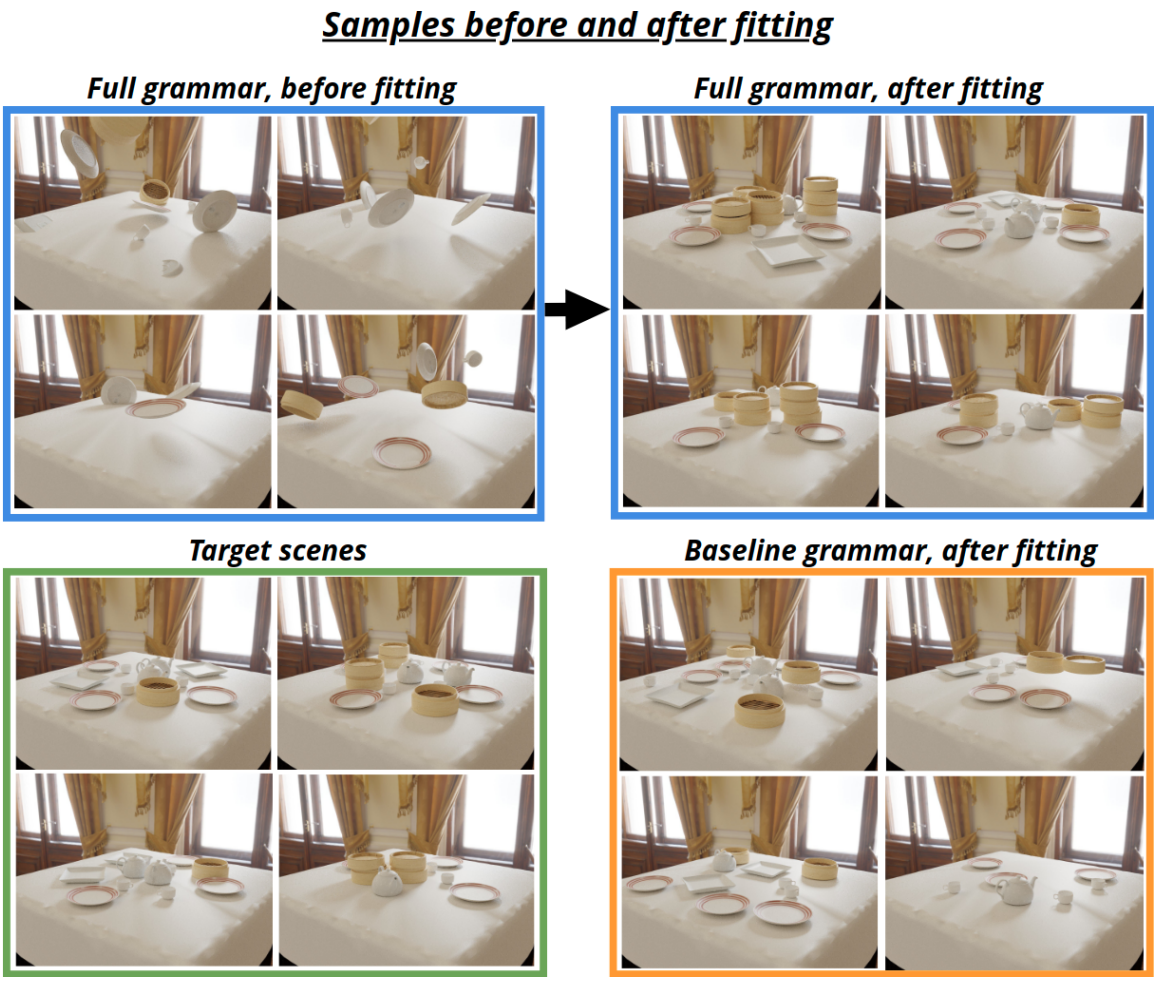
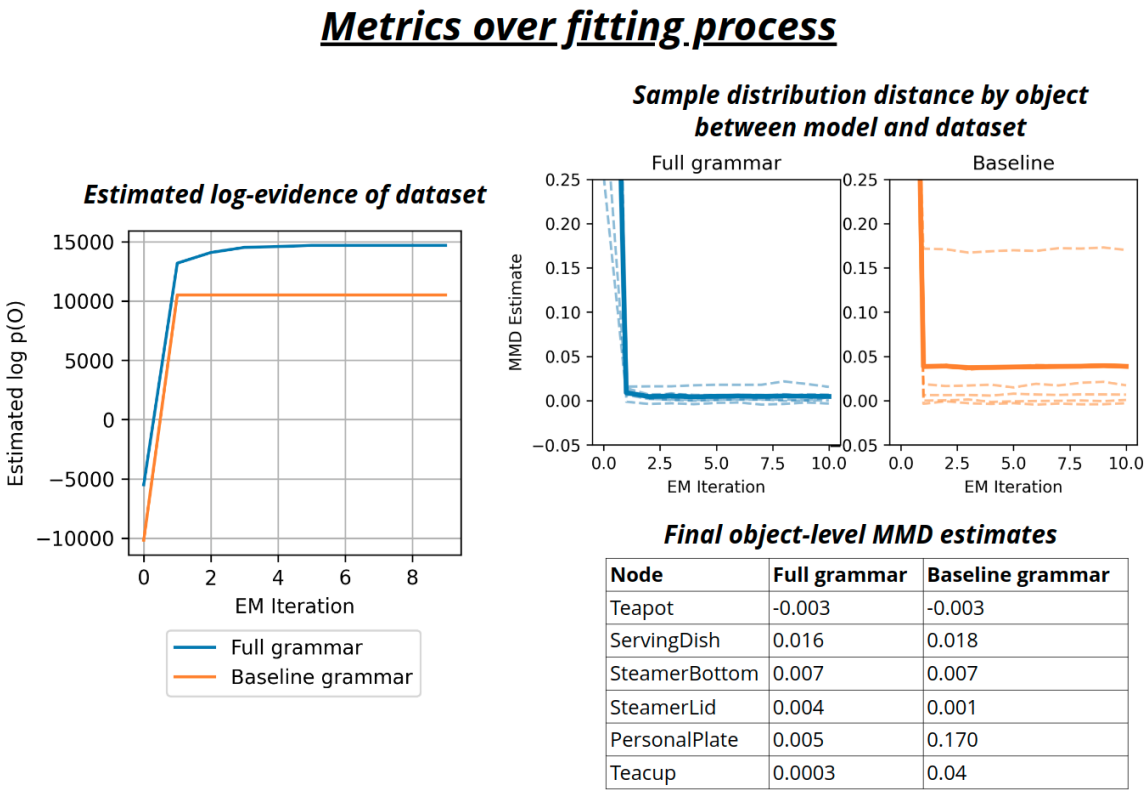


Figure 4-2: **Top**: Samples from before and after approximate-EM fitting of the table grammar to the target dataset. The scenes before and after fitting are sampled under the same set of pose constraints that require object origins to be above the table surface with adequate horizontal clearance to not interpenetrate other objects. Additional regularity in object placement is due to tightly fit grammar parameters. **Bottom left**: Samples from the target dataset. **Bottom right**: Samples from the baseline grammar after fitting to the target dataset. These scenes are sampled under similar pose constraints to the full grammar. Note that these scenes fail to accurately capture the arrangements of individual place settings or steamer stacks.



Node	Full grammar	Baseline grammar
Teapot	-0.003	-0.003
ServingDish	0.016	0.018
SteamerBottom	0.007	0.007
SteamerLid	0.004	0.001
PersonalPlate	0.005	0.170
Teacup	0.0003	0.04

Figure 4-3: Evolution of the estimated log-evidence of the dataset under the model (**left**) and the two-sample MMD distribution distance metric between the model and dataset (**right**) while fitting the table grammar and its baseline version to the target dataset. MMD traces across training are separated out for each object (dashed lines) alongside the mean across objects (solid line). MMD estimates at the last iteration of both methods are included for reference.

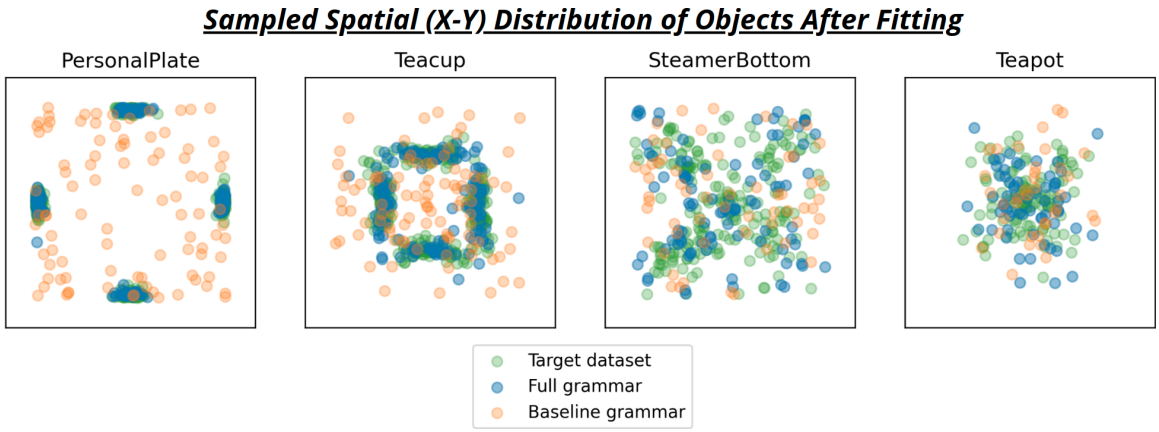


Figure 4-4: Scatterplots of sampled x-y locations of a handful of object types from the Table grammar generated by sampling 30 scenes from each model (under the same constraints used to generate the target dataset) and plotting the x-y location of all object of the corresponding type from those scenes on the table surface. Samples from the baseline (orange) and full (blue) grammar models after fitting are overlaid with samples from the target dataset (blue). The bounds of each plot correspond exactly to the size of the table.

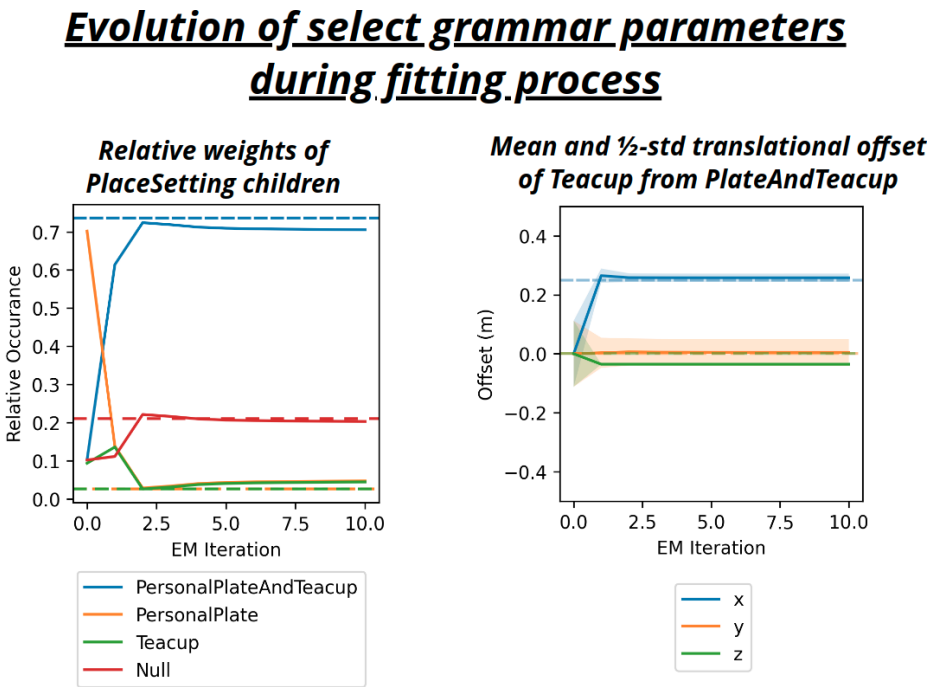


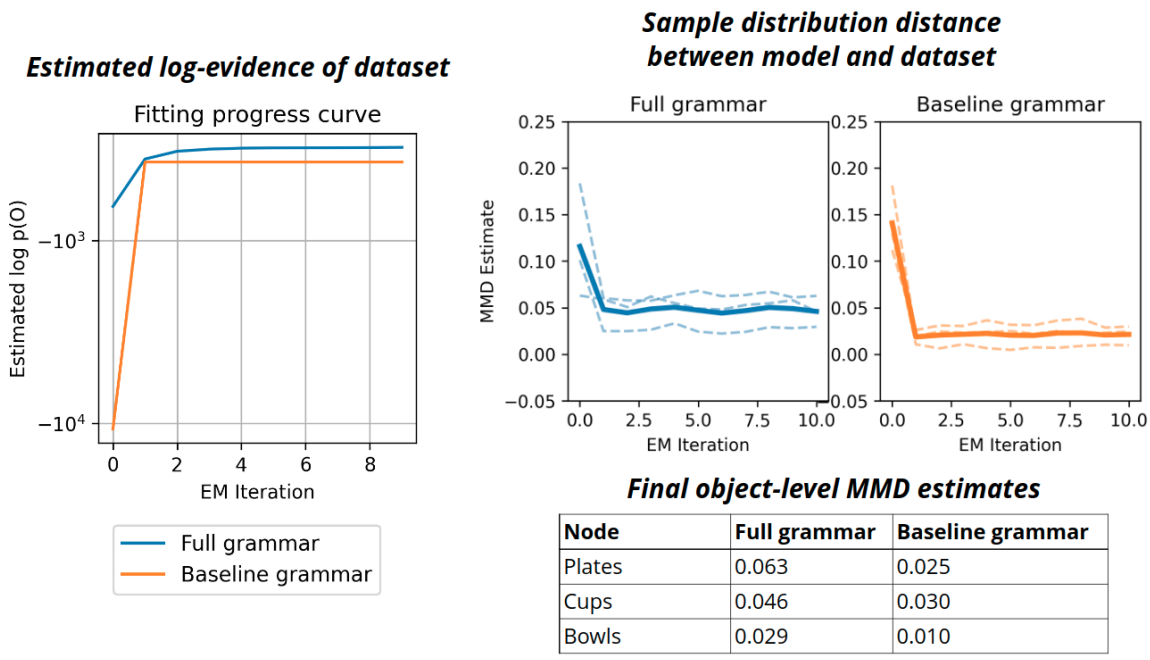
Figure 4-5: Plots of some selected parameters of the table grammar during the fitting process (solid lines) compared to the ground truth values used to generate the target dataset (dashed lines).

nodes.

We initialize a grammar with naive parameters that are far from capturing the distribution of interest[3], and apply our approximate EM procedure using the top 10 parse trees per scene. Figure 4-3 illustrates that our parsing procedure is able to recover the target distribution and the parameters of interest. Figure 4-4 visually illustrates this distribution match. The pose-specific kernel used to estimate the MMD was used with $\beta = 0.1$, as we expected translation errors to be an order of magnitude smaller than angular errors. The full grammar outperforms the baseline grammar on both metrics: it is able to both assign more density to the target scenes (owing to its more specific rules), and better captures the distributions of each object type. The primary object type for which the full grammar performs significantly better (in terms of MMD) is on the Plate object: in the target dataset, plates are tightly clustered around the seating locations of the table, which only the full grammar has the expressive power to accurately to capture. Figure 4-5 illustrates parameter traces across training for a handful of parameters in the grammar; these parameters converge to close to the ground truth values used to generate the target dataset.

Additionally, we compare against a simpler scene modeling approach, as implemented in a baseline grammar. This baseline grammar (detailed in Section 2.6.4 and illustrated in Figure 2-8) models the table as having a random number of objects of each type, where each object's pose is drawn from a Table-frame pose distribution specific to the object type. Note that this grammar is less expressive than our full Table grammar: it cannot capture the covariance of Steamers in a stack, or cups appearing next to plates. We apply our approximate EM procedure to this grammar as well and include its performance for comparison in Figures 4-2, 4-4, and 4-3.

[3]In particular, we set Gaussian pose offset means to zero and variances to 0.05, reflecting a naive but not enormous distribution of all object types. All other parameters are randomly initialized from broad domains. Setting the pose offsets to zero-mean ameliorates convergence to suboptimal solutions in which e.g. cups are associated with plates that happen to co-occur across the table.



Node	Full grammar	Baseline grammar
Plates	0.063	0.025
Cups	0.046	0.030
Bowls	0.029	0.010

Figure 4-6: Evolution of the estimated log-evidence of the dataset under the model (**left**) and the two-sample MMD distribution distance metric between the model and dataset (**right**) while fitting the sink grammar and its baseline version to a dataset of example sinks. MMD traces across training are separated out for each object class (dashed lines) alongside the mean across all object classes (solid line). MMD estimates at the last iteration of both methods are included for reference.

4.4.3 Case study on the sink grammar

Like the table grammar, our sink grammar (Section 2.6.3) has physically intuitive parameters whose precise values would benefit from fine-tuning to data. In particular, the sink grammar has rules for capturing the distribution of objects within a sink, including specific rules capturing the stacking of plates on top of each other, and the grouping of objects inside of bowls. We can use our parameter estimation techniques to analyze how often these different scene structures occur, and what the precise pose distribution of a "objects in a bowl" looks like in the first place.

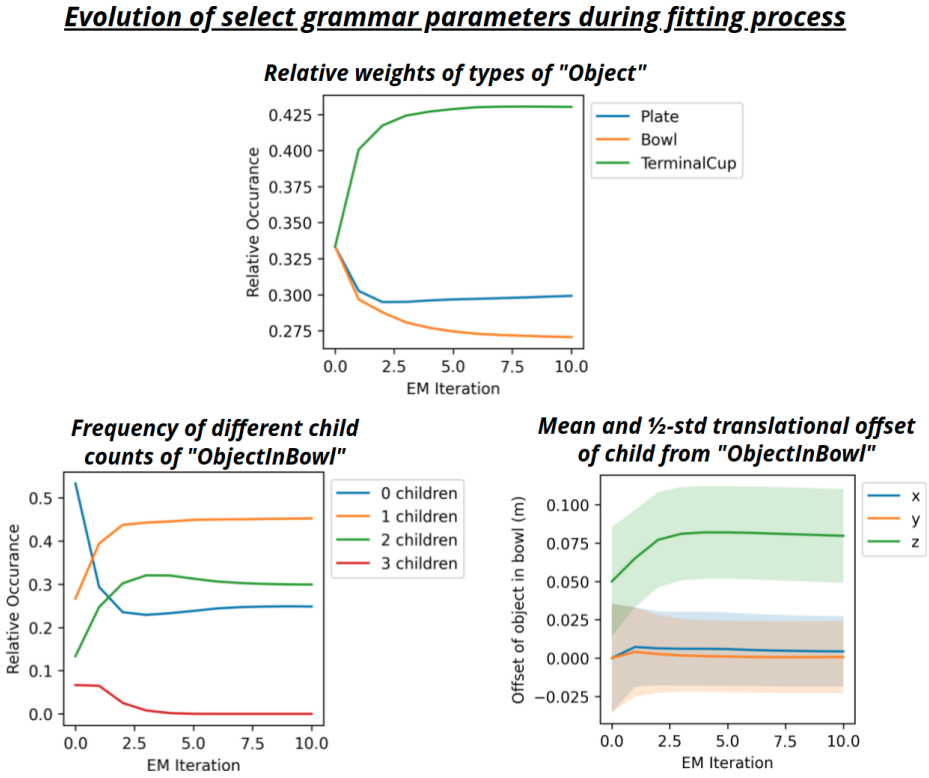


Figure 4-7: Plots of some selected parameters of the sink grammar during the fitting process.

Parameter estimation

As detailed in Section 2.6.3, we collected a modest dataset of 68 cluttered sink scenes, each containing between 1 and 10 objects. After initializing the grammar parameters to qualitatively reasonable values[4], we apply our approximate EM algorithm using by finding the top 10 parses for each scene via the IP parsing strategy from Chapter 3; updating parameters to best match those parses; and repeating for 10 iterations. Quantitative analysis and select parameter histories from this process are illustrated in Figures 4-6 and 4-7. These results indicate that our parameter estimation process is able to significantly improve the specificity of our grammar under both of our metrics. The full grammar beats the baseline grammar in terms of model specificity as captured by the estimated log-evidence over the dataset; however, the baseline slightly beats out the full grammar in terms of a two-sample population comparison using our MMD metric. This may be a factor of lack of sensitivity in the MMD metric given our relatively small target dataset; or could be an indication of overfitting of our model. Given the complex, cluttered, and likely very non-Gaussian distribution of objects within the rectangular bounds of the sink, perhaps neither approach has the expressive power to fully capture the spatial arrangements of these objects.

Outlier detection

While this grammar describes a class of scenes that are beyond our ability to meaningfully produce samples (due to its highly cluttered nature and complex object geometries – see discussion in Section 2.6.3), it still provides value as a tool for scene understanding. A benefit of the parameter estimation process is that once a grammar is tuned to a given dataset, it becomes more sensitive and capable of detecting outlier scenes that do not match that data distribution. Figure 4-8 illustrates this point: we show that parsing a handful of intentionally-crafted "outlier" scenes with the *fit* grammar reveals that they tend to be less likely under the model than the rest of the scenes. These outlier scenes were created to intentionally violate the rules

[4]We have found that initializing with at least approximately accurate parameters is important, as the EM process is a local optimization that is at the mercy of local minima.

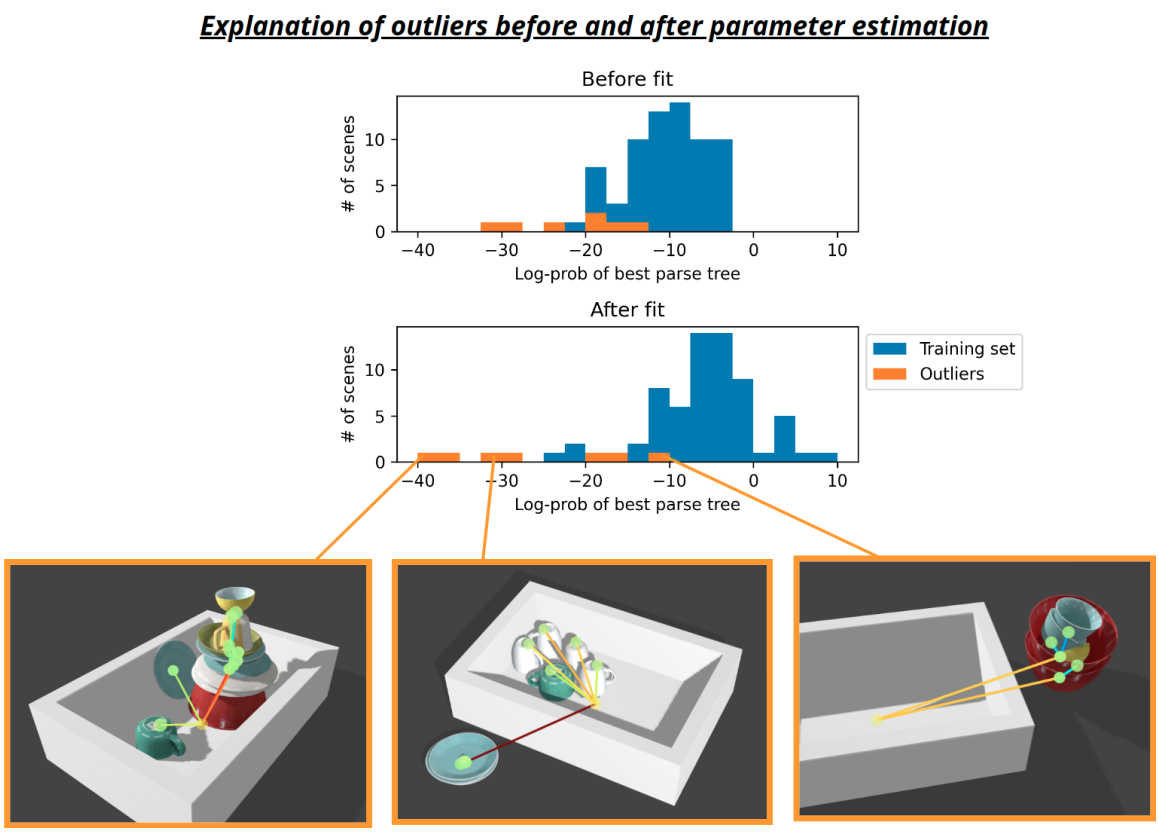


Figure 4-8: Histograms of the best parse tree score of each scene in the dataset before and after fitting. The fitting process increases the likelihood assigned to trees in the target dataset, at the expense of likelihood of trees not following the same distribution. Example parse trees, with nodes and edges colored by their calculated likelihood (red-low to green-high), are shown for a few of the outlier scenes.

used to assemble the original scenes: the original sink scenes all have objects placed *within* the sink in small or orderly stacks, while the outlier scenes consist of objects outside of the sink or stacked unusually high. Because we can inspect scene parses at a node and rule level, this tool provides a *part-level explanation* of which parts of the scene are unusual: the orange and red edges in Figure 4-8 indicate low probabilities on the corresponding production rule in the parse tree, indicating that the specific object they connect is in an unusual configuration. This is particularly relevant in the highest-scoring outlier scene (i.e. the rightmost illustrated scene in the figure): this scene has two bowls which are clearly outside of the sink, but the rest of the objects within the bowls follow the rules of the grammar.

As an important caveat, it's important to see that comparisons of tree scores $P(T)$ between scene trees with different structure are only partially informative, since these densities are not truly comparable. In practice, we have seen that trees with more objects tend to have inflated scores compared to smaller trees that should be considered to be equally likely, since we sum up the score of individual rules that tend to have positive individual scores. In this case, this is reflected in "outlier" trees that still achieve good scores by having the one or two low-scoring rules that explain the outlier objects counterbalanced by many adequately-scoring rules for other reasonably-placed objects.

4.5 Discussion

In this Chapter, we have demonstrated that we can perform parameter estimation of our spatial scene grammar model using an approximate EM procedure that iteratively recovers parse trees from a dataset of observed scenes, and updates grammar parameters to increase the likelihood of those parse trees. We have demonstrated that our method recovers ground truth parameters from random initializations; and we've used this method to show that we can use our spatial scene grammars to construct models that are more *specific and accurate* than typically-deployed baseline models.

4.5.1 Extension to variational EM

The use of closed-form parameter updates in the M step is a boon to our technique: as indicated by the plots in Figures 4-2 and 4-6, the parameter estimation procedure converges to locally optimal parameter settings in a handful of iterations. These closed-form updates are possible because all of the node and rule types allowed in the grammar use simple primitive distributions whose parameters can be easily estimated from observed draws. However, we note that the same general EM procedure can be applied to general rule types that do not admit closed-form parameter updates by replacing the closed-form M step with a gradient-based update, as in the variational inference procedure presented in [4]. This variational-EM extension could, for example, be useful for constructing richer grammars that use highly-parameterized and expressive distribution models (e.g. autoregressive flows [87]) to capture complex relative pose distributions between object types.

4.5.2 Comparison to sampling-based parameter estimation

Our parameter estimation approach fundamentally relies on scene parsing in its inner loop: in order to understand the relationship between the model parameters and observed objects, it uses parsing to reconstruct latent scene trees. Unfortunately, this parsing process can be expensive and difficult for complex grammars. Sampling-based distribution alignment methods (as used in e.g. MetaSim [18]) are appealing in that they completely bypass the parsing process. These techniques draw a population of samples from their model; compute a two-sample distribution alignment metric between the sampled population and a target population; and update model parameters accordingly (using e.g. a gradient update or REINFORCE).

A major barrier to applying these methods to updating scene grammar parameters is that it is fundamentally difficult to construct a two-sample test statistic that accurately measures the distribution distance between sampled *scenes*. [18] sidesteps this issue by performing comparison in an embedding space of images; but an equivalent comparison for comparing *populations of sets of objects* is not obvious. One possible

solution could be to use our object-level MMD comparison statistic as the alignment metric; however, pilot experiments indicate that this strategy may be too unstable and data-hungry for our small datasets. (A possible issue is that the object-level MMD metric is insufficiently sensitive, and does not capture the difference in distribution of *object count* in particular.) We have experimented with using a recurrent neural network to encode object *sets* into a fixed-size vector using a Seq2Seq architecture [88]; it seems plausible that a meaningful and sensitive two-sample statistic could be constructed in the latent space of such an autoencoder.

Chapter 5

Discussion and Future Work

In the previous chapters, we've detailed a specialized probabilistic procedural model – *spatial scene grammars* – that captures distributions over scenes of rigid objects of varying number and class. While we have contributed algorithms for doing various forms of inference in this model class, we've merely scratched the surface of interesting and exciting questions and applications related to this scene modeling problem. In this chapter, we discuss a handful of future research directions we've considered during the preparation of this thesis.

5.1 Closing the Vision Gap

A clear practical limitation of our scene grammar tool is that it describes a distribution over *sets of fully-observed objects*, while robots typically directly observe the world as an *image*. From a procedural modeling point of view, one often closes this gap by adding another stage to the model: after sampling a set of objects from the grammar, we pass that set of objects through a standard computer-graphics renderer to produce an image.

Including a renderer directly in the procedural model introduces an often complex and non-differentiable model stage that complicates inference – but despite this difficulty, this tactic is broadly popular. The vision-as-inverse-graphics approach is built on the idea of using a renderer to understand an image by finding an input

to the renderer (i.e. a scene graph with objects, lights, camera information, etc.) that reproduces it. These techniques typically use amortized proposal engines to produce suggestions of likely objects and other scene graph components, which are scored (and can be refined) by rendering the resulting scene graph to produce an image. This broad theme is reflected in a wide variety of approaches. The render-and-compare loss can be used to train reliable one-shot object pose estimators [89]; custom *differentiable* renderers can be used directly for iterative refinement of scene graph components [90, 91]; and one can even simultaneously learn a neural renderer [92]. Closely related to our work, MetaSim [18, 20] even accomplishes model parameter estimation on a grammar-like model from an image dataset by rendering draws from their object-production model and minimizing a two-sample distribution distance metric; they resort to numerical differentiation of the rendering process to make parameter updates.

A major difficulty with closing this gap for our scene grammar model is that the rendering process introduces multiple kinds of noise and uncertainty that our model must be augmented to capture. In any given image, objects may be occluded (i.e. present, for the purpose of the scene tree, but not observed), and their poses may be mis-estimated; this discrete and continuous uncertainty in the *observation* of each object's geometry is functionally an additional random choice that needs to be baked into the grammar model. Additionally, variations in object shape, texture, and lighting all impact the resulting image; a complete model for images might also need to capture these traits. In the rest of this section, we discuss possible ways of scoping solutions to some of these problems.

5.1.1 Consuming pose proposals

Perhaps the most direct and obvious approach to apply our grammar model parsing and parameter estimation strategies to images would be to first recover proposed object sets from each observed image using one of a variety of perception systems. A method to propose objects from images could either be used once, up-front, to transform a dataset of images into a dataset of object sets – which would allow direct

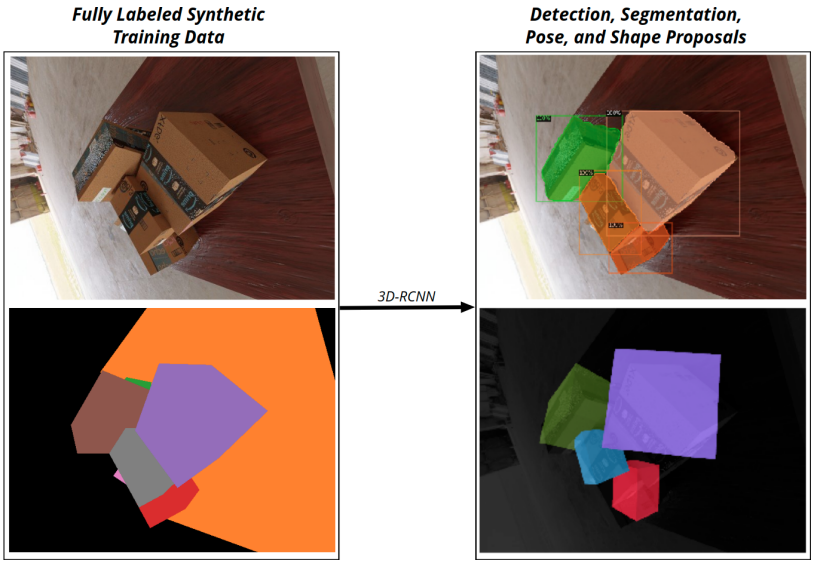


Figure 5-1: **Left**: Synthetic RGB[D] images with full segmentation and pose labels from a synthetic dataset of cardboard boxes in cluttered piles. **Right**: Using that dataset, we train a custom 3D-RCNN implementation to detect and segment these objects and regress their poses and shapes; the proposed box poses and shapes are illustrated in the bottom-right. (The color for each box instance in each subfigure are randomly chosen and do not correspond between images.) While detection and segmentation performance is excellent, rotation and shape estimation prove unreliable.

application of the tools in this thesis – or could be used as a proposal generator in a more involved iterative inference scheme to simultaneously search over object proposals and scene trees that are reasonable under the grammar *and* explain the observed scenes under a render-and-compare loss.

To begin probing what sorts of perception systems could suit this task, we experimented with applying 3D-RCNN pose and shape regression heads [89] (which we replicated by writing the appropriate regression heads on top of a Mask-RCNN backbone [3]) to detect the poses and shapes of cardboard shipping boxes from images of cluttered piles. We trained this detection network using fully-supervised images generated using our Blender-rendering pipeline (Section 2.5.3); we relied on direct supervision of the pose and shape regression from labels to train the network rather than using a render-and-compare loss. Unfortunately, while we were able to recover very accurate detection and segmentation masks, rotations and shape estimates were unstable and proved hard to train. (See Figure 5-1 for an example.) While there

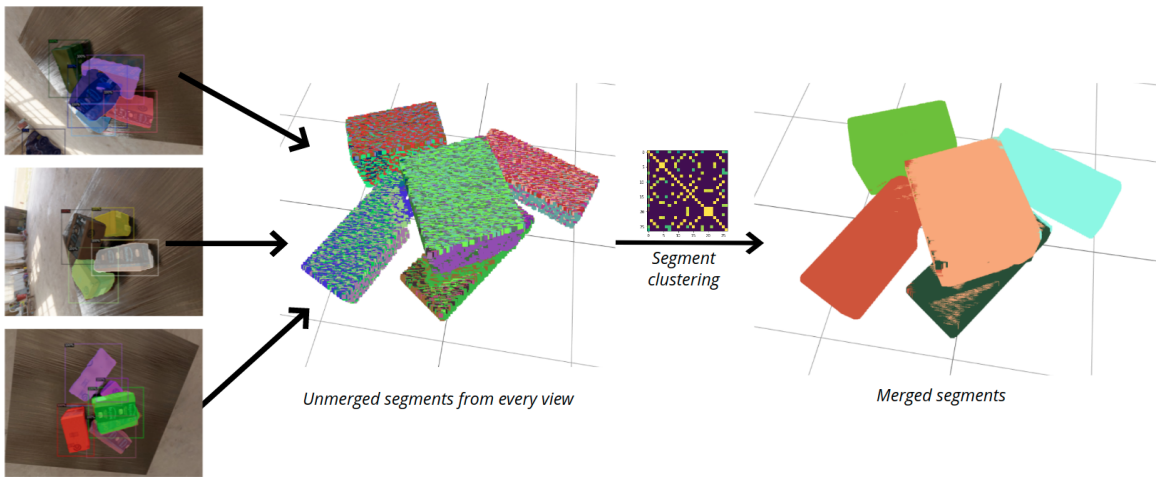


Figure 5-2: An illustration of a pipeline for merging independently segmented point clouds from multiple views into a combined, segmented point cloud. Each segment is indicated by points of a different color. Point clouds from multiple RGB[D] images are segmented using Mask-RCNN (left) and overlaid in world frame (middle). A graph is constructed over all segments across all independent views, with edge weights increasing with segment overlap (as defined by the total number of nearby points). Graph clusters correspond roughly to objects (right), providing multi-view, segmented point clouds that are typically detailed enough for deformable ICP to succeed.

are a number of things that could have gone wrong (including incorrect image normalization, insufficient data, bad hyperparameters, etc.), we suspect that the use of rotationally symmetric boxes as a target made regressing rotation (and thus shape) particularly difficult. We did not thoroughly explore alternative costs that were robust to rotation invariances; it's possible that the intermediate result we achieved here would be good enough to initialize the render-and-compare-based fine-tuning used in [89], which would itself be rotation-invariant.

As an alternative approach, we also experimented with using a variant of the iterative-closest-point object-in-point-cloud pose estimation algorithm [93] adapted to handle anisotropic scaling of objects in their body frame [94]. For each scene, we converted depth images from multiple perspectives[1] into a combined point cloud and performed rough segmentation based on the object detections from our trained Mask-RCNN backbone using a graph-clustering approach (see Figure 5-2). With these

[1]Using multiple perspectives is critical here: a single view will only observe one of the two sides of each box in each of its axis, making its size unobservable without considering scene context (which ICP does not use).

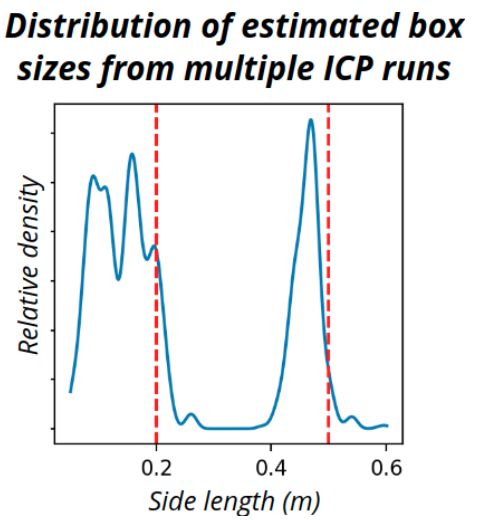


Figure 5-3: Given a single segmented point cloud from Fig 5-2 corresponding to a single box, we can randomly initialize many runs of deformable ICP to estimate possible poses and sizes of the box. The distribution over the resulting estimated side lengths for a box with actual side lengths [0.2, 0.2, 0.5] (marked with dashed red lines) is illustrated here. A perfect result would be sharp peaks at the location of the red lines. Due to occlusion, partial observation, and noise, our approach frequently underestimates sizes in each axis.

multiple views, we found that deformable ICP succeeded at recovering reasonable object poses and shapes – but unfortunately, it did not recover the shapes with sufficient accuracy or consistency to extract sensitive shape distributions (see Figure 5-3). We found that the the core difficulties are *observability* and *sensitivity*: since ICP ignores object context, it needs positive point returns on all sides of an object to be sure of its shape. Because all objects are on a tabletop, this will never occur. In combination with noisy estimation of the other shape dimensions of each object (due to ICP getting trapped in local minima induced by poor segmentations in these very cluttered scenes), the distribution of detected shapes was noisy. While this technique would likely be sufficient for detecting objects and their coarse poses, it did not show enough precision to also accurately recover distributions over the *shapes* of those objects.

These two pilot experiments (and their mixed results!) are reminders of the fundamental difficulty of perception – it is not surprising that solving these problems is the focus of an entire field! With time and effort, each of these approaches – deep perception networks and classical geometric perception strategies – could likely be made to work for this problem. In particular, that both of these strategies might be noisy and inaccurate in difficult scenes may indicate that they're a good fit to be combined with an analysis-by-synthesis refinement stage, as in many vision-as-inverse-graphics approaches. In practice, this would mean that we might use these techniques to gen-

erate a large population of object proposals of varying quality; each proposal can be parsed and tweaked to be more likely under the grammar (to reject categorically unlikely object configurations), and simultaneously rendered and tweaked to be more likely under the renderer (to better align with the final observed images).

5.1.2 Alternate perceptual spaces

Real-world RGB images are fantastically difficult to reproduce owing to the complex interplay of light and color constituting the visual world. Rather than trying to directly reproduce realistic RGB images with a rendering stage in a procedural model, it's often beneficial to instead seek to capture a realistic distribution in some other simpler-to-describe (but still sufficiently rich) perceptual space. [26] asserts, for example, that using segmentation masks, binary contour images, or deep neural features as the model's output space is likely to be much more tractable for inference. These "alternative" renderings are likely to be much easier to mathematically describe, differentiate, and invert; have smaller sim-to-real gap; and can be designed to be trivially and deterministically recoverable from raw RGB images. Two such alternative perceptual spaces that have been illustrated to be particularly effective for accomplishing robotics tasks are *dense descriptors* [2] and *keypoints* [95].

Fully-supervised dense descriptors

The dense descriptor approach entails "recoloring" an input $\langle W \times H \times 3 \rangle$ RGB image to a $\langle W \times H \times D \rangle$ D-dimensional descriptor space that has a set of desired semantic and invariance properties. (See Figure 5-4.) As originally shown in [96], a neural network can be trained to produce such dense descriptor images from RGB images in a self-supervised manner by training it to produce descriptor images that consistently "color" different parts of objects even when they are viewed from different perspectives. Given a dataset with multiple views of a set of objects within a restricted class (e.g. various mugs), training on this relatively simple loss allows the recovery of a descriptor space in which objects are colored with an arbitrary *class-consistent*

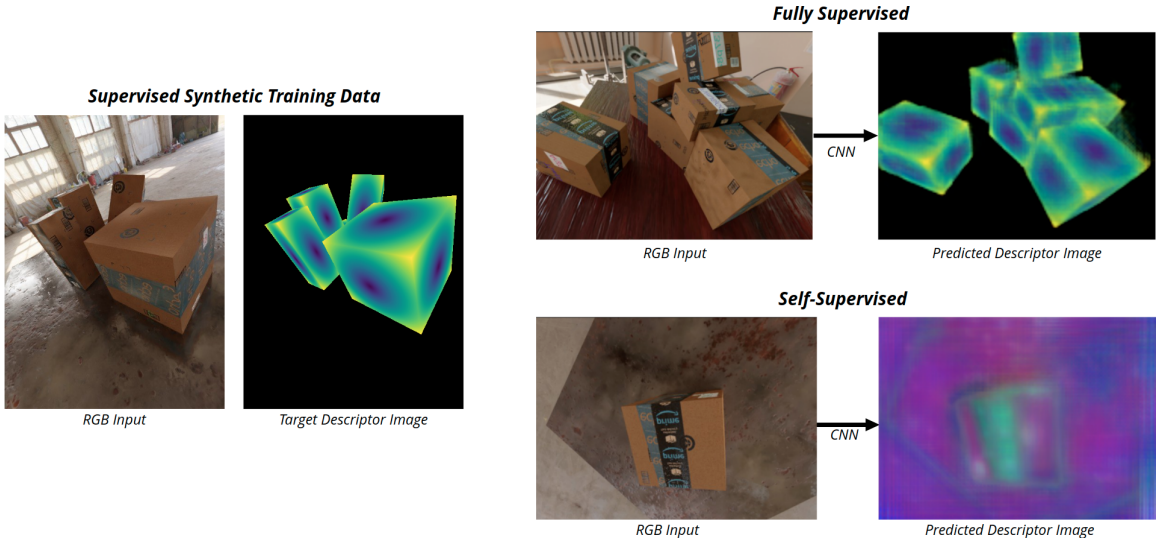


Figure 5-4: **Left**: We train a convolutional neural network (CNN) to predict dense descriptor images from RGB images by rendering paired RGB example inputs with hand-designed *target* dense descriptor images. **Top right**: After supervised training the CNN is able to consistently regress these dense descriptor images from unlabeled inputs. **Bottom right**: In contrast, applying the self-supervised descriptor learning technique of [2] leads to noisy and indistinct features for these highly-symmetric objects.

pattern that is lighting, geometry, texture, and view invariant [2, 96].

It's tempting, then, to use these learned descriptor spaces as our model's output space: in the case of a scene grammar, we might sample a set of objects; and then for each object, look up its desired or learned descriptor map and project that texture onto a camera image to create a rendering. Because this process sidesteps all lighting and texture details, this rendering operation becomes a simple project-with-occlusion operation; and it's likely that we can nearly perfectly reconstruct observed dense descriptor images with just this information. Beyond that, we can potentially use our model to *generate training data* for the dense descriptor prediction network. We illustrate results of a pilot experiment in this direction in Figure 5-4: we employ our rendering tools to product realistic RGB inputs alongside *target* dense descriptor images and train a convolutional neural network (CNN) following the architecture of [2] to regress those descriptor images.

In this case, we chose to rendering a target descriptor image directly, rather than rely on a self-supervised process to determine the optimal features as in [2, 96]. We

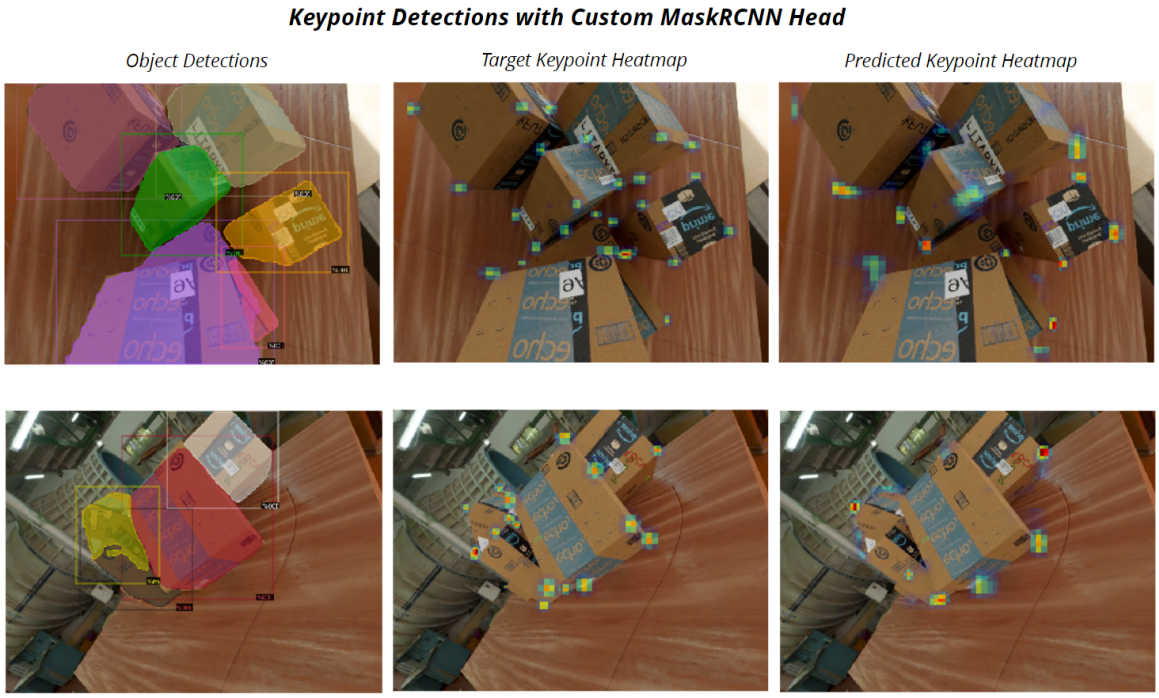


Figure 5-5: Results from a pilot experiment in which we add a custom keypoint heatmap-prediction head to a MaskRCNN [3] backbone. We find that MaskRCNN provides reasonable object detections and segmentations (**left**) and corner keypoint predictions (**right**) when trained on fully-supervised data. Each row is a different scene; target keypoint heatmaps are illustrated in the **middle** column for comparison.

attempted to find self-supervised features for these scenes with no success: we suspect that the multiple rotation and reflection symmetries of these objects made it difficult for the self-supervised process to find good descriptors, as the self-supervised process uses a contrastive loss enforcing that any given point on the object has the same feature from different perspectives, but is a different feature from any *other* point on the object. Because these objects are symmetric, the network would have to learn to break these symmetries to satisfy that loss. In the case of boxes (and, we suspect, many other objects), hand-designing task-relevant and symmetry-respecting dense descriptor spaces is easy, so we instead render pairs of RGB and target descriptor images and train the CNN in a fully-supervised fashion.[2]

[2]In our case, we use a simple fragment shader to render descriptor images where the color of a pixel corresponds to the distance the nearest corner of the box. See [97] for a recent and more principled approach to generating supervised descriptor images.

Grammars over 3D keypoints

While the rendering model for dense descriptors is vastly simpler than for full RGB images, dense descriptor images are still subject to occlusion and the nuanced shape variation of each object. An even more compact perceptual representation is to extract a small set of task-relevant *3D keypoints* from each object [95]. Each object is assigned a known constellation of keypoints that capture the task-relevant parts of its geometry – a mug, for example, might have keypoints at its bottom center, top center, and the center of its handle. All mugs have these keypoints, but for any given mug, the precise body-frame pose of those keypoints might vary in ways reflecting shape variation between mugs. [95] argues that keypoints are easy to design to be *minimal and sufficient* for any given task; that they generalize naturally across classes of objects with significant shape variation; and that they simple to work because they are geometric in nature.

As in Section 5.1.2, we can use our data generation to provide training data for a keypoint detection system. Results from a pilot experiment demonstrating this pipeline are illustrated in Figure 5-5. In this experiment, we intend to detect keypoints for boxes with significant symmetry, which is beyond the scope of the keypoint detection system used in [95].[3] Instead, we add a convolutional keypoint-heatmap prediction head on top of a Mask-RCNN backbone that produces a multimodal heatmap image of possible keypoint locations in pixel coordinates. For further investigation and details building on this approach, see [98]. This work not only improves on this keypoint detection pipeline, but also uses the output to predict object poses, and puts the entire system under stringent test by performing end-to-end adversarial example search for input object configurations that produce bad final pose estimates.

Spatial scene grammars are particularly well-suited to describing the spatial ar-

[3]The "standard" keypoint formulation (which is largely motivated by human pose estimation) is to give each object a list of semantically unique keypoints – the mug bottom, the mug top, the handle center – and predict a *separate* heatmap over pixel locations and possible depths for each of those unique keypoints. This is very convenient, as it allows the keypoint location to be computed by taking an integral over these heatmaps, which in turn produces a dense and empirically stable training loss. This approach doesn't extend to objects with keypoints that are interchangeable – like the corners of a rotationally symmetric box.

rangement of 3D keypoints. One might imagine adding production rules to each object representing the production of each expected keypoint, where that rule captures the typical pose distribution of that keypoint in body frame. (For a mug, this variation can capture things like typical height variation or handle placement across mugs.) These keypoints are then directly observed in an image through a simple camera projection, and supervised training data for estimating keypoints from raw RGB images is readily available from our data generation pipeline. Among practical future directions, we find this one particularly natural and exciting.

5.2 Downstream Robotics Applications

Our ultimate motivation for developing these procedural modeling tools is to make our robots better. The notion of a *distribution over worlds* seems to appear in every corner of robotics; here, we remark on a few choice downstream applications for tools like ours.

5.2.1 Rich priors for perception

A common way to frame the problem of robot perception is that, given a model over worlds $p(x)$ and an observation o, we wish to find a parsimonious explanation for that observation

$$\hat{x} = \arg\max_x p(x|o).$$

Very often, this objective is rearranged using Bayes' rule into

$$\hat{x} = \arg\max_x p(o|x)p(x),$$

which intuitively says that the best explanation for the observation is one that balances *matching* the observation with the model $p(o|x)$ and finding a *reasonable* explanation in the first place by using a prior over worlds $p(x)$.

Many common perception methods gloss over this prior over worlds. The classic ICP objective [93], for example, purely seeks out object locations to match a point

cloud without considering of whether those object locations are reasonable. As a result, it's easy for ICP to produce solutions that are physically or semantically implausible – objects standing on-end, floating in the air, or embedded inside of other objects. In the same vein, these priors can, if sufficiently specific, vastly reduce the search space that must be considered: if a robot is looking for a plate in a point cloud scan of an entire kitchen, it should probably concentrate its search on the countertops and not bother evaluating object proposals on the floor (or high in the air).

Scene grammars offer a means of expressing this prior over worlds in a way that we've shown can capture fine details in the distributions of presence and poses of objects. As discussed in Section 5.1, there are plenty of scene grammar, scene graph, and other procedural modeling analysis-by-synthesis approaches in which the scene-explanation process can be viewed as solving this perception problem. [19], for example, performs object detection and pose estimation directly from images by doing inference over a (physical support) grammar-like model and appearance model. Indeed, some scene grammar work is explicitly motivated as trying to better capture and take advantage of this rich prior structure: [23], for example, demonstrates improved pedestrian detection performance by using a handful of grammars as priors over the spatial arrangement and attributes of each detected segment of a person. Using these tools – even just as sanity-checks run post-hoc after applying standard perception techniques – is not yet standard practice in robotics, but we believe they could have an important part to play in improving both the efficiency and reliability of perception systems.

5.2.2 Design and verification considering all possible worlds

Using tools from robust control, one can make powerful statements about the evolution and performance of complex robotic systems – including guaranteeing robust flight trajectories of UAVs [99] and finding control sequences through contact for manipulation [100]. However, the majority of these powerful techniques for planning, control synthesis, and verification have been developed for systems with fixed state dimension; it's not immediately obvious how to extend them to domains where the

number of states may itself vary.

These grammar models may be a good fit for reasoning about the various "modes" – in terms of state dimension – that a system of interest is in. For example, one might wish to investigate the robustness of a manipulation controller under all expected arrangements of objects on a tabletop. Our scene grammar structure (and, in particular, our exhaustive mixed-integer parsing strategy) offers a scaffolding for performing a search over all possible sets of objects and their arrangements. This mixed-integer (or, for simple grammars, purely enumerative) search seems ripe for combination with, in particular, convex and mixed-integer-convex optimization strategies from robust control and verification. For example, one could conceivably frame a problem of finding an optimal control strategy that can be shared across *all possible worlds* by computing Lyapunov certificates (a la [99]) for a policy shared *across all scene trees*, where our supertree structure provides a scaffolding for performing that exhaustive enumeration.

Additionally, these models might be used as both a structure and a regularizer for failure case mining and adversarial example search. It has been thoroughly demonstrated that many effective-looking systems can be alarmingly easily broken by reasonable-seeming inputs found through an adversarial optimization process[4], but depending on the search method and objective, these failure cases might be arbitrarily unusual environment configurations that are not actually informative to the system's designer. Extreme pixel-perturbation-based attacks, for example, may produce adversarial input images that could not be generated by any physically plausible scene. Performing counterexample search directly in the space of simulator state (e.g. object poses, as in [98]) can help prevent these departures from reality, and procedural models like scene grammars provide a parameterization of that open-world simulator state conducive to optimization. Probabilistic procedure models can also provide useful *regularization* by distinguishing between realistic and unrealistic scene arrangements – which could be used to formulate a search for the *most likely* (and

[4]See [101] for a review of adversarial attacks as relevant to ML; but these ideas extend to almost any system one might want to deploy (even motion planners! [102]).

hence most important) failure cases.

5.2.3 Performance estimation with tuned world models

In a similar vein, scene grammars (and the broader class of data-tunable procedural or generative models in which they reside) have a critical role to play in simulation-based Monte Carlo analysis of robot systems. The increasing availability of large-scale compute makes it particularly tempting to test the performance of systems through [massively] parallel simulation of randomly sampled realistic scenarios. This strategy can be adapted to perform fine-grained performance assessment, even in systems with tiny failure rates (by e.g. adaptive bridge sampling [39]). Critically, these analyses are meaningless if one can not measure how likely any given scenario is in the real world – a measurement these models are poised to provide. In many cases, these models could potentially be slotted directly into Monte Carlo test engines as environment generators or world models.

A critical related question is *how important* an accurate world model is in any particular case. While it's convincing that getting accurate *dynamic* models is critical for sim-to-real transfer of dynamic behaviors [34], when is it important to capture realistic distributions over the precise occurrence rates and placements of objects in the world – and when can one get away with naive, simpler models? MetaSim [18] includes some preliminary experimental results indicating that deep neural vision systems for self-driving cars are more accurate on real data when trained on more realistically distributed simulated worlds, indicating that these highly-parameterized, data-hungry vision models may indeed be able to take advantage of more specific and realistic data distributions. Likewise, in situations involving robust control design, more specific world models make the design problem vastly easier.[5] However, naive background replacement – which functions on the hope that training on a data distribution that is intentionally *wider* than reality will lead to good performance in

[5]Consider designing a car controller to be robust to adversarial behavior of other drivers. If the behavior of other drivers is unconstrained, they may always choose to ram into the controlled vehicle, making the control design task impossible. It's necessary, in this case, to restrict the behavior of other drivers to follow a more specific and realistic world model.

reality – remains a popular and effective domain randomization strategy in computer vision. Indeed, experiments on DOPE [32] indicate that it performs best when it is trained on a *combination* of realistically-distribution and naively-domain-randomized training data, and that removing either decreases performance. We suspect that there is a need for principled and careful experiments that probe this question on robotics-relevant tasks. Can we use accurate world models, like our grammar model, to improve our ability to predict real-world performance from simulation, or improve sim-to-real transfer of learned behaviors?

5.3 Grammars vs General Procedural Models

The spatial scene grammars that we have formulated in this thesis are the tip of an iceberg of possible procedural models over worlds. While we have expanded on a number of appealing properties of scene grammars, there are plenty of environments for which scene grammars in particular are not a good fit. However, we hope that some of the core lessons and beliefs in this thesis can generalize to more flexible procedural models appropriate for these more complex cases.

Consider the problem of modeling the arrangement of densely packed boxes on a pallet. While one might imagine a grammar to describe this scene (perhaps focusing on capturing stacks and support structure as in [19], and then using constraints to implement non-penetration between boxes), it's apparent that each box on a pallet has critical and tight spatial relationships with *all* of its vertical and horizontal neighbors. These relationships form a locally-dense graph over all of the boxes in the scene – a far cry from the tree relational structure grammars are best suited to capture.

A tempting alternative procedural model for this scene would instead be to model the placement of each box in sequence *conditioned on the placement of all previous boxes.* Such a model would describe how the scene would be built up object-by-object.[6] For physical scenes like cluttered arrangements of objects, this approach has

[6]This "autoregressive" factorization – in which each generation is conditioned on all of the ones before it – offers an appealing completeness, in the sense that it can represent any distribution over objects. It is commonly applied to describing furniture arrangement in rooms that add one piece of

the additional appealing property that generated scenes can be made to *automatically* follow physical feasibility and stability constraints by enforcing that that each object placement is performed in a physically plausible way (by e.g. using a simulator).

We note that the core concepts of constrained sampling, parsing, and parameter estimation map to this model like they do to ours. In particular, the *parsing* problem now entails guessing in what order objects were placed to generate the scene; and given a solution to that parsing problem, one might imagine tackling *parameter estimation* of this more complex model with a similar EM approach to Chapter 4. This strategy is functionally similar to systems like Attend-Infer-Repeat [28] that tune autoregressive models to data using techniques like amortized variational inference.

Any procedural model over these sorts of open-world environments will have to contend with the mixed-discrete-and-continuous randomness that is fundamental to class of distributions. We believe that it is beneficial to attack that problem head-on by carefully designing procedural models that separate out the discrete and continuous decisions and enable principled scene generation and understanding by performing principled inference over possible worlds. Whether the underlying procedural model is grammar or graph structured (or something else entirely), we hope the algorithms and ideas in this thesis inspire new approaches to make our robots better prepared for the worlds they face.

furniture after the other until the room is complete [29, 103]; the same idea can be generalized all the way up to sequential generation of images by drawing patch after patch [28, 104].

Bibliography

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[2] Peter R Florence, Lucas Manuelli, and Russ Tedrake. Dense object nets: Learning dense visual object descriptors by and for robotic manipulation. *arXiv preprint arXiv:1806.08756*, 2018.

[3] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[4] Gregory Izatt and Russ Tedrake. Generative Modeling of Environments with Scene Grammars and Variational Inference. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6891–6897, Paris, France, May 2020. IEEE.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*, May 2019. arXiv: 1810.04805.

[6] Tero Karras, Samuli Laine, and Timo Aila. A Style-Based Generator Architecture for Generative Adversarial Networks. *arXiv:1812.04948 [cs, stat]*, March 2019. arXiv: 1812.04948.

[7] Andrew Brock, Theodore Lim, J. M. Ritchie, and Nick Weston. Generative and Discriminative Voxel Modeling with Convolutional Neural Networks. *arXiv:1608.04236 [cs, stat]*, August 2016. arXiv: 1608.04236.

[8] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning Representations and Generative Models for 3D Point Clouds. *arXiv:1707.02392 [cs]*, June 2018. arXiv: 1707.02392.

[9] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. *arXiv:2003.08934 [cs]*, August 2020. arXiv: 2003.08934.

[10] Adam R. Kosiorek, Heiko Strathmann, Daniel Zoran, Pol Moreno, Rosalia Schneider, Soňa Mokrá, and Danilo J. Rezende. NeRF-VAE: A Geometry Aware 3D Scene Generative Model. *arXiv:2104.00587 [cs, stat]*, April 2021. arXiv: 2104.00587.

[11] Wei Gao. *Representing Unstructured Environments for Robotic Manipulation: Toward Generalization, Dexterity and Robustness*. PhD thesis, MIT, August 2021.

[12] Przemyslaw Prusinkiewicz, Mark Hammel, Jim Hanan, and Radomir Mech. L-systems: from the theory to visual models of plants. In *Proceedings of the 2nd CSIRO Symposium on Computational Challenges in Life Sciences*, volume 3, pages 1–32. Citeseer, 1996.

[13] George Stiny. Introduction to shape and shape grammars. *Environment and planning B: planning and design*, 7(3):343–351, 1980.

[14] Feng Han and Song-Chun Zhu. Bottom-Up/Top-Down Image Parsing with Attribute Grammar. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(1):59–73, January 2009.

[15] Yibiao Zhao and Song-chun Zhu. Image Parsing with Stochastic Scene Grammar. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 73–81. Curran Associates, Inc., 2011.

[16] Jeroen Chua. *Probabilistic Scene Grammars: A General-Purpose Framework For Scene Understanding*. PhD thesis, Brown University, May 2018.

[17] Tianqiang Liu, Siddhartha Chaudhuri, Vladimir G. Kim, Qixing Huang, Niloy J. Mitra, and Thomas Funkhouser. Creating consistent scene graphs using a probabilistic grammar. *ACM Transactions on Graphics*, 33(6):1–12, November 2014.

[18] Amlan Kar, Aayush Prakash, Ming-Yu Liu, Eric Cameracci, Justin Yuan, Matt Rusiniak, David Acuna, Antonio Torralba, and Sanja Fidler. Meta-Sim: Learning to Generate Synthetic Datasets. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4550–4559, Seoul, Korea (South), October 2019. IEEE.

[19] Ben Zinberg, Marco Cusumano-Towner, and K Mansinghka Vikash. Structured differentiable models of 3d scenes via generative scene graphs. In *Workshop on Perception as Generative Reasoning, NeurIPS, Submitted September*, 2019.

[20] Jeevan Devaranjan, Amlan Kar, and Sanja Fidler. Meta-sim2: Unsupervised learning of scene structure for synthetic data generation. In *European Conference on Computer Vision*, pages 715–733. Springer, 2020.

[21] Yi-Ting Yeh, Lingfeng Yang, Matthew Watson, Noah D Goodman, and Pat Hanrahan. Synthesizing open worlds with constraints using locally annealed reversible jump mcmc. *ACM Transactions on Graphics (TOG)*, 31(4):1–11, 2012.

[22] Matthew Fisher, Daniel Ritchie, Manolis Savva, Thomas Funkhouser, and Pat Hanrahan. Example-based synthesis of 3D object arrangements. *ACM Transactions on Graphics*, 31(6):1–11, November 2012.

[23] Seyoung Park, Bruce Xiaohan Nie, and Song-Chun Zhu. Attribute And-Or Grammar for Joint Parsing of Human Pose, Parts and Attributes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(7):1555–1569, July 2018.

[24] Antoni Rosinol, Andrew Violette, Marcus Abate, Nathan Hughes, Yun Chang, Jingnan Shi, Arjun Gupta, and Luca Carlone. Kimera: from slam to spatial perception with 3d dynamic scene graphs. *arXiv preprint arXiv:2101.06894*, 2021.

[25] Daniel Ritchie. *Probabilistic programming for procedural modeling and design.* PhD thesis, Stanford University, 2016.

[26] Tejas D Kulkarni, Pushmeet Kohli, Joshua B Tenenbaum, and Vikash Mansinghka. Picture: A probabilistic programming language for scene perception. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4390–4399, Boston, MA, USA, June 2015. IEEE.

[27] Vikash K Mansinghka, Tejas D Kulkarni, Yura N Perov, and Josh Tenenbaum. Approximate bayesian image interpretation using generative probabilistic graphics programs. *Advances in Neural Information Processing Systems*, 26:1520–1528, 2013.

[28] SM Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Geoffrey E Hinton, et al. Attend, infer, repeat: Fast scene understanding with generative models. *Advances in Neural Information Processing Systems*, 29:3225–3233, 2016.

[29] Daniel Ritchie, Kai Wang, and Yu-An Lin. Fast and Flexible Indoor Scene Synthesis via Deep Convolutional Generative Models. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6175–6183, Long Beach, CA, USA, June 2019. IEEE.

[30] Kai Wang, Manolis Savva, Angel X. Chang, and Daniel Ritchie. Deep convolutional priors for indoor scene synthesis. *ACM Transactions on Graphics*, 37(4):1–14, August 2018.

[31] Matthew Johnson-Roberson, Charles Barto, Rounak Mehta, Sharath Nittur Sridhar, Karl Rosaen, and Ram Vasudevan. Driving in the Matrix: Can Virtual Worlds Replace Human-Generated Annotations for Real World Tasks? *arXiv:1610.01983 [cs]*, February 2017. arXiv: 1610.01983.

[32] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects. *arXiv:1809.10790 [cs]*, September 2018. arXiv: 1809.10790.

[33] Bhairav Mehta, Ankur Handa, Dieter Fox, and Fabio Ramos. A User's Guide to Calibrating Robotics Simulators. *arXiv:2011.08985 [cs]*, November 2020. arXiv: 2011.08985.

[34] Fabio Ramos, Rafael Carvalhaes Possas, and Dieter Fox. BayesSim: adaptive domain randomization via probabilistic inference for robotics simulators. *arXiv:1906.01728 [cs]*, June 2019. arXiv: 1906.01728.

[35] Daniel J. Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Scenic: A Language for Scenario Specification and Scene Generation. *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 63–78, June 2019. arXiv: 1809.09310.

[36] Chengshu Li, Fei Xia, Roberto Martín-Martín, Michael Lingelbach, Sanjana Srivastava, Bokui Shen, Kent Vainio, Cem Gokmen, Gokul Dharan, Tanish Jain, Andrey Kurenkov, C. Karen Liu, Hyowon Gweon, Jiajun Wu, Li Fei-Fei, and Silvio Savarese. iGibson 2.0: Object-Centric Simulation for Robot Learning of Everyday Household Tasks. *arXiv:2108.03272 [cs]*, November 2021. arXiv: 2108.03272.

[37] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *International Conference on 3D Vision (3DV)*, 2017.

[38] Yizhou Zhao, Kaixiang Lin, Zhiwei Jia, Qiaozi Gao, Govind Thattai, Jesse Thomason, and Gaurav S Sukhatme. Luminous: Indoor scene generation for embodied ai challenges. *arXiv preprint arXiv:2111.05527*, 2021.

[39] Aman Sinha, Matthew O'Kelly, Russ Tedrake, and John Duchi. Neural Bridge Sampling for Evaluating Safety-Critical Autonomous Systems. *arXiv:2008.10581 [cs, stat]*, August 2021. arXiv: 2008.10581.

[40] Charles Richter and Nicholas Roy. Safe Visual Navigation via Deep Learning and Novelty Detection. In *Robotics: Science and Systems XIII*. Robotics: Science and Systems Foundation, July 2017.

[41] Chris Paxton, Chris Xie, Tucker Hermans, and Dieter Fox. Predicting Stable Configurations for Semantic Placement of Novel Objects. *arXiv:2108.12062 [cs]*, August 2021. arXiv: 2108.12062.

[42] George Stiny and James Gips. Shape grammars and the generative specification of painting and sculpture. In *IFIP congress (2)*, volume 2(3), pages 125–135, 1971.

[43] Carlos A. Vanegas, Daniel G. Aliaga, and Bedrich Benes. Building reconstruction using manhattan-world grammars. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 358–365, San Francisco, CA, USA, June 2010. IEEE.

[44] Peter Wonka, Michael Wimmer, François Sillion, and William Ribarsky. Instant architecture. *ACM Transactions on Graphics (TOG)*, 22(3):669–677, 2003.

[45] Jerry O Talton, Yu Lou, Steve Lesser, Jared Duke, Radomír Měch, and Vladlen Koltun. Metropolis procedural modeling. *ACM Transactions on Graphics (TOG)*, 30(2):1–14, 2011.

[46] Pedro F. Felzenszwalb. A Stochastic Grammar for Natural Shapes. In Sven J. Dickinson and Zygmunt Pizlo, editors, *Shape Perception in Human and Computer Vision*, pages 299–310. Springer London, London, 2013.

[47] Joseph Schlecht, Kobus Barnard, Ekaterina Spriggs, and Barry Pryor. Inferring Grammar-based Structure Models from 3D Microscopy Data. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, Minneapolis, MN, USA, June 2007. IEEE.

[48] Ondrej Šťava, Bedrich Beneš, Radomir Měch, Daniel G Aliaga, and Peter Krištof. Inverse procedural modeling by automatic generation of l-systems. In *Computer Graphics Forum*, volume 29 (2), pages 665–674. Wiley Online Library, 2010.

[49] Jerry Talton, Lingfeng Yang, Ranjitha Kumar, Maxine Lim, Noah Goodman, and Radomír Měch. Learning design patterns with bayesian grammar induction. In *Proceedings of the 25th annual ACM symposium on User interface software and technology - UIST '12*, page 63, Cambridge, Massachusetts, USA, 2012. ACM Press.

[50] Song-Chun Zhu and David Mumford. A Stochastic Grammar of Images. *Foundations and Trends® in Computer Graphics and Vision*, 2(4):259–362, 2006.

[51] Yibiao Zhao and Song-Chun Zhu. Scene Parsing by Integrating Function, Geometry and Appearance Models. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3119–3126, Portland, OR, USA, June 2013. IEEE.

[52] Jared Marshall Glover. *The Quaternion Bingham Distribution, 3D Object Detection, and Dynamic Manipulation.* PhD thesis, Massachusetts Institute of Technology, 2014.

[53] Christopher Bingham. An Antipodally Symmetric Distribution on the Sphere. *The Annals of Statistics*, 2(6):1201–1225, November 1974. Publisher: Institute of Mathematical Statistics.

[54] Jared Glover and Leslie Pack Kaelbling. Tracking 3-d rotations with the quaternion bingham filter. Technical report, MIT, 2013.

[55] Daniel Ritchie, Sharon Lin, Noah D. Goodman, and Pat Hanrahan. Generating Design Suggestions under Tight Constraints with Gradient-based Probabilistic Programming. *Computer Graphics Forum*, 34(2):515–526, May 2015.

[56] Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics letters B*, 195(2):216–222, 1987.

[57] Matthew D Hoffman, Andrew Gelman, et al. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623, 2014.

[58] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul A. Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep universal probabilistic programming. *J. Mach. Learn. Res.*, 20:28:1–28:6, 2019.

[59] Russ Tedrake and the Drake Development Team. Drake: Model-based design and verification for robotics, 2019. Available at `drake.mit.edu`.

[60] Philip E. Gill, Walter Murray, Michael A. Saunders, and Elizabeth Wong. User's guide for SNOPT 7.7: Software for large-scale nonlinear programming. Center for Computational Mathematics Report CCoM 18-1, Department of Mathematics, University of California, San Diego, La Jolla, CA, 2018.

[61] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[62] Robin Deits. Meshcat, 2022. Available at `https://github.com/rdeits/meshcat-python`.

[63] Open Robotics and Contributors. Ignition. Models from contributors at `app.ignitionrobotics.org/fuel/models`.

[64] Turbosquid 3d models. Models with royalty-free licenses collected from `www.turbosquid.com`.

[65] Dawson-Haggerty et al. Trimesh. Available at `www.trimsh.org`.

[66] Khaled Mammou. V-hacd. `https://github.com/kmammou/v-hacd`, 2020.

[67] Open Source Robotics Foundation. Simulation description format. Details at `sdformat.org`.

[68] Michael Gschwandtner, Roland Kwitt, Andreas Uhl, and Wolfgang Pree. Blensor: Blender sensor simulation toolbox. In *International Symposium on Visual Computing*, pages 199–208. Springer, 2011.

[69] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam, 2022. `http://www.blender.org`.

[70] Russ Tedrake. TRI Taking on the Hard Problems in Manipulation Research Toward Making Human-Assist Robots Reliable and Robust. Press release, published 2019-06-27 at `www.tri.global/news/tri-taking-on-the-hard-problems-in-manipulation-re-2019-6-27/`.

[71] Razer hydra. Information at `http://support.razer.com/console/razer-hydra/`.

[72] Daniel Ritchie, Ben Mildenhall, Noah D. Goodman, and Pat Hanrahan. Controlling procedural modeling programs with stochastically-ordered sequential Monte Carlo. *ACM Transactions on Graphics*, 34(4):1–11, July 2015.

[73] Jake Porway, Qiongchen Wang, and Song Chun Zhu. A hierarchical and contextual model for aerial image parsing. *International journal of computer vision*, 88(2):254–283, 2010.

[74] George Matheos, Alexander K Lew, Matin Ghavamizadeh, Stuart Russell, Marco Cusumano-Towner, and Vikash Mansinghka. Transforming worlds: Automated involutive mcmc for open-universe probabilistic models. In *Third Symposium on Advances in Approximate Bayesian Inference*, 2020.

[75] Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. Learning to Infer Graphics Programs from Hand-Drawn Images. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6059–6068. Curran Associates, Inc., 2018.

[76] Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. InverseCSG: automatic conversion of 3D models to CSG trees. *ACM Transactions on Graphics*, 37(6):1–16, January 2019.

[77] Hongkai Dai, Gregory Izatt, and Russ Tedrake. Global inverse kinematics via mixed-integer convex optimization. *The International Journal of Robotics Research*, 38(12-13):1420–1441, 2019.

[78] Garth P McCormick. Computability of global solutions to factorable nonconvex programs: Part i—convex underestimating problems. *Mathematical programming*, 10(1):147–175, 1976.

[79] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2021.

[80] James Saunderson, Pablo A Parrilo, and Alan S Willsky. Semidefinite descriptions of the convex hull of rotation matrices. *SIAM Journal on Optimization*, 25(3):1314–1343, 2015.

[81] Heng Yang, Jingnan Shi, and Luca Carlone. Teaser: Fast and certifiable point cloud registration. *IEEE Transactions on Robotics*, 37(2):314–333, 2020.

[82] Chenfanfu Jiang, Siyuan Qi, Yixin Zhu, Siyuan Huang, Jenny Lin, Lap-Fai Yu, Demetri Terzopoulos, and Song-Chun Zhu. Configurable 3D Scene Synthesis and 2D Image Rendering with Per-pixel Ground Truth Using Stochastic Grammars. *International Journal of Computer Vision*, 126(9):920–941, September 2018.

[83] 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):1–22, September 1977.

[84] Igor Gilitschenski, Roshni Sahoo, Wilko Schwarting, Alexander Amini, Sertac Karaman, and Daniela Rus. Deep orientation uncertainty learning based on a bingham loss. In *International Conference on Learning Representations*, 2020.

[85] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

[86] Nachman Aronszajn. Theory of reproducing kernels. *Transactions of the American mathematical society*, 68(3):337–404, 1950.

[87] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. *Advances in neural information processing systems*, 30, 2017.

[88] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[89] Abhijit Kundu, Yin Li, and James M Rehg. 3d-rcnn: Instance-level 3d object reconstruction via render-and-compare. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3559–3568, 2018.

[90] Lukasz Romaszko, Christopher KI Williams, Pol Moreno, and Pushmeet Kohli. Vision-as-inverse-graphics: Obtaining a rich 3d explanation of a scene from a single image. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 851–859, 2017.

[91] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Transactions on Graphics (TOG)*, 37(6):1–11, 2018.

[92] Tejas D Kulkarni, William F Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. *Advances in neural information processing systems*, 28, 2015.

[93] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor fusion IV: control paradigms and data structures*, volume 1611, pages 586–606. Spie, 1992.

[94] Richard Everson. Orthogonal, but not orthonormal, procrustes problems. *Advances in computational Mathematics*, 3(4), 1998.

[95] Lucas Manuelli, Wei Gao, Peter Florence, and Russ Tedrake. kpam: Keypoint affordances for category-level robotic manipulation. *arXiv preprint arXiv:1903.06684*, 2019.

[96] Tanner Schmidt, Richard Newcombe, and Dieter Fox. Self-supervised visual descriptor learning for dense correspondence. *IEEE Robotics and Automation Letters*, 2(2):420–427, 2016.

[97] Andras Kupcsik, Markus Spies, Alexander Klein, Marco Todescato, Nicolai Waniek, Philipp Schillinger, and Mathias Bürger. Supervised training of dense object nets using optimal descriptors for industrial robotic applications. *arXiv preprint arXiv:2102.08096*, 2021.

[98] Maggie Wang. Robust pose estimation from 3d keypoints, May 2021. Available at `https://maggiewang.org/assets/pdfs/robust_pose_estimation_from_3d_keypoints.pdf`.

[99] Anirudha Majumdar and Russ Tedrake. Funnel libraries for real-time robust feedback motion planning. *The International Journal of Robotics Research*, 36(8):947–982, 2017.

[100] Albert Wu, Sadra Sadraddini, and Russ Tedrake. R3t: Rapidly-exploring random reachable set tree for optimal kinodynamic planning of nonlinear hybrid systems. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4245–4251. IEEE, 2020.

[101] Shilin Qiu, Qihe Liu, Shijie Zhou, and Chunjiang Wu. Review of artificial intelligence adversarial attack and defense technologies. *Applied Sciences*, 9(5):909, 2019.

[102] Sai Vemprala and Ashish Kapoor. Adversarial attacks on optimization based planners. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9943–9949. IEEE, 2021.

[103] Despoina Paschalidou, Amlan Kar, Maria Shugrina, Karsten Kreis, Andreas Geiger, and Sanja Fidler. ATISS: Autoregressive transformers for indoor scene synthesis. *Advances in Neural Information Processing Systems*, 34, 2021.

[104] Martin Engelcke, Adam R Kosiorek, Oiwi Parker Jones, and Ingmar Posner. Genesis: Generative scene inference and sampling with object-centric latent representations. *arXiv preprint arXiv:1907.13052*, 2019.