Probabilistic Geometric Grammars for Object Recognition

by

Margaret Aida Aycinena

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

Master of Science in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2005

Author . . . .
Department of Electrical Engineering and Computer Science
August 1, 2005

Certified by . . . .
Leslie Pack Kaelbling
Professor
Thesis Supervisor

Certified by . . . .
Tomás Lozano-Pérez
Professor
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Probabilistic Geometric Grammars for Object Recognition

by

Margaret Aida Aycinena

Submitted to the Department of Electrical Engineering and Computer Science on August 1, 2005, in partial fulfillment of the requirements for the degree of Master of Science in Computer Science

Abstract

This thesis presents a generative three-dimensional (3D) representation and recognition framework for classes of objects. The framework uses probabilistic grammars to represent object classes recursively in terms of their parts, thereby exploiting the hierarchical and substitutive structure inherent to many types of objects. The framework models the 3D geometric characteristics of object parts using multivariate conditional Gaussians over dimensions, position, and rotation. I present algorithms for learning geometric models and rule probabilities given parsed 3D examples and a fixed grammar. I also present a parsing algorithm for classifying unlabeled, unparsed 3D examples given a geometric grammar. Finally, I describe the results of a set of experiments designed to investigate the chosen model representation of the framework.

Thesis Supervisor: Leslie Pack Kaelbling
Title: Professor

Thesis Supervisor: Tomás Lozano-Pérez
Title: Professor

Acknowledgments

I have been extremely fortunate in the support I have received in this research. My first acknowledgements go to my advisors, Leslie Kaelbling and Tomás Lozano-Pérez. The work presented in this thesis is directly based on their previous work on learning three-dimensional models for objects, so they deserve much of the credit for the content. Furthermore, Leslie and Tomás have been wonderful advisors – they have shown great patience as I explored new topics, offered helpful ideas and encouragement when I was stuck, even checked my math.

Second, I owe a huge debt of gratitude to my lab and office mates: Michael Ross, Sam Davies, Han-Pang Chiu, Luke Zettlemoyer, Sarah Finney, Natalia Hernandez-Gardiol, Kurt Steinkraus, Nick Matsakis, and James McLurkin. They have provided enlightenment on all aspects of machine learning, probability theory, computer vision, natural language processing and linguistics, experimental methods, and research in general, as well as political food-for-thought and afternoon frozen yogurt runs.

Third, I am deeply grateful to my family and friends. My parents, Peggy and Alex, have not only supported and encouraged me in everything I have done, but proofread and edited every page of this document (any remaining errors are my own). My sister and brother, Diana and Alex, have been incredibly supportive, were always willing to listen, and made well-timed phone calls for maximum middle-of-the-night encouragement. Finally, my boyfriend Shaun has been an invaluable sounding board and source of strength, for ideas, successes, and frustrations, in research and in life.

Contents

1 Introduction **15**

1.1 Objectives . . . 16

1.2 Grammars for Language and Objects . . . 17

1.3 Motivation . . . 19

1.3.1 Three-Dimensional Models . . . 19

1.3.2 A Parts-Based Approach . . . 22

1.3.3 Capturing Structural Variability With Grammars . . . 22

1.3.4 A Final Motivation . . . 23

1.4 Related Work . . . 23

1.4.1 Approaches to Object Recognition . . . 23

1.4.2 Cognitive Science Perspectives . . . 26

1.4.3 Context-Free Grammars . . . 26

1.5 Strengths and Weaknesses . . . 27

2 The Probabilistic Geometric Grammar Framework **29**

2.1 An Introduction to PGGs . . . 30

2.2 Incorporating Geometric Information . . . 32

2.2.1 Variables and Constant Symbols . . . 32

2.2.2 Issues With Geometric Grammars . . . 32

2.2.3 Representing Object Parts as 3D Boxes . . . 33

2.2.4 Types of Geometric Models . . . 34

2.3 Root Geometric Models . . . 35

2.3.1 Multivariate Gaussians Over Quaternions . . . 35

2.3.2 Multivariate Gaussians Over Object Parts . . . 36

2.4 Part Geometric Models . . . 37

2.4.1 Conditional Multivariate Gaussians Over Magnitudes . . . 37

2.4.2 Conditional Multivariate Gaussians Over Quaternions . . . 38

2.4.3 Conditional Multivariate Gaussians Over Object Parts . . . 41

2.5 Conditional Independence Assumptions . . . 43

2.6 The Likelihood of a Parsed Instance . . . 44

2.6.1 PGG Parse Trees . . . 44

2.6.2 Calculating the Likelihood . . . 45

3 Learning in the PGG Framework **49**
3.1 Matching Tree Fragments to Rules 49
3.1.1 Tree Fragments 50
3.1.2 Matching To Rules 50
3.2 Learning Expansion Probabilities on Rules 51
3.3 Learning Geometric Models 51
3.3.1 Estimating Multivariate Gaussians over Quaternions 52
3.3.2 Estimating Multivariate Gaussians over Object Parts 53
3.3.3 Estimating Conditional Multivariate Gaussians over Object Parts 55
3.3.4 An Algorithm For Learning Geometric Models From Examples 56
3.4 Training Data 58

4 Parsing in the PGG Framework **59**
4.1 Geometric Parsing 59
4.1.1 The Unordered Nature of 3D Space 59
4.1.2 Chomsky Normal Form for PGGs 60
4.1.3 Subtree Notation 61
4.1.4 The Bounding Box: An Approximate Maximum Likelihood . . 62
4.1.5 A Penalty For Clutter Parts 64
4.2 The Inside Algorithm for PGGs 65
4.2.1 Calculating the Likelihood of a Set of Object Parts 65
4.2.2 Finding the Most Likely Parse of a Set of Object Parts 69
4.2.3 Log Likelihoods 71
4.3 Complexity Concerns 71

5 Experimental Results **73**
5.1 Hypothesis and Approach 73
5.2 PGG Implementation 75
5.3 Experimental Setup 75
5.3.1 Baseline Models 75
5.3.2 Object Classes 76
5.3.3 Synthetic Data 77
5.3.4 Training and Testing Procedure 77
5.4 Results and Discussion 78

6 Conclusion **87**
6.1 Future Work 87
6.1.1 Recognition 87
6.1.2 Representation 89
6.1.3 Learning 89
6.2 In Conclusion 90

A Background **91**
A.1 Probabilistic Context-Free Grammars 91
A.1.1 An Introduction to PCFGs for Language 92
A.1.2 Parsing in PCFGs: The Inside Algorithm 94
A.1.3 Sources and Related Work 97
A.2 Representing Rotation with Quaternions 97
A.2.1 Representation Choices for Rotation 97
A.2.2 An Introduction to Quaternions 100
A.2.3 Gaussian Distributions Over Unit Quaternions 104
A.2.4 Parameter Estimation in Gaussians Over Unit Quaternions . . 106
A.2.5 Sources and Related Work 109
A.3 Conditional Multivariate Gaussian Distributions 110
A.3.1 Partitioned Matrices 110
A.3.2 Marginalizing and Conditioning 111
A.3.3 Sources and Related Work 112
A.4 Estimating The Minimum Volume Bounding Box 113
A.4.1 An Approximation Algorithm 113
A.4.2 Sources and Implementation 113

B Experiment Models **115**

List of Figures

1-1 The distinction between the gross shape and structure of an object versus detailed shape and texture information. This thesis focuses on capturing shape information at the level of the left image, rather than the right. [22] 16

1-2 A context-free grammar for a tiny subset of English noun phrases. . . 17

1-3 Parse trees for the English noun phrases *the big red barn* and *the big red ball*. 17

1-4 A simple context-free grammar for chairs. 18

1-5 The same object can appear drastically different when observed from different viewpoints. [8] 19

1-6 Man-made objects exhibit a large amount of structural variability [6]. 20

1-7 The distinction between objects with the same class but different shapes, and objects with the same shape but different appearances. [22] . . . 21

1-8 A simple probabilistic context-free grammar for chairs. 23

2-1 A PGG for chairs. 30

2-2 A PGG parse tree for a chair with four legs and no arms. 31

2-3 Assumed indexing of nodes for the derivation of the likelihood of a parse tree. 46

4-1 If $B = \{\mathbf{b}_{g_1}, \ldots, \mathbf{b}_{g_n}\}$, then $\square(B)$ is defined to be the minimum volume bounding box of the parts at the leaves of the tree, $\text{leaves}(t) = \{\mathbf{b}_{h_1}, \ldots, \mathbf{b}_{h_m}\}$, rather than the actual parts in B. 63

4-2 The growth of the Stirling numbers of the second kind $S(m, n)$. . . . 71

5-1 Performance of the PGG framework and the baseline models on all 12 ground classes, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance. 78

5-2 Performance of the PGG framework and the baseline models on the chair-with-legs ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance. 79

5-3 Performance of the PGG framework and the baseline models on the chair-with-legs-and-arms ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to show asymptotic performance. 80
5-4 Performance of the PGG framework and the baseline models on the chair-with-3-wheels ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance. 81
5-5 Performance of the PGG framework and the baseline models on the chair-with-3-wheels-and-arms ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance. 82
5-6 Performance of the PGG framework and the baseline models on the chair-with-5-wheels ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance. 82
5-7 Performance of the PGG framework and the baseline models on the chair-with-5-wheels-and-arms ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance. 83
5-8 Performance of the PGG framework and the baseline models on the bench ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance. 83
5-9 Performance of the PGG framework and the baseline models on the bench-with-arms ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance. 84
5-10 Performance of the PGG framework and the baseline models on the stool ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance. 84
5-11 Performance of the PGG framework and the baseline models on the table ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance. 85
5-12 Performance of the PGG framework and the baseline models on the coffee-table ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance. 85
5-13 Performance of the PGG framework and the baseline models on the lamp ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance. 86

A-1 A probabilistic context-free grammar for a tiny subset of English noun phrases. . . . 92
A-2 Probabilistic parse trees for the English noun phrases *the big red barn* and *the big red ball*. . . . 93
A-3 A depiction of an inductive step in the calculation of the inside probability of a substring w_{pq}. . . . 95
A-4 In a Gaussian distribution over quaternions, the distribution is defined in the three-dimensional tangent space to the four-dimensional unit hypersphere at the mean, and then the tails of the distribution are "wrapped" back onto the hypersphere to produce a spherical distribution. Here, for illustrative purposes, the 4D hypersphere is depicted as a 2D circle and the 3D tangent space as a 1D tangent line. We also ignore the other peak of the bimodal distribution. [22] 104

B-1 The PGG used in the experiments (continued in next figure). The learned expansion probabilities are shown. The presence of a learned root geometric model is denoted with a #, and the presence of a learned part geometric model with a %. . . . 115
B-2 The PGG used in the experiments (continued). . . . 116
B-3 The fully connected models used in the experiments. The learned prior structural probabilities over models are shown. The presence of a learned geometric model is denoted with a #. . . . 117
B-4 The Bayes net models used in the experiments (continued in next figure). The learned prior structural probabilities over models are shown. The presence of a learned root geometric model is denoted with a #, and the presence of a learned part geometric model with a %. . . . 118
B-5 The Bayes net models used in the experiments (continued). . . . 119

Chapter 1

Introduction

In this thesis, we present a generative parts-based three-dimensional (3D) representation and recognition framework for classes of objects.

By *generative*, we mean that the framework explicitly models a sufficient number of properties of each object class that new members of an object class can be "generated" given a model.[1] The term *parts-based* means that classes of objects are represented in terms of their parts, and *three-dimensional* means that the geometric characteristics of object classes are modeled in three dimensions, independent of viewpoint.

This thesis is organized as follows:

Chapter 1 introduces and motivates the approach of this thesis, as well as discusses previous related work.

Chapter 2 describes the probabilistic geometric grammar (PGG) framework, and the form of the geometric models used by the framework.

Chapter 3 presents algorithms for learning geometric models and rule probabilities given parsed 3D examples and a fixed grammar.

Chapter 4 describes a parsing algorithm for the PGG framework, which allows the classification of unlabeled 3D instances given a learned geometric model.

Chapter 5 describes the experiments that were conducted to test the framework and algorithms, and the results of these experiments.

Chapter 6 concludes and discusses future work.

Appendix A presents the fundamentals of several topics on which this thesis is built, including probabilistic context-free grammars, quaternions, and conditional multivariate Gaussians.

[1]The use of generative models for classification contrasts with the use of discriminative models, which is the other major classification paradigm in machine learning and artificial intelligence. Discriminative models represent enough information about each class to "discriminate" members of that class from those of other classes, but not enough information to generate new examples of the class from scratch.

Figure 1-1: The distinction between the gross shape and structure of an object versus detailed shape and texture information. This thesis focuses on capturing shape information at the level of the left image, rather than the right. [22]

1.1 Objectives

The focus of this work is to design a representation framework for classes of objects that captures *structural variability* within object classes. We use the term *object class* to refer to a "basic" semantic class of physical objects, such as "chair" or "lamp". Structural variability within an object class, then, refers to discrete variations in the existence, number, or arrangement of the parts of objects within a single class—these differences can be thought of as defining possibly overlapping subclasses within the object class, such as "chairs with legs and arms", "chairs with wheels and arms", and "chairs with legs and no arms".

In Section 1.3, we argue that structural variability is highly related to the number and type of the parts and their physical relationships with one another. Therefore, we are more interested in the gross overall shape and structure of the object parts than in the details of their precise shapes, materials, or textures. See Figure 1-1 for an example of this distinction.

Thus, the primary objectives of this thesis can be summarized as follows:

- **to design a representation framework for classes of objects that captures gross structural shape and *structural variability* within object classes;**
- **to investigate algorithms for learning models and recognizing instances in this framework;** and
- **to demonstrate that this framework learns more quickly (performs more effectively given fewer training examples) than baseline approaches.**

NP	$\mapsto$	ART NP	ART	$\mapsto$	*the*
NP	$\mapsto$	ADJ NP	ADJ	$\mapsto$	*big*
NP	$\mapsto$	N	ADJ	$\mapsto$	*red*
			N	$\mapsto$	*barn*
			N	$\mapsto$	*ball*

Figure 1-2: A context-free grammar for a tiny subset of English noun phrases.

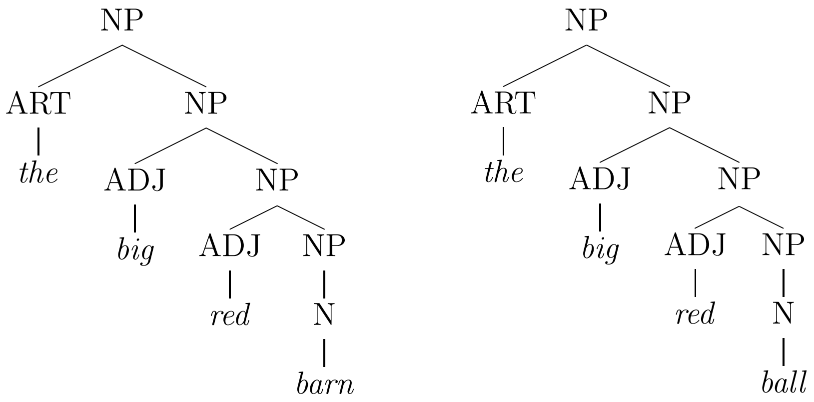


Figure 1-3: Parse trees for the English noun phrases *the big red barn* and *the big red ball*.

The probabilistic geometric grammar (PGG) framework is designed with these objectives in mind. The framework uses probabilistic context-free grammars to recursively represent classes of objects, such as chairs and tables, in terms of their parts. In Chapter 5, we shall show that the PGG models can indeed outperform the baseline models when trained with fewer examples.

More generally, however, this work seeks to combine previous approaches that have been used in object recognition in the past. In particular, we shall discuss in Section 1.4 that the (deterministic) use of 3D models was quite common in older object recognition work, but has largely given way in more recent work to highly statistical 2D methods. By applying modern probabilistic machine learning techniques to the older three-dimensional approaches, we hope to leverage the strengths of both, while avoiding some of the weaknesses inherent to each when used alone.

This thesis is focused on recognition and learning given three-dimensional input, although eventually recognition and learning must occur from two-dimensional images; see Chapter 6 for possible future work in this area.

1.2 Grammars for Language and Objects

Before we present the motivations for our approach, let us informally discuss how grammars, such as those used to model natural and theoretical languages, might be used to represent classes of objects.

The concept of a context-free grammar (CFG) is borrowed from theoretical com-

chair	$\mapsto$	top base
top	$\mapsto$	seat back
top	$\mapsto$	seat back arm arm
base	$\mapsto$	leg leg leg leg
base	$\mapsto$	axle wheel-leg wheel-leg wheel-leg

Figure 1-4: A simple context-free grammar for chairs.

puter science, linguistics, and natural language processing. In these fields, a grammar for a language is a formal specification of the strings or sentences that are members of that language. Parsing is the process by which a sentence is analyzed to decide whether it is a member of the language, and its structure determined according to the grammar [1]. An example of a grammar for a tiny subset of noun phrases in the English language is shown in Figure 1-2.

A crucial feature of a context-free grammar for a language is its ability to represent the structure of a sentence recursively. Thus *the big red barn* is a noun phrase, but it is also composed of the article *the* and the noun phrase *big red barn*, which is in turn composed of an adjective *big* and the noun phrase *red barn*, and so on. The context-free aspect of a CFG allows a compact representation of substitution—both *barn* and *ball* are nouns, so either can serve as the object of the modifying phrase *the big red*. (See Figure 1-3.)

When using grammars for object recognition, visual two- or three-dimensional images of objects are analogous to sentences, and parsing is the recognition mechanism by which an object's class and internal part structure are determined.

Using CFGs for object recognition exploits the hierarchical and substitutive structure inherent to many types of objects. For example, almost all chairs can be divided into a "top" and "base", but on some chairs the base consists of four legs, while on others it consists of a central post (an "axle") and some number of low horizontal branches with wheels on the ends ("wheel-legs"). An example of a simple context-free grammar for chairs is given in Figure 1-4.

Unlike strings in a language, objects in the world have geometric properties that must be modeled as well. The PGG framework incorporates geometric information into the grammar itself; it models the 3D geometric characteristics of object parts using multivariate conditional Gaussians over dimensions, position, and rotation. The approach is explained in full in Section 2.2.4.

Another major difference between strings in a language and objects is that, unlike the words of a sentence, the parts of an objects do not have a natural or inherent ordering. There is no "correct" order in which to match the four legs of a chair to the four "leg" parts on the right side of a rule—all possible assignments must be considered. This introduces a significant additional level of complexity to the parsing problem for objects, for which there is no analogy in language.

Figure 1-5: The same object can appear drastically different when observed from different viewpoints. [8]

1.3 Motivation

Given the objectives of this thesis as outlined above, the PGG framework shall combine several traditional approaches: the use of three-dimensional models, a parts-based approach, and the use of probabilistic grammars to capture structural variability. In Section 1.4, we shall briefly outline previous work in each of these areas. In this section, however, we explain our motivation for the choice of each approach.

1.3.1 Three-Dimensional Models

As we have mentioned, the majority of the current work in object recognition is largely image based. Thus, because of its rarity, the use of three-dimensional (3D) models must be motivated, and we do so in several ways.

View-based versus Structural Variation

First, the infinite variations in the appearance of objects in a class can be loosely separated into two categories:

- variations that occur between multiple views of the same object instance—including pose (see Figure 1-5), illumination, etc.; and

Figure 1-6: Man-made objects exhibit a large amount of structural variability [6].

- variations that occur between different instances of a single object within its class of objects—including structure (see Figure 1-6), material, texture, etc.

Another way of thinking about this distinction would be to consider a set of different images of objects with the same class. The variation among some of the images is due to differences in the inherent *shape* of the objects represented by the images, while the variation among other images is due to differences in appearance, despite the images representing objects with the same shape. For a visual example, see Figure 1-7.

The drastic contrasts in the appearance of a single object instance from viewpoint to viewpoint and under different lighting conditions, can be explained more compactly and accurately as the composition of the 3D shape of the object with the viewing projection and illumination, than as a collection of views.

Different instances of an object class are subject to countless sources of variation in appearance, as well. The sources of variation include structural variability within the object class, as well as differences in material, texture, and reflectance properties. Certainly there are some classes of objects that are primarily defined by these "image-based" variations like texture, such as paintings.

In this thesis, however, we are focused primarily on capturing this structural variability within object classes. Thus, with this goal, and with classification and learning in mind, the most natural way to represent the object class is with the characteristics shared by all members of the class, which are generally three-dimensional characteristics such as shape and relative position of parts. These are the characteristics of the class that we would expect to generalize most effectively to unseen instances of the class, so a three-dimensional representation may enable more efficient learning from

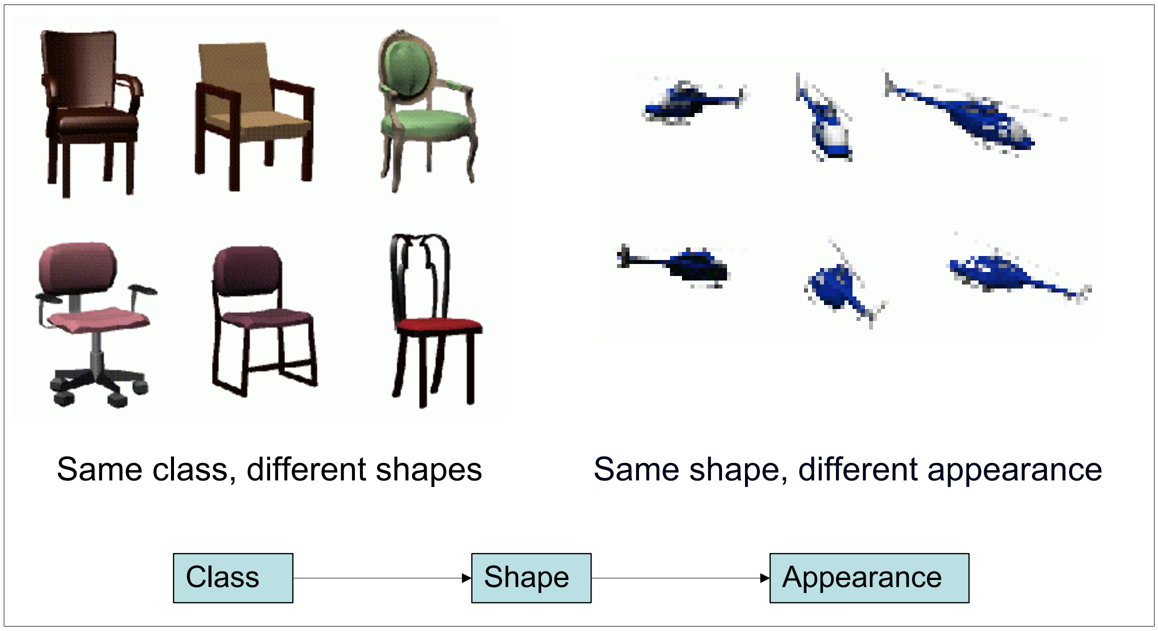


Figure 1-7: The distinction between objects with the same class but different shapes, and objects with the same shape but different appearances. [22]

fewer examples.

Parts-based Recognition

Second, the use of three-dimensional models can allow a much more intuitive parts-based approach to recognition (motivated in Section 1.3.2), because such strategies allow the spatial relationship between parts to be modeled independent of viewpoint. The use of 3D models can also lead to a principled way of dealing with occluded or missing parts; although this thesis does not explore this opportunity, future work certainly will do so.

Because, as mentioned above, we are more interested in the gross overall structure of the object parts than in the details of their shapes or textures, we choose to represent all primitive parts as simple three-dimensional boxes; this choice is discussed further in Section 2.2.3. However, the framework could be extended to richer representations of 3D shape, similar to those described by Forsyth and Ponce [13].

Three-dimensional Interaction

Third, the long term goal of object recognition is to allow interaction between an intelligent agent and the recognized objects *in three-dimensional space*. The modeling and recognition of the *function* of object classes is intimately involved in this goal and is best described three-dimensionally. Like the handling of occlusion and missing parts, the goal of enabling interaction between agents and objects is also not addressed in this thesis, but that long-term goal provides further motivation for the acquisition of three-dimensional information about objects and scenes.

1.3.2 A Parts-Based Approach

Unlike three-dimensional models, a parts-based approach to recognition is relatively common in modern computer vision, as we discuss below. Representing and recognizing objects in terms of their parts is attractive for several reasons. Many object classes are too complex to be described well using a single shape or probabilistic distribution over a single shape. However, such objects can be naturally modeled as a distribution over a collection of shapes and the relationships between them.

Another reason to consider a parts-based approach is because it offers a natural way to integrate image segmentation and object recognition, which are related but often artificially separated tasks.

1.3.3 Capturing Structural Variability With Grammars

We have already informally suggested how grammars might be used to model classes of objects, but here we attempt a more thorough motivation of the use of grammars to capture structural variability within an object class.

There is high variability in shape among instances of an object class, especially in classes of objects made by humans, like furniture (refer again to Figure 1-6). The structure of human-made objects is often defined only by functional constraints and by custom. Therefore, unlike many natural object classes, such as animals, the variability among instances of a human-made object class cannot be described well using a prototype shape and a model over variations from this single shape.

The variability is highly related to the parts (further motivating the part-based approach described in Section 1.3.2) and displays certain modular, hierarchical, and substitutive structure. There is structure in the type and number of parts that are present; for example, a chair consists of a back, seat, and four legs, or a back, seat, two arms, and four legs, but not a back, seat, one arm, and three legs.

There are also conditional independences in the presence and shape of the parts; whether a chair has arms or not is independent of whether its base consists of four legs or an axle and wheel-legs, given the location of the seat and back.

Context-free grammars are ideal for capturing structural variability in object classes. They model hierarchical groupings and substitution of subparts, and can naturally represent conditional independences between subgroups with the context-free assumption. They also allow a compact representation of the combinatorial variability in complex human-made shapes. Refer again to the simple context-free grammar for chairs in Figure 1-4.

An extension to basic CFGs, probabilistic context-free grammars (PCFGs) allow the specification of distributions over the combination of subparts by attaching probabilities to the rule expansions for a given head class (see Figure 1-8). PCFGs are the basis on which PGG models will be built.

1.0	chair	$\mapsto$	top base
0.4	top	$\mapsto$	seat back
0.6	top	$\mapsto$	seat back arm arm
0.7	base	$\mapsto$	leg leg leg leg
0.3	base	$\mapsto$	axle wheel-leg wheel-leg wheel-leg

Figure 1-8: A simple probabilistic context-free grammar for chairs.

1.3.4 A Final Motivation

We conclude this section with a final motivation for the combined use of a three-dimensional representation and probabilistic grammars. In the previous sections, we argued that three-dimensional models enable a more natural parts-based approach to recognition because such models effectively represent the spatial relationship between parts independent from the viewpoint and lighting conditions. This line of reasoning applies equally well to the use of grammars. Grammars are more suitable for use with a 3D representation than with one in 2D because the grammatical structure can focus on modeling only the structural variability due to the combination and shape of parts, and not on modeling view-based variation in the object class, which is systematic and independent from the structural variability.

1.4 Related Work

The visual recognition and classification of objects by computers has been the subject of extensive research for decades. Because of the enormous amount of literature in this area, we only provide a brief survey of general approaches to the problem here. We also focus on previous work that is closely related to the approaches used in this thesis: the use of three-dimensional (3D) models, a parts-based approach, and context-free grammars.

1.4.1 Approaches to Object Recognition

Three-Dimensional Models

In the 1970's and 80's, object recognition research was largely characterized by the use of high-level three-dimensional models of shape.

One approach to modeling the qualitative relationship between solid shapes and their images is the use of *aspect graphs*. Introduced by Koenderink and van Doorn in 1976 [20], aspect graphs represent a finite, discrete set of views of a 3D shape, so that each node of a graph corresponds to a view, and an edge between two nodes means that a small camera motion may cause a dramatic change in image structure from one view to the next. However, although they are intuitively appealing, exact aspect graphs are extremely difficult to build and use in recognition. The use of approximate aspect graphs has shown more promising results. Forsyth and Ponce describe the mathematics and uses of aspect graphs in Chapter 20 of *Computer Vision: A Modern Approach* [13].

Another instance of the use of 3D models is the work on 3D volumetric primitives that can be used to represent parts of an object, and then connected using relations among the parts. (We return to the idea of object parts and relations between them, but in 2D, below.) The most well-known type of volumetric primitives are *generalized cylinders*, originally introduced by Binford in 1971 [5], and also known as *generalized cones*. A generalized cylinder, informally, is a solid swept by a one-dimensional set of cross-sections that smoothly deform into one another [13]. Generalized cylinders were appealing because they provide an intuitive mathematical description for simplified shape representations for many relatively complex objects.

Currently, however, the use of 3D volumetric primitives and geometric inference on the relations between them is quite unpopular. This is possibly because the methods that have been developed do not scale well to large numbers of objects, and because the simplest approaches do not deal at all with hierarchical abstraction among levels of object classes or generality between object classes[2]. Furthermore, it is always difficult to build accurate 3D models of object parts. And, there has been little work on introducing probabilistic inference and learning to this area, which might help to overcome some of these limitations.

More generally, the use of three-dimensional models in any form is not common in current work, with a few exceptions; e.g. the work of Rothganger et al. (2003) [30].

Model-Based Vision

Much of the research in the 1980's and early 90's approached the object recognition problem by attempting to build libraries of relatively detailed two- or three-dimensional models of objects, and recognize instances based on these libraries. This *model-based* approach to computer vision focuses on the relationship between object features, image features, and camera models. In particular, there is assumed to be a collection of geometric models of the objects to be recognized, called the "model-base". Algorithms solve the problems of alignment, correspondence, and registration of points and regions between the test and model objects, as well as inferring the pose of an object from its image and the camera parameters.

Model-based vision encounters problems, again however, because acquiring or learning the collection of geometric models in the first place is difficult and expensive. Furthermore, as Forsyth and Ponce say, it "scales poorly with increasing numbers of models. Linear growth in the number of models occurs because the modelbase is *flat*. There is no hierarchy, and every model is treated the same way." [13]

Representative work from this body of research includes that of Huttenlocher and Ullman (1986) [16], Rothwell et al. (1992) [31], and Beis and Lowe (1993) [3]. Forsyth and Ponce also offer a survey of model-based vision in Chapter 18 of *Computer Vision: A Modern Approach* [13].

[2]One of the benefits of using grammars for object recognition is that it provides a natural way to encompass the idea of hierarchical abstraction among levels of object classes.

Template Matching

A popular approach to object recognition in recent years has been the use of statistical classifiers to find 2D *templates*—image windows that have a simple shape and stylized content. According to this approach, object recognition can be thought of as a search over all image windows and a test on each window for the presence of the object. Thus the recognition task reduces to a machine learning problem in which each image windows (or some function of it) is an input vector to a statistical classifier.

The most well-known domains for the pure form of this approach are the detection and recognition of faces, pedestrians, and road signs. Prominent examples include the *eigenfaces* face recognition algorithm presented by Turk and Pentland (1991) [36], Viola and Jones' real-time face detection system (2001) [37], and Papageorgiou and Poggio's work on pedestrian detection (2000) [26].

Template matching, in its most simple version, has been shown to offer excellent performance on object classes that can be easily represented two-dimensionally—hence the success with faces—because it can exploit statistical approaches to discriminate on the basis of brightness and color information. However, template matching has limited applicability to the recognition of large or complex classes of objects, because it is purely view-based, usually requires a separate segmentation step, and can be sensitive to changes in illumination. [13]

Relational Matching and Parts-Based Recognition

A more robust use of templates is as descriptions of *parts* of objects; then entire object classes can be described using relations between templates. There is quite a bit of current work on statistically modeling the relations between object parts, almost entirely in two dimensions—usually the object parts are simple image patches.

Prominent recent examples include the *constellation* model of Fergus et al. (2003) [11], and the K-fans work done by Crandall et al. (2005) [9].

Although relational matching using templates can be more successful than simple template matching, it is still inherently image and view-based; thus the approach works well on local patches or simple objects, but it does not provide for generalization between views of the same object, or within or between object classes.

Besides relational matching, however, there are numerous approaches that can be considered "parts-based". As we mentioned above, key reason to consider a parts-based approach is because it offers a natural way to integrate image segmentation and object recognition, which are related but often artificially separated tasks. An example of this combination is the work of Tu et al. (2003) [35], in which "generic regions" can be seen as two-dimensional parts.

Classification versus Recognition versus Detection

Before continuing, we should note that there exists an important distinction between the tasks of general object *classification*—that is, assigning a given object instance to its appropriate object class—and specific object *recognition* – that is, remembering that a given object instance is the same as an instance that was seen previously.

So, although *object recognition* is the commonly accepted term for the subfield of computer vision that deals with both these problems, it should formally refer only to the second. Furthermore, yet another task that often falls under the umbrella term of object recognition is object *detection*—determining the presence and location of an object in an image.

We use the terms *recognition* and *classification* more or less interchangeably in this document, because of this customary blurring of the distinction in the literature, but this thesis is actually focused on object classification, rather than recognition. This thesis also does not consider the task of object detection within an image.

1.4.2 Cognitive Science Perspectives

The field of cognitive science offers some insight into the task of object recognition, from the perspective of the human vision system. In particular, there is great debate among cognitive scientists about the form of the representation used by the human brain for classes of objects. One dimension of the debate is concerned with the 2D versus 3D nature of the internal representation of object classes.

Some researchers argue that the internal representation is inherently three-dimensional, possibly supplemented by cached 2D views. Biederman, one of the canonical proponents of this theory, states that "there is a representation of an object's shape independent of its position, size, and orientation (up to occlusion and accretion)" (2001) [4]. He proposes a three-dimensional and hierarchical representation in which *geons*—simple 3D geometric shapes—are the primitives from which all complex objects can be composed. Hummel supports his high-level theory in his paper *Where view-based theories break down: the role of structure in shape perception and object recognition* (2000) [15].

Others, however, insist that a full 3D representation is impossible, and instead claim that the brain represents a class of objects as a collection of 2D models. Tarr provides the prototypical voice on this side of the argument, presenting a range of experimental and computational evidence for viewpoint *dependence* of basic visual recognition tasks in humans; thus he concludes that "visual object recognition, regardless of the level of categorization, is mediated by viewpoint-dependent mechanisms" (2001) [34].

In fact, computation models of the human visual recognition system designed using the collection of 2D models approach have heretofore worked better than those using the 3D modeling theory. However, as computer scientists, we have the liberty of using cognitive science perspectives as mere inspiration rather than gospel; much of this chapter discusses our reasons for using 3D models, albeit supplemented in the future by 2D view or image-based techniques.

1.4.3 Context-Free Grammars

As discussed in Section 1.2, grammars have been used to model languages and sentence structure in linguistics and natural language processing for decades. Section A.1 offers an introduction to probabilistic context-free grammars (PCFGs) for language.

Thorough surveys of CFGs and PCFGs in natural language processing can be found in Allen [1], Jurafsky and Martin [19], and Manning and Schütze [23].

As with three-dimensional models, the use of (deterministic) grammars and syntactic models in computer vision and pattern recognition were quite popular in very early computer vision research. Rosenfeld's 1973 ACM survey gives a brief description of contemporary work on syntactic pattern recognition and two-dimensional picture grammars [29]. However, grammars have all but disappeared from modern computer vision. Recent notable exceptions are constrained to two dimensions and include Pollak et al., who use PCFGs to classify 2D images (2003) [27], and Moore and Essa, who use PCFGs to recognize activity sequences from 2D images (2001) [25].

1.5 Strengths and Weaknesses

In this chapter, we have presented a variety of arguments in favor of the approach of this thesis, but there are naturally inherent weaknesses, in addition to the strengths. Here we attempt to address these trade-offs explicitly.

A key weakness of the 3D modeling approach is that accurate three-dimensional models are notoriously difficult to learn—this issue was raised numerous times in the discussion of previous work above. In large part, this explains the general shift of the computer vision community away from earlier 3D approaches and towards image-based object recognition techniques that has occurred in recent years.

Another weakness of the 3D approach is that it is not completely understood how best to recognize an object from its 2D image using a 3D model, although there has been a large amount of research on this problem.

However, we have stated that the objectives of this thesis are limited to capturing gross structural information. Thus, we are not attempting to achieve extremely faithful models in the current framework, because ultimately we will exploit image-based methods as well. We discuss possible future work on this front very briefly in Chapter 6. We also propose some possibilities for the interaction between 3D models and 2D images in that chapter.

Furthermore, the use of 3D models has the host of strengths that we have discussed. In particular, a 3D modeling approach offers the best hope of capturing the crucial information necessary to learn quickly and perform robust classification given the learned models, which is one of the primary objectives of this thesis.

Chapter 2

The Probabilistic Geometric Grammar Framework

In Chapter 1, we motivated the use of probabilistic context-free grammars to model classes of objects. A PCFG allows a compact representation of conditional independences between parts, and also defines a probabilistic distribution over instances of the object classes defined by the grammar.

In this chapter, we describe how probabilistic geometric grammars (PGGs) extend generic PCFGs by incorporating geometric models into the nonterminals and rule parts. The chapter is organized as follows:

Section 2.1 defines PGG models as PCFGs, and introduces some new notation and terminology.

Section 2.2 discusses some issues with incorporating geometric information to a PCFG, and introduces the two types of geometric models in a PGG.

Section 2.3 describes the form of a root geometric model in a PGG as a multivariate Gaussian distribution over the space of geometric characteristics of object parts.

Section 2.4 describes the form of a part geometric model in a PGG as a *conditional* multivariate Gaussian distribution over object parts.

Section 2.5 discusses the conditional independence assumptions expressed by a PGG, and their consequences on the expressive power of the framework.

Section 2.6 derives the process of calculating the likelihood of a parsed labeled object instance according to a PGG model.

Note that this chapter builds directly on the material presented in Appendix A: the fundamentals of PCFGs, quaternions for rotation, Gaussian distributions over quaternions, and conditional multivariate Gaussians. The reader who is not familiar with these topics is encouraged to read Appendix A before continuing.

In the subsequent sections, follow along with the descriptions using the example PGG for chairs as shown in Figure 2-1. Ignore the variables until Section 2.2, the ρs until Section 2.3, and the ϕs until Section 2.4.

[1] chair[ρ^1].

    [1] 1.0 chair(C) $\mapsto$ [1] chair-top(CT)[ϕ^{111}], [2] chair-base(CB)[ϕ^{112}].

[2] chair-top[ρ^2].

    [1] 0.4 chair-top(CT) $\mapsto$ [1] chair-back(ck)[ϕ^{211}], [2] chair-seat(cs)[ϕ^{212}].

    [2] 0.6 chair-top(CT) $\mapsto$ [1] chair-back(ck)[ϕ^{221}], [2] chair-seat(cs)[ϕ^{222}], [3] chair-arm($ca1$)[ϕ^{223}], [4] chair-arm($ca2$)[ϕ^{224}].

[3] chair-base[ρ^3].

    [1] 0.5 chair-base(CB) $\mapsto$ [1] chair-leg($cl1$)[ϕ^{311}], [2] chair-leg($cl2$)[ϕ^{312}], [3] chair-leg($cl3$)[ϕ^{313}], [4] chair-leg($cl4$)[ϕ^{314}].

    [2] 0.2 chair-base(CB) $\mapsto$ [1] chair-axle(cx)[ϕ^{321}], [2] chair-wheel-leg($cw1$)[ϕ^{322}], [3] chair-wheel-leg($cw2$)[ϕ^{323}], [4] chair-wheel-leg($cw3$)[ϕ^{324}].

    [3] 0.3 chair-base(CB) $\mapsto$ [1] chair-axle(cx)[ϕ^{331}], [2] chair-wheel-leg($cw1$)[ϕ^{332}], [3] chair-wheel-leg($cw2$)[ϕ^{333}], [4] chair-wheel-leg($cw3$)[ϕ^{334}], [5] chair-wheel-leg($cw4$)[ϕ^{335}], [6] chair-wheel-leg($cw5$)[ϕ^{336}].

[4] chair-back[ρ^4].
[5] chair-seat[ρ^5].
[6] chair-arm[ρ^6].
[7] chair-leg[ρ^7].
[8] chair-axle[ρ^8].
[9] chair-wheel-leg[ρ^9].

Figure 2-1: A PGG for chairs.

2.1 An Introduction to PGGs

Formally, a probabilistic geometric grammar (PGG) G is defined as a set of object or part classes[1]:

$$G = \{C^1, \ldots, C^N\} \ .$$

A class C^i is defined by a set of rules:

$$C^i = \{r^{i1}, \ldots, r^{in_i}\}$$

[1] In Appendix A, we use the terms *nonterminal* and *terminal*, because they are traditional in the CFG literature. For PGGs, however, we use the term *part class*, or simply *class*, instead of nonterminal, and *primitive class* instead of terminal, because they better reflect the role of these elements in the grammar – they label and modify primitive or composite geometric parts, rather than existing as free-standing symbolic entities as in CFGs for language. We use the term *object class* to refer to a part class that corresponds to a "basic" semantic class of physical objects, such as "chair" or "lamp".

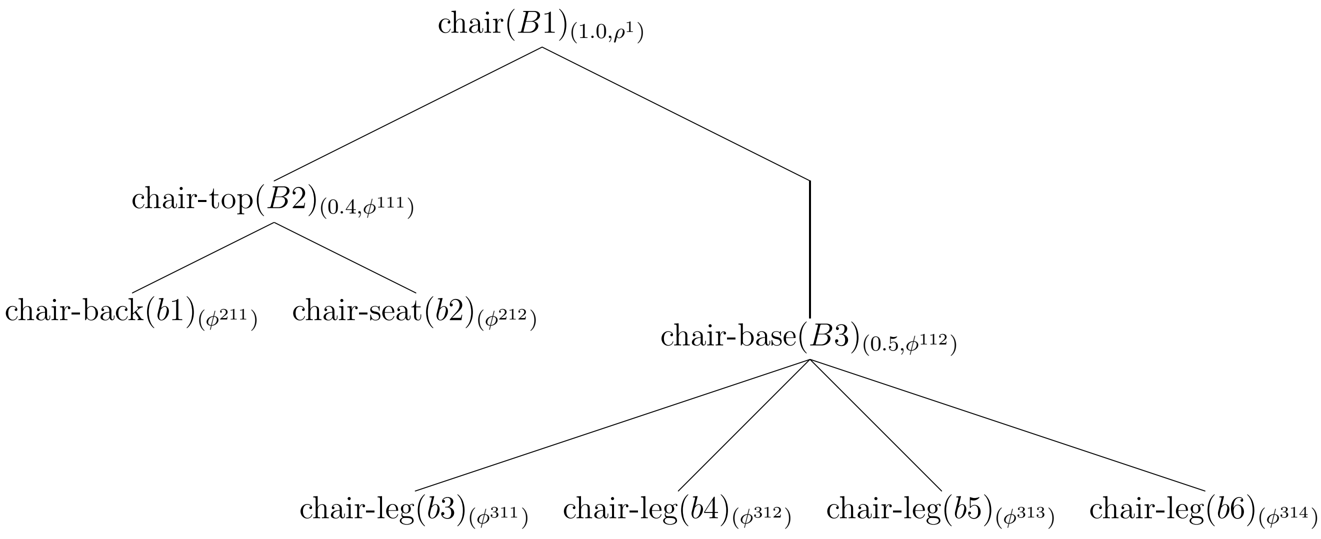


Figure 2-2: A PGG parse tree for a chair with four legs and no arms.

where the class C^1 is designated as the special starting class. A class is called *primitive* if its rule set is empty; i.e., if no rule expands it. Otherwise it is called a *composite* class. In Figure 2-1, chair is the starting class, the classes chair, chair-top, and chair-base are composite, while chair-back, chair-seat, chair-arm, chair-leg, chair-axle, and chair-wheel-leg are primitive.

The rules of a PGG define how composite part classes are broken down into subpart classes. A rule $r^{ij} \in C^i$ maps the head class C^i to an ordered sequence of rule parts ξ^{ij} with a probability γ^{ij}, and is written in the form:

$$\gamma^{ij} \;\; C^i \mapsto \xi^{ij} \quad .$$

We will also refer to the sequence of rule parts ξ^{ij} as the *right hand side* (RHS) of the rule r^{ij}.

The kth rule part of ξ^{ij} is written s_c^{ijk}. Each rule part s_c^{ijk} has an associated class:

$$\text{class}(s_c^{ijk}) = C^c \in G \quad .$$

Notationally we shall use the subscript c to indicate the index of the class of s_c^{ijk} in the PGG. (This is unnecessary when rule parts are written out, as in Figure 2-1.) The class of a rule part may be primitive or composite. For example, in Figure 2-1, the classes of both rule parts in the chair rule – chair-top and chair-base – are composite. In contrast, the classes of all the rule parts in all other rules are primitive.

The expansion probability γ^{ij} is the likelihood $\Pr(r^{ij} \,|\, C^i)$ that the rule r^{ij} is chosen given the head class C^i. Thus, the expansion probabilities for all the rules of a class must sum to one:

$$\forall C^i \in G \; \sum_j \gamma^{ij} = 1 \quad .$$

Given a PGG, parsing is the process by which the primitive parts of an instance are analyzed to determine the best possible hierarchical structure according to the grammar. The hierarchical structure that is produced is called a parse tree, and the primitive instance parts form the leaves of the tree. A parse tree that could have been produced by the PGG in Figure 2-1 is shown in Figure 2-2.

2.2 Incorporating Geometric Information

So far, our definition of a PGG has been very similar to that of a simple PCFG for language. In order to model classes of three-dimensional objects, however, we need to introduce geometric information into the grammar.

2.2.1 Variables and Constant Symbols

To facilitate the representation of geometric information in the grammar, we add variables. A variable represents the geometric characteristics of a primitive or composite part. Notationally, we write "classname(var)" to denote that the class label modifies the part and its geometric characteristics. Equivalently, the primitive and composite parts of instances are represented with constant symbols.[2]

For convenience, we adopt the convention that lowercase variables and symbols represent primitive parts, while uppercase variables and symbols represent composite parts. This parallels the use of lowercase letters for terminals and uppercase letters for nonterminals that is traditional in CFGs. Variables are in most cases hand chosen to be reminiscent of the class of the part (e.g., the variable CT for a part of class chair-top). Symbols, however, are anonymously named b or B with a number appended, in order to reflect the uncertainty regarding the classes of the instance parts they represent.

2.2.2 Issues With Geometric Grammars

As we discuss representing geometric information in a grammar, it is important that the reader fully understand the relationship between the left and right hand sides of the grammar rules. A rule states that a part of the class on the left hand side of the rule can be broken up into parts of the number and classes on the right hand side of the rule. Thus a rule expresses a *consists of* relationship. Similarly, any composite instance part in a parse tree also consists of its children – in Figure 2-2, the chair-base $B3$ consists of its children, the legs $b3$, $b4$, $b5$, and $b6$.[3]

[2] The use of variables and constant symbols will also be semantically helpful when we consider parsing. The task of parsing is to find the best possible assignment of parts in the instance to rule parts in the grammar, given the geometric models in the grammar and the geometric characteristics of the instance parts; this can also be thought of as finding the best possible binding of variables in the grammar to constants in the instance.

[3] The *consists of* relationship that is inherent to context-free grammars is a crucial difference between the PGG framework and other approaches in object recognition that also use trees to represent the relationship between object parts, such as Huttenlocher's work on k-fans [9]. In most

It is also necessary to understand the context-free nature of a PGG, and its consequences for modeling geometric information. The phrase "context-free" means that any instance part with a particular class label c_i can be matched against any rule part with the same class label, *regardless of the surrounding context*; in other words, regardless of the classes or arrangement of the other rule parts on the right hand side of that rule.

Because of the *consists of* relationship between parent and children parts, any part represented by the left hand side variable must have geometric characteristics that "summarize" the geometric characteristics of all the parts represented by the right hand side variables. For example, in the first rule of the chair-top class in the PGG in Figure 2-1, we could write:

$$CT = \text{summarize}(ck, cs)$$

while for the second rule of the same class we have:

$$CT = \text{summarize}(ck, cs, ca1, ca2) \ .$$

However, consider the context-free aspect of a PGG: the CT part created by either of these rules needs to be able to be used in any situation that a part of class chair-top is called for, independent of the number and arrangement of parts in its internal structure. Therefore, any candidate summarization function must be able to package up the geometric characteristics of a set of subparts into a consistent "interface", with consistent dimensionality, that can be consistently interpreted regardless of the number of subparts or their individual properties.

In theory, this geometric interface and summarization function must only be consistent across different rules within the same class. In practice, however, a single interface can be used for all classes throughout the grammar. In the next section, we will argue that a simple three-dimensional box is a reasonable choice for a geometric interface and the bounding box function is a good candidate for summarization.

2.2.3 Representing Object Parts as 3D Boxes

The primary focus of this thesis is capturing structural variability within object classes, which we have argued is highly related to the number and type of the parts and their physical relationships with one another. Because we are more interested in the gross overall structure of the object parts than in the details of their shapes or textures, we choose to represent all primitive parts as simple three-dimensional boxes.

We have also explained the necessity of a consistent geometric interface across all

approaches, including the PGG framework, a tree is used to represent the statistical conditional independences among the parts. In other approaches, however, all nodes of the tree including internal ones are primitive parts of the object. In the PGG framework, primitive parts can only exist at the leaves of the tree, while the internal nodes represent non-primitive composite parts such as chair-base. This is consistent with the use of grammars in language: actual words can only exist at the leaves of the parse tree of a sentence, while internal nodes such as NP represent a composite structure containing multiple words.

rules for a given composite part class. Furthermore, just as for primitive parts, the most important geometric information to capture for composite parts is related to their general size and position in the scene with respect to other parts. For these reasons, a three-dimensional box representation is an adequate choice for composite parts as well.

As an added motivation, the choice of a consistent representation for all parts, primitive or composite, significantly simplifies the modeling task.

A three-dimensional box can be fully specified with:

- three half-dimensions,
- a three-dimensional center position point, and
- a rotation with respect to a coordinate frame;

i.e., as a vector

$$\mathbf{b} = \begin{bmatrix} \mathbf{d} \\ \mathbf{p} \\ \hat{q} \end{bmatrix}$$

of three magnitudes, three reals, and a unit quaternion. (The choice of quaternions as a representation for rotation is discussed in detail in Section A.2.) Thus we define the space of all possible primitive and composite parts to be:

$$\mathbb{B} \stackrel{\mathbf{def}}{=} \mathbb{R}^{+3} \times \mathbb{R}^3 \times \hat{\mathbb{H}} \ .$$

Since we have chosen that all primitive and composite parts shall be represented as 3D boxes, we can use the minimum volume bounding box as a simple, intuitive, and consistent geometric summarization function. The minimum volume bounding box of a set of boxes can be efficiently approximated using the algorithm described in Section A.4. This may seem like a rather arbitrary choice, but we will defend it further in Chapter 4.

2.2.4 Types of Geometric Models

The geometric models in a PGG are of two types:

- A **root geometric model** ρ^i is defined for each class C^i in the PGG G. It is probability distribution that describes how the geometric characteristics of the root part vary, independently of any parent or children parts.
- A **part geometric model** ϕ^{ijk}, in contrast, is defined for each rule part s_c^{ijk} on the right hand side of each rule r^{ij} in G. It is a probability distribution that describes how the geometric characteristics of the rule part vary conditioned on the characteristics of its parent part (of class C^i).

Multivariate Gaussians are a natural choice for the geometric model distributions. However, because we are modeling all primitive and composite object parts as three-dimensional boxes, the space of geometric representations of a single part is the set of all possible 3D boxes $\mathbb{B}$. And, vectors in $\mathbb{B}$ are not elements of a Euclidean vector space $\mathbb{R}^n$ – we discuss in Section A.2 that the quaternion group $\hat{\mathbb{H}}$ is not a vector space, so it would be suspicious indeed if vectors containing quaternions were members of one.

In the next two sections, we shall discuss how to circumvent this problem in order to define unconditional and conditional multivariate Gaussians over $\mathbb{B}$.

2.3 Root Geometric Models

A root geometric model ρ^i takes the form of a multivariate Gaussian over the space of geometric characteristics of object parts. In this section, we first show how to define a multivariate Gaussian over the group of unit quaternions, and then over the specialized space of 3D boxes $\mathbb{B} = \mathbb{R}^{+3} \times \mathbb{R}^3 \times \hat{\mathbb{H}}$.

2.3.1 Multivariate Gaussians Over Quaternions

Recall that in Section A.2.3 we show how to define a Gaussian distribution over the group of unit quaternions $\hat{\mathbb{H}}$:

$$\mathrm{p}(\hat{q}) = \mathcal{N}(\hat{q}; \hat{\mu}, \mathbf{\Sigma}) = \frac{1}{(2\pi)^{3/2}|\mathbf{\Sigma}|^{1/2}} \exp\left\{ -\frac{1}{2} \ln(\hat{\mu}^* \hat{q})^T \mathbf{\Sigma}^{-1} \ln(\hat{\mu}^* \hat{q}) \right\}$$

where $\hat{q}$ is the query quaternion, $\hat{\mu}$ is the mean quaternion, and $\mathbf{\Sigma}$ is the 3×3 covariance matrix defined in the tangent space at the mean.

This result can be easily extended to deal with vectors of quaternions by performing the mode-tangent operation point-wise with the elements of the mean and query vectors, and then concatenating the resulting 3-vectors in the tangent space. If the mean and and query vectors are:

$$\boldsymbol{\mu} = \begin{bmatrix} \hat{\mu}_1 \\ \vdots \\ \hat{\mu}_n \end{bmatrix} \qquad\qquad \mathbf{q} = \begin{bmatrix} \hat{q}_1 \\ \vdots \\ \hat{q}_n \end{bmatrix}$$

then, with a small abuse of notation, we can write the mode-tangent of $\boldsymbol{\mu}$ and $\mathbf{q}$ as:

$$\mathbf{m} = \ln(\boldsymbol{\mu}^* \mathbf{q}) = \begin{bmatrix} \ln(\hat{\mu}_1^* \hat{q}_1) \\ \vdots \\ \ln(\hat{\mu}_n^* \hat{q}_n) \end{bmatrix} \quad .$$

The covariance matrix will then have dimensions $(3n \times 3n)$, because the logarithmic map converts each single quaternion into a 3-vector in $\mathbb{R}^3$. The resulting density function can be written as:

$$\mathrm{p}(\mathbf{q}) = \mathcal{N}(\mathbf{q}; \boldsymbol{\mu}, \mathbf{\Sigma}) = \frac{1}{(2\pi)^{3n/2}|\mathbf{\Sigma}|^{1/2}} \exp\left\{ -\frac{1}{2} \ln(\boldsymbol{\mu}^* \mathbf{q})^T \mathbf{\Sigma}^{-1} \ln(\boldsymbol{\mu}^* \mathbf{q}) \right\} \quad . \tag{2.1}$$

2.3.2 Multivariate Gaussians Over Object Parts

Multivariate Gaussians Over $\mathbb{B}^*$

As we discussed above, an object part in a PGG is represented as a vector of three magnitudes, three reals, and a unit quaternion; $\mathbf{b} \in \mathbb{B}$. The space $\mathbb{B}$ is a subspace of a more general space:

$$\mathbb{B}^* = \mathbb{R}^{+m} \times \mathbb{R}^r \times \hat{\mathbb{H}}^t$$

for which $m = r = 3$ and $t = 1$. Clearly $\mathbb{B}^*$ is not a Euclidean vector space.

Thus we must define multivariate Gaussians in a space that is the product of subspaces of the spaces of each parameter. In other words, we apply a mapping to each element of a vector in $\mathbb{B}^*$ so that the result is a vector in $\mathbb{R}^n$, and then define the Gaussian in that space. This approach is described by Fletcher et al. [12].

Position parameters are already members of $\mathbb{R}$ so this mapping is just the identity function. Half-dimensions are strictly positive so we use log to map them into the real domain. And of course, we have just seen how to map quaternions (and vectors of quaternions) into a vector space where they can be treated as normal vectors.

Consider a query vector in $\mathbb{B}^*$:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_d \\ \mathbf{x}_p \\ \mathbf{x}_q \end{bmatrix}$$

where $\mathbf{x}_d$ is a vector of m magnitudes, $\mathbf{x}_p$ is a vector of r reals, and $\mathbf{x}_q$ is a vector of t unit quaternions. We first calculate the deviation of $\mathbf{x}$ from the mean $\boldsymbol{\mu}$, and then map it into $\mathbb{R}^n$; this is like an extended version of the zero-centered *mode-tangent* vector we discuss in Section A.2.3:

$$\mathbf{m} = \begin{bmatrix} \log(\mathbf{x}_d) - \log(\boldsymbol{\mu}_d) \\ \mathbf{x}_p - \boldsymbol{\mu}_p \\ \ln(\boldsymbol{\mu}_q^* \mathbf{x}_q) \end{bmatrix} = \begin{bmatrix} \log(\mathbf{x}_d / \boldsymbol{\mu}_d) \\ \mathbf{x}_p - \boldsymbol{\mu}_p \\ \ln(\boldsymbol{\mu}_q^* \mathbf{x}_q) \end{bmatrix} .$$

We can now write the probability density function for $\mathbf{x}$ as:

$$\mathrm{p}(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{n/2} |\boldsymbol{\Sigma}|^{1/2}} \exp\left\{ -\frac{1}{2} \mathbf{m}^T \boldsymbol{\Sigma}^{-1} \mathbf{m} \right\} \tag{2.2}$$

where $n = m + r + 3t$, and

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_d \\ \boldsymbol{\mu}_p \\ \boldsymbol{\mu}_q \end{bmatrix} \qquad \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{dd} & \boldsymbol{\Sigma}_{dp} & \boldsymbol{\Sigma}_{dq} \\ \boldsymbol{\Sigma}_{pd} & \boldsymbol{\Sigma}_{pp} & \boldsymbol{\Sigma}_{pq} \\ \boldsymbol{\Sigma}_{qd} & \boldsymbol{\Sigma}_{qp} & \boldsymbol{\Sigma}_{qq} \end{bmatrix} .$$

Multivariate Gaussians over $\mathbb{B}$

Based on Equation (2.2), the root geometric models ρ^i that are associated with each class C^i in a PGG take the form:

$$\rho^i = \langle \boldsymbol{\mu}^i, \boldsymbol{\Sigma}^i \rangle, \qquad \boldsymbol{\mu}^i = \begin{bmatrix} \boldsymbol{\mu}^i_d \\ \boldsymbol{\mu}^i_p \\ \hat{\mu}^i_q \end{bmatrix} \qquad \boldsymbol{\Sigma}^i = \begin{bmatrix} \boldsymbol{\Sigma}^i_{dd} & \boldsymbol{\Sigma}^i_{dp} & \boldsymbol{\Sigma}^i_{dq} \\ \boldsymbol{\Sigma}^i_{pd} & \boldsymbol{\Sigma}^i_{pp} & \boldsymbol{\Sigma}^i_{pq} \\ \boldsymbol{\Sigma}^i_{qd} & \boldsymbol{\Sigma}^i_{qp} & \boldsymbol{\Sigma}^i_{qq} \end{bmatrix}$$

such that for a simple box

$$\mathbf{b} = \begin{bmatrix} \mathbf{d} \\ \mathbf{p} \\ \hat{q} \end{bmatrix}$$

in which $\mathbf{d}$ has length 3, $\mathbf{p}$ has length 3, and $\hat{q}$ is a single unit quaternion, the *root geometric score* of the part $\mathbf{b}$ given the model ρ^i is calculated as:

$$\boxed{\mathrm{p}(\mathbf{b} \mid \rho^i) = \mathcal{N}(\mathbf{b}; \boldsymbol{\mu}^i, \boldsymbol{\Sigma}^i) = \frac{1}{(2\pi)^{9/2}|\boldsymbol{\Sigma}^i|^{1/2}} \exp\left\{ -\frac{1}{2} \begin{bmatrix} \log(\mathbf{d}/\boldsymbol{\mu}^i_d) \\ \mathbf{p} - \boldsymbol{\mu}^i_d \\ \ln((\hat{\mu}^i_q)^* \hat{q}) \end{bmatrix}^T (\boldsymbol{\Sigma}^i)^{-1} \begin{bmatrix} \log(\mathbf{d}/\boldsymbol{\mu}^i_d) \\ \mathbf{p} - \boldsymbol{\mu}^i_d \\ \ln((\hat{\mu}^i_q)^* \hat{q}) \end{bmatrix} \right\}} \ . \tag{2.3}$$

2.4 Part Geometric Models

Unlike the root geometric models, which are unconditional, the part geometric models ϕ^{ijk} take the form of conditional multivariate Gaussians. In this section, we first show how to define a conditional multivariate Gaussian over vectors of strictly positive real numbers, then over vectors of unit quaternions, and finally over the specialized space of 3D boxes $\mathbb{B} = \mathbb{R}^{+3} \times \mathbb{R}^3 \times \hat{\mathbb{H}}$. In each section, we follow the approach of Section A.3, in which we factored a multivariate Gaussian over $\mathbb{R}^n$ into component marginal and conditional distributions.

2.4.1 Conditional Multivariate Gaussians Over Magnitudes

In Section 2.3.2, we showed how to use log to map half-dimensions in $\mathbb{R}^+$, which are strictly positive, to the real domain $\mathbb{R}$. We use that technique here and also extend the factorization approach of Section A.3 in order to define a conditional multivariate Gaussian over vectors of magnitudes in $\mathbb{R}^{+n}$.

We have a vector $\mathbf{x}$ of magnitudes of length n, and we would like to partition it into two subvectors $\mathbf{x}_1$ and $\mathbf{x}_2$, of lengths n_1 and n_2 respectively such that $n_1 + n_2 = n$:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} \quad .$$

We can define a joint multivariate Gaussian distribution for $\mathrm{p}(\mathbf{x}) = \mathrm{p}(\mathbf{x}_1, \mathbf{x}_2)$ using the approach shown in Section 2.3.2, and we would like to factor the joint into the marginal distribution $\mathrm{p}(\mathbf{x}_1)$ and the conditional distribution $\mathrm{p}(\mathbf{x}_2 \,|\, \mathbf{x}_1)$.

As before, we partition the mean and covariance parameters of the joint Gaussian in the same way we partitioned $\mathbf{x}$:

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix} \qquad\qquad \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}$$

and write out the joint Gaussian distribution:

$$\mathrm{p}(\mathbf{x}) = \frac{1}{(2\pi)^{(n_1+n_2)/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{ -\frac{1}{2} \begin{bmatrix} \log(\mathbf{x}_1/\boldsymbol{\mu}_1) \\ \log(\mathbf{x}_2/\boldsymbol{\mu}_2) \end{bmatrix}^T \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}^{-1} \begin{bmatrix} \log(\mathbf{x}_1/\boldsymbol{\mu}_1) \\ \log(\mathbf{x}_2/\boldsymbol{\mu}_2) \end{bmatrix} \right\} .$$

Using Equations (A.4a) and (A.5) from Section A.3, we can then factor this expression for the joint, yielding expressions for the marginal and conditional distributions:

$$\mathrm{p}(\mathbf{x}_1) = \frac{1}{(2\pi)^{n_1/2}|\boldsymbol{\Sigma}_{11}|^{1/2}} \exp\left\{ -\frac{1}{2} \log\left(\frac{\mathbf{x}_1}{\boldsymbol{\mu}_1}\right)^T \boldsymbol{\Sigma}_{11}^{-1} \log\left(\frac{\mathbf{x}_1}{\boldsymbol{\mu}_1}\right) \right\} \tag{2.4}$$

$$\begin{aligned} \mathrm{p}(\mathbf{x}_2 \,|\, \mathbf{x}_1) = {} & \frac{1}{(2\pi)^{n_2/2}|\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{11}|^{1/2}} \\ & \times \exp\left\{ -\frac{1}{2} \left(\log\left(\frac{\mathbf{x}_2}{\boldsymbol{\mu}_2}\right) - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1} \log\left(\frac{\mathbf{x}_1}{\boldsymbol{\mu}_1}\right) \right)^T (\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{11})^{-1} \right. \\ & \qquad \left. \times \left(\log\left(\frac{\mathbf{x}_2}{\boldsymbol{\mu}_2}\right) - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1} \log\left(\frac{\mathbf{x}_1}{\boldsymbol{\mu}_1}\right) \right) \right\} . \end{aligned} \tag{2.5}$$

We can summarize the parameters $\langle \boldsymbol{\mu}_1^m, \boldsymbol{\Sigma}_1^m \rangle$ of the marginal Gaussian over magnitudes as:

$$\boldsymbol{\mu}_1^m = \boldsymbol{\mu}_1 \tag{2.6a}$$

$$\boldsymbol{\Sigma}_1^m = \boldsymbol{\Sigma}_{11} \tag{2.6b}$$

and the parameters $\langle \boldsymbol{\mu}_{2|1}^c, \boldsymbol{\Sigma}_{2|1}^c \rangle$ of the conditional Gaussian as:

$$\boldsymbol{\mu}_{2|1}^c = \boldsymbol{\mu}_2 \exp\left(\boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1} \log\left(\frac{\mathbf{x}_1}{\boldsymbol{\mu}_1}\right) \right) \tag{2.7a}$$

$$\boldsymbol{\Sigma}_{2|1}^c = \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12} \;\; . \tag{2.7b}$$

2.4.2 Conditional Multivariate Gaussians Over Quaternions

Again we follow the now quite familiar approach from Section A.3 of factoring a multivariate Gaussian over $\mathbb{R}^n$ into component marginal and conditional distributions, this time in order to define a conditional multivariate Gaussian over vectors of unit quaternions.

Formal Definition

We have a vector $\mathbf{q}$ of unit quaternions of length n, and again we would like to partition it into two subvectors $\mathbf{q}_1$ and $\mathbf{q}_2$, of lengths n_1 and n_2 respectively such that $n_1 + n_2 = n$.

$$\mathbf{q} = \begin{bmatrix} \mathbf{q}_1 \\ \mathbf{q}_2 \end{bmatrix} \quad .$$

We can define a joint multivariate Gaussian distribution for $\mathrm{p}(\mathbf{q}) = \mathrm{p}(\mathbf{q}_1, \mathbf{q}_2)$ using Equation (2.1), and we would like to factor it into the marginal distribution $\mathrm{p}(\mathbf{q}_1)$ and the conditional distribution $\mathrm{p}(\mathbf{q}_2 \,|\, \mathbf{q}_1)$.

As before, we begin by partitioning the mean and covariance parameters of the joint Gaussian in the same way we partitioned $\mathbf{q}$:

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix} \qquad\qquad \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}$$

where $\boldsymbol{\mu}$ has length $n_1 + n_2$ and $\boldsymbol{\Sigma}$ has dimensions $3(n_1 + n_2) \times 3(n_1 + n_2)$. Then we can write the joint Gaussian distribution as:

$$\mathrm{p}(\mathbf{q}) = \frac{1}{(2\pi)^{3(n_1+n_2)/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{ -\frac{1}{2} \begin{bmatrix} \ln(\boldsymbol{\mu}_1^* \mathbf{q}_1) \\ \ln(\boldsymbol{\mu}_2^* \mathbf{q}_2) \end{bmatrix}^T \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}^{-1} \begin{bmatrix} \ln(\boldsymbol{\mu}_1^* \mathbf{q}_1) \\ \ln(\boldsymbol{\mu}_2^* \mathbf{q}_2) \end{bmatrix} \right\} \quad .$$

Using Equations (A.4a) and (A.5) from Section A.3, we can then factor this expression for the joint. In the process, we must be extremely careful to respect the non-commutativity of quaternion multiplication! The factoring yields expressions for the marginal and conditional distributions:

$$\mathrm{p}(\mathbf{q}_1) = \frac{1}{(2\pi)^{3n_1/2}|\boldsymbol{\Sigma}_{11}|^{1/2}} \exp\left\{ -\frac{1}{2} \ln(\boldsymbol{\mu}_1^* \mathbf{q}_1)^T \boldsymbol{\Sigma}_{11}^{-1} \ln(\boldsymbol{\mu}_1^* \mathbf{q}_1) \right\} \tag{2.8}$$

$$\begin{aligned} \mathrm{p}(\mathbf{q}_2 \,|\, \mathbf{q}_1) = {} & \frac{1}{(2\pi)^{3n_2/2}|\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{11}|^{1/2}} \\ & \times \exp\left\{ -\frac{1}{2} \Big(\ln(\boldsymbol{\mu}_2^* \mathbf{q}_2) - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} \ln(\boldsymbol{\mu}_1^* \mathbf{q}_1) \Big)^T (\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{11})^{-1} \right. \\ & \qquad \left. \times \Big(\ln(\boldsymbol{\mu}_2^* \mathbf{q}_2) - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} \ln(\boldsymbol{\mu}_1^* \mathbf{q}_1) \Big) \right\} \quad . \end{aligned} \tag{2.9}$$

Note that these expressions are identical to Equations (2.4) and (2.5), except for a $3n_2$ in the normalizing constant rather than n_2, and $\ln(\boldsymbol{\mu}_2^* \mathbf{q}_2)$ and $\ln(\boldsymbol{\mu}_1^* \mathbf{q}_1)$ terms rather than $\log\left(\frac{\mathbf{x}_2}{\boldsymbol{\mu}_2}\right)$ and $\log\left(\frac{\mathbf{x}_1}{\boldsymbol{\mu}_1}\right)$ in the expression for the deviation.

A Gaussian in the Tangent Space Only

In Sections A.3 and 2.4.1 we continued on to parameterize a new Gaussian distribution for the marginal and conditional in terms of the partitions of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. In this case, it is clear that the marginal is a Gaussian, parameterized the same as before:

$$\boldsymbol{\mu}_1^m = \boldsymbol{\mu}_1$$
$$\boldsymbol{\Sigma}_1^m = \boldsymbol{\Sigma}_{11}$$

and that the covariance of the conditional is also the same as before:

$$\boldsymbol{\Sigma}_{2|1}^c = \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12} \quad .$$

However, the mean of the conditional is not so straightforward. (So far we have been performing point-wise operations on vectors of unit quaternions, but let us switch to dealing with single unit quaternions $\hat{q}_1$, $\hat{\mu}_1$, $\hat{q}_2$, and $\hat{\mu}_2$ for clarity. We then also assume that the matrices $\boldsymbol{\Sigma}_{21}$ and $\boldsymbol{\Sigma}_{11}$ each have dimensions 3×3.)

In order to find a mean for the conditional distribution, we need to find a $\hat{\mu}$ such that:

$$\ln(\hat{\mu}^*\hat{q}_2) = \ln(\hat{\mu}_2^*\hat{q}_2) - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\ln(\hat{\mu}_1^*\hat{q}_1) \quad .$$

Thus we need to solve for $\hat{\mu}$, while also allowing the $\hat{q}_2$ on each side to cancel out. We might try the following approach :

$$\begin{aligned}
\ln(\hat{\mu}^*\hat{q}_2) &= \ln(\hat{\mu}_2^*\hat{q}_2) - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\ln(\hat{\mu}_1^*\hat{q}_1) \\
\hat{\mu}^*\hat{q}_2 &= \exp\big(-\boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\ln(\hat{\mu}_1^*\hat{q}_1) + \ln(\hat{\mu}_2^*\hat{q}_2)\big) \\
&\stackrel{?}{=} \exp\big(-\boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\ln(\hat{\mu}_1^*\hat{q}_1)\big)\exp\big(\ln(\hat{\mu}_2^*\hat{q}_2)\big) \\
\hat{\mu}^*\hat{q}_2\hat{q}_2^{-1} &= \exp\big(-\boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\ln(\hat{\mu}_1^*\hat{q}_1)\big)\hat{\mu}_2^*\hat{q}_2\hat{q}_2^{-1} \\
\hat{\mu} &= \Big(\exp\big(-\boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\ln(\hat{\mu}_1^*\hat{q}_1)\big)\hat{\mu}_2^*\Big)^* \quad .
\end{aligned}$$

However, the operation marked with the $\stackrel{?}{=}$ is in fact incorrect. The problem arises because quaternion multiplication is not commutative in general, as we have foreshadowed. This means the standard identities for exponential and log functions do not necessarily hold. In particular the identity

$$\exp(p)\exp(q) = \exp(p+q)$$

is only allowed if p and q are a *commutative subgroup* of the quaternion group $\mathbb{H}$. This, in turn, is true if and only if the vector components of p and q are equal; in other words, if the quaternions are parallel, which is not the case here.

Another way to think about this is that we are adding a displacement d to the point $\ln(\boldsymbol{\mu}_2^*\mathbf{q}_2)$ in the tangent space at the mean, for $d = -\boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\ln(\boldsymbol{\mu}_1^*\mathbf{q}_1)$. But, the general operation of adding a displacement to a point in the tangent space cannot necessarily be described as a rotation on the quaternion hypersphere. Thus, although $f(\mathbf{q}_2) = \ln(\boldsymbol{\mu}_2^*\mathbf{q}_2) - d$ is a well defined function in the tangent space, it cannot be described as a rotation, so there can be no $\boldsymbol{\mu}$ such that $f(x_2) = \ln(\boldsymbol{\mu}^*\mathbf{q}_2)$.

What does this mean? Quite simply – there is no beautiful and compact way of describing the factored conditional distribution of a multivariate Gaussian over quaternions. The conditional distribution given in Equation (2.9) above is still a perfectly valid probability density function, however, and it is actually still Gaussian in the tangent space. Therefore, this result is sufficient for our part geometric models.

2.4.3 Conditional Multivariate Gaussians Over Object Parts

We have now demonstrated how to define multivariate distributions over vectors of reals in $\mathbb{R}^n$, vectors of magnitudes in $\mathbb{R}^{+n}$, and vectors of unit quaternions. Now, we can combine these results to define the general form of a conditional multivariate Gaussian distribution over $\mathbb{B}^* = \mathbb{R}^{+m} \times \mathbb{R}^r \times \hat{\mathbb{H}}^t$.

Conditional Multivariate Gaussians Over $\mathbb{B}^*$

Again we consider a query vector in $\mathbb{B}^*$:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_d \\ \mathbf{x}_p \\ \mathbf{x}_q \end{bmatrix}$$

where $\mathbf{x}_d$ is a vector of m magnitudes, $\mathbf{x}_p$ is a vector of r reals, and $\mathbf{x}_q$ is a vector of t unit quaternions. We partition the query vector so that each section of the original vector is subdivided, and so that $\mathbf{x}_1$ and $\mathbf{x}_2$ refer to the appropriately extracted portions of $\mathbf{x}$, concatenated together:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_{d1} \\ \mathbf{x}_{d2} \\ \mathbf{x}_{p1} \\ \mathbf{x}_{p2} \\ \mathbf{x}_{q1} \\ \mathbf{x}_{q2} \end{bmatrix} \qquad \mathbf{x}_1 = \begin{bmatrix} \mathbf{x}_{d1} \\ \mathbf{x}_{p1} \\ \mathbf{x}_{q1} \end{bmatrix} \qquad \mathbf{x}_2 = \begin{bmatrix} \mathbf{x}_{d2} \\ \mathbf{x}_{p2} \\ \mathbf{x}_{q2} \end{bmatrix} .$$

Let the lengths of $\mathbf{x}_{d1}$, $\mathbf{x}_{p1}$, and $\mathbf{x}_{q1}$ be m_1, r_1, and t_1, respectively, and the lengths of $\mathbf{x}_{d2}$, $\mathbf{x}_{p2}$, and $\mathbf{x}_{q2}$ be m_2, r_2, and t_2.

Now, we perform the analogous partitioning with a mean vector and covariance matrix for a joint multivariate Gaussian over $\mathbb{B}^*$:

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_{d1} \\ \boldsymbol{\mu}_{d2} \\ \boldsymbol{\mu}_{p1} \\ \boldsymbol{\mu}_{p2} \\ \boldsymbol{\mu}_{q1} \\ \boldsymbol{\mu}_{q2} \end{bmatrix} \qquad \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{d1d1} & \boldsymbol{\Sigma}_{d1d2} & \boldsymbol{\Sigma}_{d1p1} & \boldsymbol{\Sigma}_{d1p2} & \boldsymbol{\Sigma}_{d1q1} & \boldsymbol{\Sigma}_{d1q2} \\ \boldsymbol{\Sigma}_{d2d1} & \boldsymbol{\Sigma}_{d2d2} & \boldsymbol{\Sigma}_{d2p1} & \boldsymbol{\Sigma}_{d2p2} & \boldsymbol{\Sigma}_{d2q1} & \boldsymbol{\Sigma}_{d2q2} \\ \boldsymbol{\Sigma}_{p1d1} & \boldsymbol{\Sigma}_{p1d2} & \boldsymbol{\Sigma}_{p1p1} & \boldsymbol{\Sigma}_{p1p2} & \boldsymbol{\Sigma}_{p1q1} & \boldsymbol{\Sigma}_{p1q2} \\ \boldsymbol{\Sigma}_{p2d1} & \boldsymbol{\Sigma}_{p2d2} & \boldsymbol{\Sigma}_{p2p1} & \boldsymbol{\Sigma}_{p2p2} & \boldsymbol{\Sigma}_{p2q1} & \boldsymbol{\Sigma}_{p2q2} \\ \boldsymbol{\Sigma}_{q1d1} & \boldsymbol{\Sigma}_{q1d2} & \boldsymbol{\Sigma}_{q1p1} & \boldsymbol{\Sigma}_{q1p2} & \boldsymbol{\Sigma}_{q1q1} & \boldsymbol{\Sigma}_{q1q2} \\ \boldsymbol{\Sigma}_{q2d1} & \boldsymbol{\Sigma}_{q2d2} & \boldsymbol{\Sigma}_{q2p1} & \boldsymbol{\Sigma}_{q2p2} & \boldsymbol{\Sigma}_{q2q1} & \boldsymbol{\Sigma}_{q2q2} \end{bmatrix} \tag{2.10}$$

$$\boldsymbol{\mu}_1 = \begin{bmatrix} \boldsymbol{\mu}_{d1} \\ \boldsymbol{\mu}_{p1} \\ \boldsymbol{\mu}_{q1} \end{bmatrix} \quad \boldsymbol{\Sigma}_{11} = \begin{bmatrix} \boldsymbol{\Sigma}_{d1d1} & \boldsymbol{\Sigma}_{d1p1} & \boldsymbol{\Sigma}_{d1q1} \\ \boldsymbol{\Sigma}_{p1d1} & \boldsymbol{\Sigma}_{p1p1} & \boldsymbol{\Sigma}_{p1q1} \\ \boldsymbol{\Sigma}_{q1d1} & \boldsymbol{\Sigma}_{q1p1} & \boldsymbol{\Sigma}_{q1q1} \end{bmatrix} \quad \boldsymbol{\Sigma}_{12} = \begin{bmatrix} \boldsymbol{\Sigma}_{d1d2} & \boldsymbol{\Sigma}_{d1p2} & \boldsymbol{\Sigma}_{d1q2} \\ \boldsymbol{\Sigma}_{p1d2} & \boldsymbol{\Sigma}_{p1p2} & \boldsymbol{\Sigma}_{p1q2} \\ \boldsymbol{\Sigma}_{q1d2} & \boldsymbol{\Sigma}_{q1p2} & \boldsymbol{\Sigma}_{q1q2} \end{bmatrix} \tag{2.11}$$

$$\boldsymbol{\mu}_2 = \begin{bmatrix} \boldsymbol{\mu}_{d2} \\ \boldsymbol{\mu}_{p2} \\ \boldsymbol{\mu}_{q2} \end{bmatrix} \quad \boldsymbol{\Sigma}_{21} = \begin{bmatrix} \boldsymbol{\Sigma}_{d2d1} & \boldsymbol{\Sigma}_{d2p1} & \boldsymbol{\Sigma}_{d2q1} \\ \boldsymbol{\Sigma}_{p2d1} & \boldsymbol{\Sigma}_{p2p1} & \boldsymbol{\Sigma}_{p2q1} \\ \boldsymbol{\Sigma}_{q2d1} & \boldsymbol{\Sigma}_{q2p1} & \boldsymbol{\Sigma}_{q2q1} \end{bmatrix} \quad \boldsymbol{\Sigma}_{22} = \begin{bmatrix} \boldsymbol{\Sigma}_{d2d2} & \boldsymbol{\Sigma}_{d2p2} & \boldsymbol{\Sigma}_{d2q2} \\ \boldsymbol{\Sigma}_{p2d2} & \boldsymbol{\Sigma}_{p2p2} & \boldsymbol{\Sigma}_{p2q2} \\ \boldsymbol{\Sigma}_{q2d2} & \boldsymbol{\Sigma}_{q2p2} & \boldsymbol{\Sigma}_{q2q2} \end{bmatrix} \tag{2.12}$$

Let

$$n = m + r + 3t \qquad n_1 = m_1 + r_1 + 3t_1 \qquad n_2 = m_2 + r_2 + 3t_2 \ .$$

Then the covariance matrix dimensions are as follows:

$$\begin{array}{lll} \boldsymbol{\Sigma} : n \times n & \boldsymbol{\Sigma}_{11} : n_1 \times n_1 & \boldsymbol{\Sigma}_{12} : n_1 \times n_2 \\ & \boldsymbol{\Sigma}_{21} : n_2 \times n_1 & \boldsymbol{\Sigma}_{22} : n_2 \times n_2 \ . \end{array}$$

We can easily factor this joint distribution to parameterize a marginal Gaussian distribution for $\mathrm{p}(\mathbf{x}_1)$:

$$\begin{aligned} \mathrm{p}(\mathbf{x}_1) &= \mathcal{N}(\mathbf{x}_1; \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11}) \\ &= \frac{1}{(2\pi)^{n_1/2} |\boldsymbol{\Sigma}_{11}|^{1/2}} \exp\left\{ -\frac{1}{2} \begin{bmatrix} \log(\mathbf{x}_{d1}/\boldsymbol{\mu}_{d1}) \\ \mathbf{x}_{p1} - \boldsymbol{\mu}_{p1} \\ \ln(\boldsymbol{\mu}_{q1}^* \mathbf{x}_{q1}) \end{bmatrix}^T \boldsymbol{\Sigma}_{11}^{-1} \begin{bmatrix} \log(\mathbf{x}_{d1}/\boldsymbol{\mu}_{d1}) \\ \mathbf{x}_{p1} - \boldsymbol{\mu}_{p1} \\ \ln(\boldsymbol{\mu}_{q1}^* \mathbf{x}_{q1}) \end{bmatrix} \right\} \ . \end{aligned} \tag{2.13}$$

For the factored conditional distribution $\mathrm{p}(\mathbf{x}_2 \,|\, \mathbf{x}_1)$, we define a mode-tangent vector $\mathbf{m}$ which combines all the deviations and mappings for the specialized spaces that we have seen, as well as the general formula for the deviation in a conditional Gaussian:

$$\mathbf{m} = \begin{bmatrix} \log(\mathbf{x}_{d2}/\boldsymbol{\mu}_{d2}) \\ \mathbf{x}_{p2} - \boldsymbol{\mu}_{p2} \\ \ln(\boldsymbol{\mu}_{q2}^* \mathbf{x}_{q2}) \end{bmatrix} - \boldsymbol{\Sigma}_{22} \boldsymbol{\Sigma}_{11}^{-1} \begin{bmatrix} \log(\mathbf{x}_{d1}/\boldsymbol{\mu}_{d1}) \\ \mathbf{x}_{p1} - \boldsymbol{\mu}_{p1} \\ \ln(\boldsymbol{\mu}_{q1}^* \mathbf{x}_{q1}) \end{bmatrix} \ .$$

We can now write the conditional distribution for $\mathrm{p}(\mathbf{x}_2 \,|\, \mathbf{x}_1)$ as:

$$\mathrm{p}(\mathbf{x}_2 \,|\, \mathbf{x}_1) = \frac{1}{(2\pi)^{n_2/2} |\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{11}|^{1/2}} \exp\left\{ -\frac{1}{2} \mathbf{m}^T (\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{11})^{-1} \mathbf{m} \right\} \tag{2.14}$$

where of course

$$\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{11} = \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12} \ .$$

Conditional Multivariate Gaussians over $\mathbb{B}$

Now, we can finally define the form of the part geometric model ϕ^{ijk} that is associated with each rule part s_c^{ijk} in each rule in a PGG. A part geometric model takes the form:

$$\phi^{ijk} = \langle \boldsymbol{\mu}_1^{ijk}, \boldsymbol{\mu}_2^{ijk}, \boldsymbol{\Sigma}_{11}^{ijk}, \boldsymbol{\Sigma}_{12}^{ijk}, \boldsymbol{\Sigma}_{21}^{ijk}, \boldsymbol{\Sigma}_{22}^{ijk} \rangle$$

where:

$$\boldsymbol{\mu}_1^{ijk} = \begin{bmatrix} \boldsymbol{\mu}_{d1}^{ijk} \\ \boldsymbol{\mu}_{p1}^{ijk} \\ \boldsymbol{\mu}_{q1}^{ijk} \end{bmatrix} \qquad \boldsymbol{\mu}_2^{ijk} = \begin{bmatrix} \boldsymbol{\mu}_{d2}^{ijk} \\ \boldsymbol{\mu}_{p2}^{ijk} \\ \boldsymbol{\mu}_{q2}^{ijk} \end{bmatrix}$$

and the covariance matrices have similar internal structure, as shown in Equations (2.11) and (2.12).

Then for a simple box $\mathbf{b}_2$ and its parent part $\mathbf{b}_1$:

$$\mathbf{b}_1 = \begin{bmatrix} \mathbf{d}_1 \\ \mathbf{p}_1 \\ \hat{q}_1 \end{bmatrix} \qquad \mathbf{b}_2 = \begin{bmatrix} \mathbf{d}_2 \\ \mathbf{p}_2 \\ \hat{q}_2 \end{bmatrix}$$

in which $\mathbf{d}_1$, $\mathbf{d}_2$, $\mathbf{p}_1$, and $\mathbf{p}_2$ each have length 3, and $\hat{q}_1$ and $\hat{q}_2$ are single unit quaternions, the *part geometric score* of the part $\mathbf{b}_2$ given its parent $\mathbf{b}_1$ and the model ϕ^{ijk} is calculated as:

$$\boxed{\mathrm{p}(\mathbf{b}_2 \,|\, \mathbf{b}_1, \phi^{ijk}) = \frac{1}{(2\pi)^{9/2}|\boldsymbol{\Sigma}^{ijk}/\boldsymbol{\Sigma}_{11}^{ijk}|^{1/2}} \exp\left\{-\frac{1}{2}\mathbf{m}^T(\boldsymbol{\Sigma}^{ijk}/\boldsymbol{\Sigma}_{11}^{ijk})^{-1}\mathbf{m}\right\}} \tag{2.15}$$

where

$$\mathbf{m} = \begin{bmatrix} \log(\mathbf{x}_{d2}/\boldsymbol{\mu}_{d2}^{ijk}) \\ \mathbf{x}_{p2} - \boldsymbol{\mu}_{p2}^{ijk} \\ \ln((\boldsymbol{\mu}_{q2}^{ijk})^*\mathbf{x}_{q2}) \end{bmatrix} - \boldsymbol{\Sigma}_{22}^{ijk}(\boldsymbol{\Sigma}_{11}^{ijk})^{-1} \begin{bmatrix} \log(\mathbf{x}_{d1}/\boldsymbol{\mu}_{d1}^{ijk}) \\ \mathbf{x}_{p1} - \boldsymbol{\mu}_{p1}^{ijk} \\ \ln((\boldsymbol{\mu}_{q1}^{ijk})^*\mathbf{x}_{q1}) \end{bmatrix}$$

and

$$\boldsymbol{\Sigma}^{ijk}/\boldsymbol{\Sigma}_{11}^{ijk} = \boldsymbol{\Sigma}_{22}^{ijk} - \boldsymbol{\Sigma}_{21}^{ijk}(\boldsymbol{\Sigma}_{11}^{ijk})^{-1}\boldsymbol{\Sigma}_{12}^{ijk} \quad .$$

2.5 Conditional Independence Assumptions

Before we discuss how to use a PGG model to calculate the likelihood of a parsed and labeled object, we must consider the conditional independence assumptions that are implied by the form of a PGG grammar and geometric models.

First, any context-free grammar assumes that the way a non-primitive class is expanded (i.e., the choice of rule used) is independent of the way its parent or sibling classes were expanded. As we have discussed several times, this assumption is the fundamental one underlying the term "context-free" in standard CFGs and PCFGs.

Second, the PGG framework assumes that, in a fixed parsed instance, the geometric characteristics of a part are conditionally independent of those of its non-descendents, given its parents, and of those of its descendents, given its children. These assumptions are identical to the topological semantics of standard Bayesian networks, in which a node is conditionally independent of all other nodes, given its Markov blanket – i.e., its parents, children, and children's parents [32]. Because PGG models produce only trees, rather than arbitrary directed acyclic graphs, a part's children will never have any other parents, so the conditional independence assumptions are the same.

What is the consequence of these assumptions on the expressive power of the PGG framework? A PGG model cannot *directly* specify geometric dependencies between parts of the object that are not related in a child-parent relationship. Of course, because we have stated that the geometric characteristics of any parent part will be some "summarization" of its children parts, a PGG model can still indirectly express many dependencies among siblings, grandparents and grandchildren, and parts with other non-parent-child relationships.

For example, a PGG model cannot directly specify that the lengths of the the legs of a chair should be the same; instead, it can specify a distribution over the dimensions of the entire base of the chair, and then penalize any leg whose length varies too greatly from the height of the base.

So, what do we gain in exchange for this limitation? As we shall see, a PGG model requires many fewer parameters than a fully connected model, and allows substantial sharing of information between models for object classes with similar subparts. These properties can dramatically increase the speed of learning and improve the learned model's ability to generalize to unseen instances.

2.6 The Likelihood of a Parsed Instance

Now, we are ready to show how to calculate the likelihood of a labeled parsed object instance using a PGG.[4] First, we describe the form of parse trees in the PGG framework.

2.6.1 PGG Parse Trees

Just like a PCFG, a PGG G can both generate new object instances and parse existing unlabeled objects. In both cases, the resulting structure takes the form of a parse tree.

In a PCFG parse tree, every node is a simple symbol – a nonterminal for an internal node, and a terminal for a leaf node. "Bookkeeping information" – recorded for each node from the rule used during generation or parsing – is limited to a simple expansion probability associated with each internal node. Refer to Section A.1.1 for details.

A parse tree t in the PGG framework is a bit more complex; refer again to Figure 2-2 for an example. Because all parts are represented as 3D boxes, every node a_ℓ in t represents an object part with an associated vector of geometric information $\mathbf{b}_\ell \in \mathbb{B}$. Notationally, this vector is represented with a constant symbol starting with b or B.

[4]We use the term *likelihood* rather than *probability* when dealing with instances in the PGG framework. This is because *probability* formally refers only to discrete probabilities, but in the PGG framework we combine discrete probabilities on the rules of the grammar with continuous density functions on the geometric characteristics of parts.

Every node also has a class label $C_\ell \in G$ that modifies the part. The class label for an internal node is composite, while in a leaf node it is primitive.

The bookkeeping information that is recorded during generation or parsing is also more complex. As in a PCFG parse tree, every internal node has an expansion probability γ_ℓ. In addition, however, every node also has a geometric model; the root has a root geometric model ρ_ℓ, while a non-root node has a part geometric model ϕ_ℓ.

Thus we can represent a node $a_\ell \in t$ as a tuple, where the contents vary slightly depending on the position of a_ℓ in t:

$$\begin{aligned}
a_\ell &= \langle C_\ell, \mathbf{b}_\ell, \gamma_\ell, \rho_\ell \rangle && a_\ell \text{ is an internal root node} \\
a_\ell &= \langle C_\ell, \mathbf{b}_\ell, \gamma_\ell, \phi_\ell \rangle && a_\ell \text{ is an internal non-root node} \\
a_\ell &= \langle C_\ell, \mathbf{b}_\ell, \rho_\ell \rangle && a_\ell \text{ is a leaf root node (a single-part object)} \\
a_\ell &= \langle C_\ell, \mathbf{b}_\ell, \phi_\ell \rangle && a_\ell \text{ is a leaf non-root node.}
\end{aligned}$$

Notationally, we assume a_1 is the root node. We also define the following functions on the structure of tree nodes:

- $\text{children}(a_\ell)$: returns a (possibly empty) sequence of children nodes of a_ℓ.
- $\text{child}(a_\ell, k)$: returns the kth element of $\text{children}(a_\ell)$, for $1 \le k \le |\text{children}(a_\ell)|$.
- $\text{parent}(a_\ell)$: returns the parent node of a_ℓ, for $\ell \neq 1$.

and these functions on the structure of parse trees (or subtrees):

$$\begin{aligned}
\text{root}(t) &= a_1 \\
\text{nonroots}(t) &= \{a_\ell \in t \,|\, \ell \neq 1\} \\
\text{internals}(t) &= \{a_\ell \in t \,|\, \text{children}(a_\ell) \neq \emptyset\} \\
\text{leaves}(t) &= \{a_\ell \in t \,|\, \text{children}(a_\ell) = \emptyset\} \quad .
\end{aligned}$$

2.6.2 Calculating the Likelihood

We can think of a PGG parse tree t as having two general types of information:

- structural information, denoted $\text{struct}(t)$, consisting of the hierarchy of the tree and the class labels on the nodes; and
- geometric information, denoted $\text{geom}(t)$, consisting of the geometric characteristics of the primitive and composite object parts represented by the nodes.

We would like to calculate $\text{p}(t \,|\, G)$. Using the distinction between structural and geometric information, and then applying the chain rule of probability, we have:

$$\begin{aligned}
\text{p}(t \,|\, G) &= \text{p}(\text{struct}(t), \text{geom}(t) \,|\, G) \\
&= \text{p}(\text{geom}(t) \,|\, \text{struct}(t), G)\, \text{P}(\text{struct}(t) \,|\, G) \quad .
\end{aligned} \tag{2.16}$$

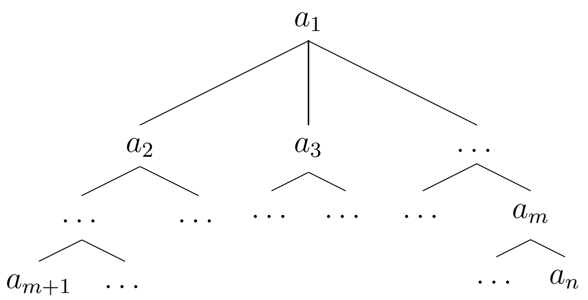


Figure 2-3: Assumed indexing of nodes for the derivation of the likelihood of a parse tree.

In the following two sections, we assume without loss of generality that the nodes in t are indexed such that the ordered sequence $a_1, a_2, \ldots, a_n$ corresponds to a breadth-first traversal of the tree starting at the root. We also let a_m be the last internal node, such that:

$$\begin{aligned} \text{internals}(t) &= \{a_1, \ldots, a_m\} \\ \text{leaves}(t) &= \{a_{m+1}, \ldots, a_n\} \quad . \end{aligned}$$

See Figure 2-3 for a depiction of the assumed indexing of nodes.

Structural Likelihood

The $\mathrm{P}(\text{struct}(t) \,|\, G)$ term is equivalent to the probability of a standard parse tree according to a PCFG. We define the *structural likelihood* of the tree t as the joint probability over the class and parent/child relationships between all nodes a_ℓ in t. Let $[C_\ell \mapsto C_g \ldots C_h]$ denote that nodes $a_g, \ldots, a_h$, with respective classes $C_g, \ldots, C_h$, are the children of node a_ℓ with class C_ℓ in t. Then we can write the joint as:

$$\mathrm{P}(\text{struct}(t) \,|\, G) = \mathrm{P}([C_1 \mapsto C_2 \ldots C_\ell], [C_2 \mapsto C_{\ell+1} \ldots C_g], \ldots, [C_m \mapsto C_h \ldots C_n], \,|\, G) \quad .$$

Now, we use the first conditional independence assumption from Section 2.5. Since we assume the way a non-primitive class is expanded (i.e., the choice of rule used) is independent of the way its parent or sibling classes were expanded, we can break up the conditional expression into a product of independent terms, each conditioned only on G. Then, each term in the new expression corresponds to the expansion probability for the rule that was used to expand the head node. Thus we can write:

$$\begin{aligned} \mathrm{P}(\text{struct}(t) \,|\, G) &= \mathrm{P}([C_1 \mapsto C_2 \ldots C_\ell], [C_2 \mapsto C_{\ell+1} \ldots C_g], \ldots, [C_m \mapsto C_h \ldots C_n], \,|\, G) \\ &= \mathrm{P}([C_1 \mapsto C_2 \ldots C_\ell] \,|\, G)\, \mathrm{P}([C_2 \mapsto C_{\ell+1} \ldots C_g] \,|\, G) \\ &\quad \times \cdots \times \mathrm{P}([C_m \mapsto C_h \ldots C_n] \,|\, G) \\ &= \gamma_1 \gamma_2 \cdots \gamma_m \\ &= \prod_{a_\ell \in \text{internals}(t)} \gamma_\ell \quad . \end{aligned}$$

Geometric Likelihood

Next we turn to the $\mathrm{p}(\mathrm{geom}(t)\,|\,\mathrm{struct}(t), G)$ term. We define the *geometric likelihood* of the tree t as the joint density over the geometric characteristics $\mathbf{b}_\ell$ of all nodes a_ℓ in t. Then, we use the chain rule to break up the expression in a convenient order.

$$\begin{aligned}
\mathrm{p}(\mathrm{geom}(t)\,|\,\mathrm{struct}(t), G) &= \mathrm{p}(\mathbf{b}_1, \ldots, \mathbf{b}_n\,|\,\mathrm{struct}(t), G) \\
&= \mathrm{p}(\mathbf{b}_n\,|\,\mathbf{b}_1 \ldots \mathbf{b}_{n-1}, \mathrm{struct}(t), G) \\
&\quad \times \mathrm{p}(\mathbf{b}_{n-1}\,|\,\mathbf{b}_1 \ldots \mathbf{b}_{n-2}, \mathrm{struct}(t), G) \\
&\quad \times \cdots \times \mathrm{p}(\mathbf{b}_2\,|\,\mathbf{b}_1, \mathrm{struct}(t), G) \\
&\quad \times \mathrm{p}(\mathbf{b}_1\,|\,\mathrm{struct}(t), G) \;\; .
\end{aligned}$$

Note that in this expression, each part $\mathbf{b}_\ell$ is conditioned on parts $\{\mathbf{b}_g\}$ for $1 \le g < \ell$, i.e., parts which are of the same generation as $\mathbf{b}_\ell$, or higher, in the tree. Thus no part is conditioned on any descendent.

Then, we use the second conditional independence assumption from above. Since we assume that each part is conditionally independent of all non-descendent parts given its parent part, we can simplify each conditional term, so that a part $\mathbf{b}_\ell$, for $\ell \neq 1$, is conditioned only on its parent part and G. (For simplicity, let the geometric characteristics of the parent of a node a_ℓ be denoted $\mathrm{parent}(\mathbf{b}_\ell)$.)

$$\begin{aligned}
\mathrm{p}(\mathrm{geom}(t)\,|\,\mathrm{struct}(t), G) &= \mathrm{p}(\mathbf{b}_n\,|\,\mathrm{parent}(\mathbf{b}_n), G)\, \mathrm{p}(\mathbf{b}_{n-1}\,|\,\mathrm{parent}(\mathbf{b}_{n-1}), G) \\
&\quad \times \cdots \times \mathrm{p}(\mathbf{b}_2\,|\,\mathbf{b}_1, G)\, \mathrm{p}(\mathbf{b}_1\,|\,G) \;\; .
\end{aligned}$$

The $\mathrm{struct}(t)$ terms can be safely removed because once the parent of a part has been determined, the geometric characteristics of that part are no longer dependent on the general structural information of the tree.

Each of the terms can now be computed using the appropriate root or part geometric model. Thus we have:

$$\begin{aligned}
\mathrm{p}(\mathrm{geom}(t)\,|\,\mathrm{struct}(t), G) &= \mathrm{p}(\mathbf{b}_n\,|\,\mathrm{parent}(\mathbf{b}_n), \phi_n)\, \mathrm{p}(\mathbf{b}_{n-1}\,|\,\mathrm{parent}(\mathbf{b}_{n-1}), \phi_{n-1}) \\
&\quad \times \cdots \times \mathrm{p}(\mathbf{b}_2\,|\,\mathbf{b}_1, \phi_2)\, \mathrm{p}(\mathbf{b}_1\,|\,\rho_1) \\
&= \mathrm{p}(\mathbf{b}_1\,|\,\rho_1) \prod_{a_\ell \in \mathrm{nonroots}(t)} \mathrm{p}(\mathbf{b}_\ell\,|\,\mathrm{parent}(\mathbf{b}_\ell), \phi_\ell) \;\; .
\end{aligned} \tag{2.17}$$

Total Likelihood

Thus, combining Equations (2.16), (2.6.2), and (2.17), we have a general expression for the likelihood of a PGG parse tree:

$$\boxed{\mathrm{p}(t\,|\,G) = \left(\prod_{a_\ell \in \mathrm{internals}(t)} \gamma_\ell \right) \mathrm{p}(\mathbf{b}_1\,|\,\rho_1) \left(\prod_{a_\ell \in \mathrm{nonroots}(t)} \mathrm{p}(\mathbf{b}_\ell\,|\,\mathrm{parent}(\mathbf{b}_\ell), \phi_\ell) \right)} \;\; . \tag{2.18}$$

Chapter 3

Learning in the PGG Framework

Learning is a crucial element in any artificial intelligence or computer vision system. In this chapter, we present algorithms for learning the numeric parameters of a PGG model, given parsed and labeled 3D examples and a fixed grammar structure. In particular, we show how to calculate the maximum likelihood estimates for the expansion probabilities on the rules, and demonstrate an algorithm for learning the root and part geometric models.

The chapter is organized as follows:

Section 3.1 describes the concept of a tree fragment and presents a matching function which maps a tree fragment to a rule in a PGG.

Section 3.2 describes a simple counting algorithm for calculating the maximum likelihood estimates for the expansion probabilities on the rules of a PGG, given parsed labeled examples and a fixed grammar structure.

Section 3.3 presents an algorithm for learning the root and part geometric models of a PGG, given parsed labeled examples and a fixed grammar structure.

Section 3.4 briefly discusses possible sources for 3D training data.

Structural learning of the grammar itself is a difficult problem that we are not considering in this thesis. Thus, given our assumption that the grammar has fixed structure and that we have been given fully parsed and labeled training examples, the algorithms presented here are quite straightforward. In future work, we will consider the problems of learning the parameters of the model from unparsed unlabeled examples, and structural learning of the grammar; see Chapter 6 for more information.

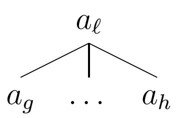
3.1 Matching Tree Fragments to Rules

In the learning algorithms for the PGG framework, we will find it convenient to have a function that maps a tree fragment in a parse tree t to its matching rule in a PGG G. This section presents such a function.

3.1.1 Tree Fragments

We define the ℓth *tree fragment* of t, denoted $\hat{a}^{\ell}_{gh}$, to be a tiny tree consisting of the internal tree node $a_\ell \in t$ and its direct children:

$\hat{a}^{\ell}_{gh}$:



where $\text{children}(a_\ell) = a_g, \ldots, a_h$.

Let $|\hat{a}^{\ell}_{gh}|$ denote the number of children in $\hat{a}^{\ell}_{gh}$. We also assume for convenience that the nodes of t are indexed according to a breadth-first traversal, so we have:

$$|\hat{a}^{\ell}_{gh}| = |\text{children}(a_\ell)| = h - g + 1 \ .$$

3.1.2 Matching To Rules

A 'matches' Predicate

We say that a tree fragment $\hat{a}^{\ell}_{gh}$ *matches* a rule r^{ij} if:

- the class C_ℓ of a_ℓ equals the head class of r^{ij};
- the number of children of $\hat{a}^{\ell}_{gh}$ equals the number of rule parts $|\xi^{ij}|$ on the right hand side of r^{ij}; and
- the class C_{g+k-1} of the kth child of $\hat{a}^{\ell}_{gh}$ equals the class of the kth rule part of the right hand side of r^{ij}.

Formally, we define this as a Boolean predicate on a tree fragment and a rule:

$$\boxed{\begin{aligned} \text{matches}(\hat{a}^{\ell}_{gh}, r^{ij}) \Leftrightarrow & \big(C_\ell = C^i\big) \\ & \wedge \big(|\hat{a}^{\ell}_{gh}| = |\xi^{ij}|\big) \\ & \wedge \Bigg(\bigwedge_{1 \le k \le |\xi^{ij}|} \big(C_{g+k-1} = \text{class}(s_c^{ijk})\big) \Bigg) \end{aligned}} \ . \tag{3.1}$$

A 'matching' Function

Let $\hat{a}^{\ell}_{gh}$ be a tree fragment in a parse tree t labeled according to a PGG G. Let $C^i = C_\ell$. Then there must exist a unique rule $r^{ij} \in C^i$ that matches $\hat{a}^{\ell}_{gh}$ according to Equation (3.1)—namely, the rule that was used to label, generate, or parse that tree fragment.

Thus we can define a matching function that returns a single rule in G given a tree fragment:

$$\boxed{\text{matching}(\hat{a}^{\ell}_{gh}) = r^{ij} \in C^i \,|\, \text{matches}(\hat{a}^{\ell}_{gh}, r^{ij})} \ . \tag{3.2}$$

3.2 Learning Expansion Probabilities on Rules

If we are given a set of parsed and labeled examples $T = \{t\}$, the expansion probabilities on the rules of a PGG G can be estimated using simple counting.

For each rule r^{ij} in each class $C^i \in G$, we count all tree fragments in each tree $t \in T$ that match the rule, and then normalize by dividing the count by the total number of tree fragments for which C^i is the class of the root node.

We can summarize this algorithm as:

1. Let G be a PGG.
2. Let $T = \{t\}$ be a set of parsed examples that were labeled according to G.
3. Initialize training sets. For each class $C^i \in G$:
   (a) Let $\sigma_i = 0$.
   (b) For each rule r^{ij} in C^i, let $\tau_{ij} = 0$.
4. For each tree fragment $\hat{a}^{\ell}_{gh}$ in each example tree $t \in T$:
   (a) Let $r^{ij} = \text{matching}(\hat{a}^{\ell}_{gh})$.
   (b) Let $\tau_{ij} = \tau_{ij} + 1$
   (c) Let $\sigma_i = \sigma_i + 1$.
5. For each rule r^{ij} in each class $C^i \in G$, let the expansion probability be:

$$\gamma^{ij} = \frac{\tau_{ij}}{\sigma_i} \quad .$$

3.3 Learning Geometric Models

We mentioned previously that one of the benefits of using a Gaussian distribution is the ability to learn its parameters from example data in an efficient closed-form manner.

However, recall that the space of object parts in a PGG, $\mathbb{B}$, is a subspace of the general space:

$$\mathbb{B}^* = \mathbb{R}^{+m} \times \mathbb{R}^r \times \hat{\mathbb{H}}^t$$

in which $m = r = 3$ and $t = 1$. And, as we repeated numerous times in Chapter 2, $\mathbb{B}^*$ is not a Euclidean vector space. Therefore, the familiar maximum likelihood estimates for multivariate Gaussian distributions over $\mathbb{R}^n$ cannot be applied without modification.

In this section, we first show how to estimate the parameters of an unconditional and conditional multivariate Gaussian distribution over unit quaternions and $\mathbb{B}^*$ from a set of examples. Then we present an algorithm for learning the root and part geometric models of a PGG from parsed and labeled training examples.

3.3.1 Estimating Multivariate Gaussians over Quaternions

Estimating the Mean

Recall that in Section A.2.4, we present Johnson's algorithm for estimating the quaternion mean of a set of unit quaternions [17]. To summarize, if $\mathbf{Q}$ is the $4 \times N$ data matrix with the ith sample unit quaternion $\hat{q}_i$ as the ith column, $\hat{\mu}$ is the maximal eigenvector of $\mathbf{Q}\mathbf{Q}^T$, or its negation. We can apply this algorithm point-wise to a set of example vectors of unit quaternions $\mathbf{q}_i \in Q$, and let $\boldsymbol{\mu}$ be the resulting vector of quaternion means.

Thus we summarize the mean estimation algorithm for vectors of unit quaternions as follows:

1. Let $Q = \{\mathbf{q}_i\}$ be a set of sample vectors of unit quaternions, where each vector $\mathbf{q}_i$ has length n and the size of Q is N.
2. For $j = 1$ to n:
   (a) Let $Q_j = \{\hat{q}_j\}$ be a set containing the jth element of each sample vector $\mathbf{q}_i \in Q$.
   (b) Let $\mathbf{Q}_j$ be the $4 \times N$ data matrix where the ith column is the vector representation of the ith unit quaternion in Q_j.
   (c) Let $\mathbf{A}_j = \mathbf{Q}_j \mathbf{Q}_j^T$.
   (d) Let $\mathbf{e}_{jk}$ be an eigenvector of $\mathbf{A}_j$ with real eigenvalue a_{jk}.
   (e) Choose one of the two eigenvectors $\pm\mathbf{e}_{j*}$ associated with the maximal eigenvalue a_{j*} as the estimate of the mean $\hat{\mu}_j$.
3. Let $\boldsymbol{\mu}$ be the estimate of the vector mean, where the jth element is $\hat{\mu}_j$.

Hemispherization

Once the quaternion mean vector has been determined, we must hemispherize each quaternion vector according to the estimated mean. Again, we simply adapt Johnson's algorithm from Section A.2.4 to operate point-wise on the sample vectors:

1. Let $\boldsymbol{\mu}$ be the quaternion mean of the example data $Q = \{\mathbf{q}_i\}$, and let $\hat{\mu}_j$ be the jth element of $\boldsymbol{\mu}$.
2. For each example vector $\mathbf{q}_i \in Q$:
   (a) For $j = 1$ to n, where n is the length of $\mathbf{q}_i$:
      i. Let $\hat{q}_{ij}$ be the jth element of $\mathbf{q}_i$.
      ii. If $\hat{\mu}_j \cdot \hat{q}_{ij} < 0$, then let the jth element of $\mathbf{q}_{ij}$ be $-\hat{q}_{ij}$.

Estimating the Covariance

As with single unit quaternions, once the quaternion mean vector has been estimated and the examples have been hemispherized according to it, estimating the covariance is straightforward.

The estimated covariance matrix $\mathbf{\Sigma}$ is defined, as before, to be the normalized outer product of the data matrix. However, the data matrix will consists of vectors of data quaternions that have each first been hemispherized, then rotated by the quaternion mean, and finally projected into the tangent space at the mean to be "linearized", using the logmap.

The covariance estimation algorithm is summarized as follows:

1. Let $\boldsymbol{\mu}$ be the quaternion mean of the example data $Q = \{\mathbf{q}_i\}$, and let $\hat{\mu}_j$ be the jth element of $\boldsymbol{\mu}$.
2. Perform point-wise hemispherization on the example data vectors, and then project them into the tangent space at the mean. For each example $\mathbf{q}_i \in Q$:
   (a) For $j = 1$ to n, where n is the length of $\mathbf{q}_i$:
      i. Let $\hat{q}_{ij}$ be the jth element of $\mathbf{q}_i$.
      ii. If $\hat{\mu}_j \cdot \hat{q}_{ij} < 0$, then $\hat{q}_{ij} \leftarrow -\hat{q}_{ij}$.
      iii. Let $\mathbf{m}_{ij} = \ln(\hat{\mu}_j^* \hat{q}_{ij})$.
   (b) Let $\mathbf{m}_i$ be the vector of length $3n$, where the jth set of three elements is $\mathbf{m}_{ij}$.
3. Let $\mathbf{M}$ be the $3n \times N$ data matrix with the $\mathbf{m}_i$ as its ith column.
4. Let the estimated covariance matrix be calculated as $\mathbf{\Sigma} = \frac{1}{N-1}\mathbf{M}\mathbf{M}^T$, where N is the number of data points.

Note that, after hemispherization, we essentially apply the mode-tangent operation to the vector of unit quaternions.

3.3.2 Estimating Multivariate Gaussians over Object Parts

In this section we discuss how to estimate the maximum likelihood parameters $\boldsymbol{\mu}$ and $\mathbf{\Sigma}$ of a Gaussian distribution over $\mathbb{B}^*$ from a set of sample vectors.

Estimating the Mean

Say that we have a set of sample vectors $X = \{\mathbf{x}_i\}$ in $\mathbb{B}^*$, where $N = |X|$. Each vector $\mathbf{x}_i$ is of length $m + r + t$, and has the form:

$$\mathbf{x}_i = \begin{bmatrix} \mathbf{x}_{id} \\ \mathbf{x}_{ip} \\ \mathbf{x}_{iq} \end{bmatrix} .$$

To estimate the mean vector $\boldsymbol{\mu}$ of X, we estimate the mean of each section of the vectors independently, and then concatenate the resulting vectors together. (Equivalently, we can think of this as computing the mean of each component of the vectors individually, where a quaternion is a single component.) Let X_d, X_p, and X_q be the sets of subvectors of magnitudes $\{\mathbf{x}_{id}\}$, reals $\{\mathbf{x}_{ip}\}$, and unit quaternions $\{\mathbf{x}_{iq}\}$, respectively.

Estimating the mean of X_p is trivial; it is simply the sample mean:

$$\boldsymbol{\mu}_p = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_{ip} \quad . \tag{3.3}$$

To estimate the mean of X_d, we first take the point-wise log of the vectors to map them into $\mathbb{R}^n$, and then take the sample mean as usual:

$$\boldsymbol{\mu}_d = \frac{1}{N}\sum_{i=1}^{N} \log(\mathbf{x}_{id}) \quad . \tag{3.4}$$

And of course, the mean of X_q is the quaternion vector mean, calculated as in the previous section.

Thus we can summarize the mean estimation algorithm as:

1. Let $X = \{\mathbf{x}_i\}$ be a set of sample vectors in $\mathbb{B}^*$, where the size of X is N and each vector $\mathbf{x}_i$ has the form $\begin{bmatrix}\mathbf{x}_{id} & \mathbf{x}_{ip} & \mathbf{x}_{iq}\end{bmatrix}^T$.
2. Let $\boldsymbol{\mu}_d$ be:
$$\boldsymbol{\mu}_d = \frac{1}{N}\sum_{i=1}^{N} \log(\mathbf{x}_{id}) \quad .$$
where $\log(\mathbf{x}_{id})$ is the vector of logs of the components of $\mathbf{x}_{id}$.
3. Let $\boldsymbol{\mu}_p$ be:
$$\boldsymbol{\mu}_p = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_{ip} \quad .$$
4. For $j = 1$ to t, where t is the length of each $\mathbf{x}_{iq}$:
   (a) Let $\mathbf{Q}_j$ be the $4 \times N$ data matrix where the ith column is the vector representation of the jth element of the quaternion subvector $\mathbf{x}_{iq}$.
   (b) Let $\mathbf{A}_j = \mathbf{Q}_j\mathbf{Q}_j^T$.
   (c) Choose one of the two eigenvectors $\pm\mathbf{e}_{j*}$ associated with the maximal eigenvalue a_{j*} of $\mathbf{A}_j$ as the estimate of the mean $\hat{\mu}_j$.
5. Let $\boldsymbol{\mu}_q$ be a vector of length t, where the jth element is $\hat{\mu}_j$.
6. Let $\boldsymbol{\mu}$ be $\begin{bmatrix}\boldsymbol{\mu}_d & \boldsymbol{\mu}_p & \boldsymbol{\mu}_q\end{bmatrix}^T$.

Estimating the Covariance

Before we can compute the covariance from the sample data X, we must of course hemispherize the quaternion elements of the sample vectors, as shown in the previous section. Once that is done, however, we can simply calculate the extended mode-tangent vector of each sample, as shown in Section 2.3.2:

$$\mathbf{m} = \begin{bmatrix} \log(\mathbf{x}_d/\boldsymbol{\mu}_d) \\ \mathbf{x}_p - \boldsymbol{\mu}_p \\ \ln(\boldsymbol{\mu}_q^* \mathbf{x}_q) \end{bmatrix}$$

for each vector $\mathbf{x} \in X$. Then, we take the inner product of a data matrix containing the mode-tangent vectors as its columns.

The covariance estimation algorithm can then be written as:

1. Let $X = \{\mathbf{x}_i\}$ be a set of sample vectors in $\mathbb{B}^*$, where each vector $\mathbf{x}_i$ has the form $\begin{bmatrix}\mathbf{x}_{id} & \mathbf{x}_{ip} & \mathbf{x}_{iq}\end{bmatrix}^T$. Let m, r, and t be the lengths of $\mathbf{x}_{id}$, $\mathbf{x}_{ip}$, and $\mathbf{x}_{iq}$, respectively.
2. Let $\boldsymbol{\mu} = \begin{bmatrix}\boldsymbol{\mu}_d & \boldsymbol{\mu}_p & \boldsymbol{\mu}_q\end{bmatrix}^T$ be the mean vector of the example data X.
3. Perform point-wise hemispherization on the quaternion elements of the example data vectors. For each $\mathbf{x}_{iq}$ in X, and for $j = 1$ to t:
   (a) Let $\hat{\mu}_j$ be the jth element of $\boldsymbol{\mu}_q$.
   (b) Let $\hat{q}_{ij}$ be the jth element of $\mathbf{x}_{iq}$.
   (c) If $\hat{\mu}_j \cdot \hat{q}_{ij} < 0$, then $\hat{q}_{ij} \leftarrow -\hat{q}_{ij}$.
4. Find the mode-tangent vector $\mathbf{m}_i$ of each example $\mathbf{x}_i$:
$$\mathbf{m}_i = \begin{bmatrix} \log(\mathbf{x}_{id}/\boldsymbol{\mu}_d) \\ \mathbf{x}_{ip} - \boldsymbol{\mu}_{ip} \\ \ln(\boldsymbol{\mu}_q^* \mathbf{x}_{iq}) \end{bmatrix} .$$
5. Let $\mathbf{M}$ be the $(m + r + 3t) \times N$ data matrix with $\mathbf{m}_i$ as its ith column.
6. Let the estimated covariance matrix be calculated as $\boldsymbol{\Sigma} = \frac{1}{N-1}\mathbf{M}\mathbf{M}^T$, where N is the number of data points.

3.3.3 Estimating Conditional Multivariate Gaussians over Object Parts

Estimating a conditional multivariate Gaussian over $\mathbb{B}^*$ is quite straightforward once we have the algorithms we presented above. Given pairs of vectors $\langle \mathbf{x}_{i1}, \mathbf{x}_{i2} \rangle$, we simply learn a joint Gaussian for $p(\mathbf{x}_1, \mathbf{x}_2)$ and then factor the resulting distribution using Equation (2.14).

The algorithm is:

1. Let $X = \{\langle \mathbf{x}_{i1}, \mathbf{x}_{i2} \rangle\}$ be a set of pairs of sample vectors, where each pair has the form:
$$\langle \mathbf{x}_{i1}, \mathbf{x}_{i2} \rangle = \left\langle \begin{bmatrix} \mathbf{x}_{id1} \\ \mathbf{x}_{ip1} \\ \mathbf{x}_{iq1} \end{bmatrix}, \begin{bmatrix} \mathbf{x}_{id2} \\ \mathbf{x}_{ip2} \\ \mathbf{x}_{iq2} \end{bmatrix} \right\rangle .$$

2. For each pair of vectors in X, construct a joint vector $\mathbf{x}_i$ with the form:
$$\mathbf{x}_i = \begin{bmatrix} \mathbf{x}_{id1} \\ \mathbf{x}_{id2} \\ \mathbf{x}_{ip1} \\ \mathbf{x}_{ip2} \\ \mathbf{x}_{iq1} \\ \mathbf{x}_{iq2} \end{bmatrix} .$$

3. Learn the parameters $\langle \boldsymbol{\mu}, \boldsymbol{\Sigma} \rangle$ of a joint Gaussian over the vectors $\{\mathbf{x}_i\}$, where:
$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_{d1} \\ \boldsymbol{\mu}_{d2} \\ \boldsymbol{\mu}_{p1} \\ \boldsymbol{\mu}_{p2} \\ \boldsymbol{\mu}_{q1} \\ \boldsymbol{\mu}_{q2} \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{d1d1} & \boldsymbol{\Sigma}_{d1d2} & \boldsymbol{\Sigma}_{d1p1} & \boldsymbol{\Sigma}_{d1p2} & \boldsymbol{\Sigma}_{d1q1} & \boldsymbol{\Sigma}_{d1q2} \\ \boldsymbol{\Sigma}_{d2d1} & \boldsymbol{\Sigma}_{d2d2} & \boldsymbol{\Sigma}_{d2p1} & \boldsymbol{\Sigma}_{d2p2} & \boldsymbol{\Sigma}_{d2q1} & \boldsymbol{\Sigma}_{d2q2} \\ \boldsymbol{\Sigma}_{p1d1} & \boldsymbol{\Sigma}_{p1d2} & \boldsymbol{\Sigma}_{p1p1} & \boldsymbol{\Sigma}_{p1p2} & \boldsymbol{\Sigma}_{p1q1} & \boldsymbol{\Sigma}_{p1q2} \\ \boldsymbol{\Sigma}_{p2d1} & \boldsymbol{\Sigma}_{p2d2} & \boldsymbol{\Sigma}_{p2p1} & \boldsymbol{\Sigma}_{p2p2} & \boldsymbol{\Sigma}_{p2q1} & \boldsymbol{\Sigma}_{p2q2} \\ \boldsymbol{\Sigma}_{q1d1} & \boldsymbol{\Sigma}_{q1d2} & \boldsymbol{\Sigma}_{q1p1} & \boldsymbol{\Sigma}_{q1p2} & \boldsymbol{\Sigma}_{q1q1} & \boldsymbol{\Sigma}_{q1q2} \\ \boldsymbol{\Sigma}_{q2d1} & \boldsymbol{\Sigma}_{q2d2} & \boldsymbol{\Sigma}_{q2p1} & \boldsymbol{\Sigma}_{q2p2} & \boldsymbol{\Sigma}_{q2q1} & \boldsymbol{\Sigma}_{q2q2} \end{bmatrix} .$$

4. Partition and extract the parameters of the joint to parameterize a conditional multivariate Gaussian as $\langle \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{11}, \boldsymbol{\Sigma}_{12}, \boldsymbol{\Sigma}_{21}, \boldsymbol{\Sigma}_{22} \rangle$, where:
$$\boldsymbol{\mu}_1 = \begin{bmatrix} \boldsymbol{\mu}_{d1} \\ \boldsymbol{\mu}_{p1} \\ \boldsymbol{\mu}_{q1} \end{bmatrix} \quad \boldsymbol{\Sigma}_{11} = \begin{bmatrix} \boldsymbol{\Sigma}_{d1d1} & \boldsymbol{\Sigma}_{d1p1} & \boldsymbol{\Sigma}_{d1q1} \\ \boldsymbol{\Sigma}_{p1d1} & \boldsymbol{\Sigma}_{p1p1} & \boldsymbol{\Sigma}_{p1q1} \\ \boldsymbol{\Sigma}_{q1d1} & \boldsymbol{\Sigma}_{q1p1} & \boldsymbol{\Sigma}_{q1q1} \end{bmatrix} \quad \boldsymbol{\Sigma}_{12} = \begin{bmatrix} \boldsymbol{\Sigma}_{d1d2} & \boldsymbol{\Sigma}_{d1p2} & \boldsymbol{\Sigma}_{d1q2} \\ \boldsymbol{\Sigma}_{p1d2} & \boldsymbol{\Sigma}_{p1p2} & \boldsymbol{\Sigma}_{p1q2} \\ \boldsymbol{\Sigma}_{q1d2} & \boldsymbol{\Sigma}_{q1p2} & \boldsymbol{\Sigma}_{q1q2} \end{bmatrix}$$
$$\boldsymbol{\mu}_2 = \begin{bmatrix} \boldsymbol{\mu}_{d2} \\ \boldsymbol{\mu}_{p2} \\ \boldsymbol{\mu}_{q2} \end{bmatrix} \quad \boldsymbol{\Sigma}_{21} = \begin{bmatrix} \boldsymbol{\Sigma}_{d2d1} & \boldsymbol{\Sigma}_{d2p1} & \boldsymbol{\Sigma}_{d2q1} \\ \boldsymbol{\Sigma}_{p2d1} & \boldsymbol{\Sigma}_{p2p1} & \boldsymbol{\Sigma}_{p2q1} \\ \boldsymbol{\Sigma}_{q2d1} & \boldsymbol{\Sigma}_{q2p1} & \boldsymbol{\Sigma}_{q2q1} \end{bmatrix} \quad \boldsymbol{\Sigma}_{22} = \begin{bmatrix} \boldsymbol{\Sigma}_{d2d2} & \boldsymbol{\Sigma}_{d2p2} & \boldsymbol{\Sigma}_{d2q2} \\ \boldsymbol{\Sigma}_{p2d2} & \boldsymbol{\Sigma}_{p2p2} & \boldsymbol{\Sigma}_{p2q2} \\ \boldsymbol{\Sigma}_{q2d2} & \boldsymbol{\Sigma}_{q2p2} & \boldsymbol{\Sigma}_{q2q2} \end{bmatrix} .$$

3.3.4 An Algorithm For Learning Geometric Models From Examples

Armed with the above algorithms, we can write a procedure for learning the root and part geometric models of a PGG G from a set of parsed and labeled instances.

The intuitive approach is similar to the algorithm in Section 3.2. We use each tree fragment $\hat{a}^{\ell}_{gh}$ in each tree to collect example parent-child data pairs for learning the part geometric models of the rule in G that matches $\hat{a}^{\ell}_{gh}$. We also learn the root geometric models from the geometric characteristics of each individual node in each example tree.

We can summarize this algorithm as:

1. Let G be a PGG.
2. Let $T = \{t\}$ be a set of parsed examples that were labeled according to G.
3. Initialize training sets. For each class $C^i \in G$:
   (a) Let $\Psi_i = \{\}$.
   (b) For each rule part s_c^{ijk} on the RHS of each rule r^{ij} in C^i, let $\Omega_{ijk} = \{\}$.
4. Collect data vectors. For each tree $t \in T$:
   (a) For each node $a_\ell \in t$:
      i. Let $C^i = C_\ell$.
      ii. Add $\mathbf{b}_\ell$ to Ψ_i.
   (b) For each tree fragment $\hat{a}^{\ell}_{gh} \in t$:
      i. Let $r^{ij} = \text{matching}(\hat{a}^{\ell}_{gh})$.
      ii. For $k = 1$ to n, where $n = |\hat{a}^{\ell}_{gh}|$:
         A. Let s_c^{ijk} be the kth rule part on the RHS of r^{ij}.
         B. Let a_k be the kth child node of $\hat{a}^{\ell}_{gh}$.
         C. Add the pair $\langle \mathbf{b}_\ell, \mathbf{b}_k \rangle$ to Ω_{ijk}.
5. Learn geometric models from data. For each class $C^i \in G$:
   (a) If $\Psi_i \neq \emptyset$:
      i. Let $\langle \boldsymbol{\mu}, \boldsymbol{\Sigma} \rangle$ be the parameters of a multivariate Gaussian over $\mathbb{B}$, estimated from the example vectors in Ψ_i.
      ii. Let $\rho^i = \langle \boldsymbol{\mu}, \boldsymbol{\Sigma} \rangle$.
   (b) For each rule r^{ij} in C^i:
      i. For each rule part s_c^{ijk} on the RHS of r^{ij}:
         A. If $\Omega_{ijk} \neq \emptyset$:
            - Let $\langle \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{11}, \boldsymbol{\Sigma}_{12}, \boldsymbol{\Sigma}_{21}, \boldsymbol{\Sigma}_{22} \rangle$ be the parameters of a conditional multivariate Gaussian over $\mathbb{B}$, estimated from the example pairs in Ω_{ijk}.
            - Let $\phi^{ijk} = \langle \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{11}, \boldsymbol{\Sigma}_{12}, \boldsymbol{\Sigma}_{21}, \boldsymbol{\Sigma}_{22} \rangle$.

3.4 Training Data

In this thesis, we constrain the form of the training data to segmented synthetic 3D objects; the form of the data is discussed further in Chapter 5.

In the future, however, the learning algorithms in this chapter might be trained on "real" data; for example, segmented or unsegmented CAD models, 3D object scan examples, or 3D models inferred from range data or complex 3D reconstruction-from-video algorithms.

Then, as the system recognizes new instances, these examples would be added to the training set in order to refine the rule and geometric probability models. Ideally, we would modify the above algorithms to operate incrementally, so that the set of all training examples need not be maintained indefinitely.

Chapter 4

Parsing in the PGG Framework

In Section A.1.2, we present the inside algorithm for PCFGs, which uses dynamic programming to efficiently calculate the probability or find the most likely parse tree of a string. In this chapter, we will modify the inside algorithm to work on unlabeled, unordered sets of object parts, rather than strings of words.

The chapter is organized as follows:

Section 4.1 describes some issues related to parsing 3D objects with geometric characteristics.

Section 4.2 presents the modified inside algorithm for calculating the inside density and most likely parse of a set of 3D object parts, according to a PGG.

Section 4.3 discusses the exponential complexity of the inside algorithm for 3D objects, and the resulting efficiency concerns.

4.1 Geometric Parsing

Given a set of unlabeled primitive 3D object parts $\{\mathbf{b}_1, \ldots, \mathbf{b}_M\}$ and a PGG G, we would like to find:

- the likelihood of $\{\mathbf{b}_1, \ldots, \mathbf{b}_M\}$ given G; and
- the most likely parse tree that yields $\{\mathbf{b}_1, \ldots, \mathbf{b}_M\}$ in some order, given G.

Before we plunge into a discussion of the algorithms themselves, we shall discuss some issues related to parsing 3D objects (rather than strings in a language), and develop some notation.

4.1.1 The Unordered Nature of 3D Space

The inside algorithm for PCFGs is worthy of imitation, because it efficiently solves an exponential problem in polynomial time through its use of dynamic programming. In adapting a parsing algorithm designed to work on strings in a language to work for object recognition, however, we must consider one of the most important differences between strings and objects.

In language processing, the input to a parsing algorithm is a one-dimensional string of words, and assuming reasonably accurate text, there is little ambiguity about the appropriate order in which to consider the words. In object recognition, however, there is no inherent ordering on the parts of an object in 3D space; i.e., there is no "right" way to match the four legs of a chair to the four "leg" parts on the right side of a rule.

This fundamental difference means that a parsing algorithm for objects must consider all possible assignments of object parts to rule parts, (although, some assignments will be more likely than others, due to geometric constraints).We shall soon see that this is one of the major necessary modifications to the inside algorithm for object recognition.

4.1.2 Chomsky Normal Form for PGGs

The derivation of the inside algorithm for PCFGs in Section A.1.2 depends on the PCFG being in Chomsky Normal Form; i.e., that all rules in the grammar are of one of two forms:

$$N^i \mapsto N^j N^k \qquad\qquad \text{or} \qquad\qquad N^i \mapsto w^j \quad .$$

This assumption greatly simplifies the parsing task, because it means that any single application of a rule produces exactly two subtrees; therefore the parsing algorithm only needs to search over a single split point in the string.

We could make a similar assumption, and require that any PGG be converted into Chomsky Normal Form before parsing. However, although the resulting PGG would produce an equivalent "structural language" to the original PGG—one which can generate sets of primitive boxes with the same class labels—there is no guarantee that the geometric distributions produced by the two PGGs would be equivalent. We might expect that the converted PGG would have a more limited ability to express geometric relationships between parts, because its parse trees would be "taller" in general, rather than "wider", and we have seen that there are conditional independence assumptions between generations of the trees produced by the grammar.

On the other hand, requiring PGGs to have some limit on the number of rule parts in each rule might be helpful, from the standpoint of the complexity of the algorithm. This is because of the summarization function that is necessary for parsing with a geometric grammar. By definition, summarization of a set of object parts requires the identities of the component parts to be fixed. Thus, there is no way to compute the geometric likelihood of a single part given its parent without committing to the assignments of the other sibling parts in the tree fragment; e.g. the choice of all four legs of a chair must be fixed before the geometric score of any individual leg (as well as the entire chair base) can be determined. This aspect of the model might make attempts to incorporate aggressive pruning into the parsing process more difficult, or at least less effective; some general form of a Chomsky Normal Form requirement might allow better pruning.

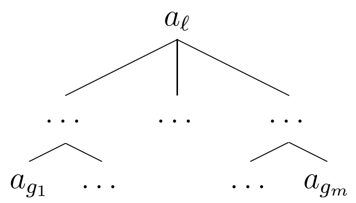
However, we have not formally or experimentally verified these speculations, so they are an avenue for future work. Thus, for the purposes of this thesis, we must further modify the inside algorithm to allow for the non-CNF form of the PGGs.

4.1.3 Subtree Notation

Before we continue discussing parsing in a geometric grammar, let us define some notation.

We define the ℓth *subtree* in a tree t, denoted $\bar{a}^{\ell}_{\{g_1,\ldots,g_m\}}$, to be a node $a_\ell \in t$, and the tree rooted at it:

$\bar{a}^{\ell}_{\{g_1,\ldots,g_m\}}$:



.

where $\mathrm{leaves}(a^{\ell}_{\{g_1,\ldots,g_m\}}) = \{a_{g_1}, \ldots, a_{g_m}\}$ in any order.[1]

Let $|\bar{a}^{\ell}_{\{g_1,\ldots,g_m\}}|$ denote the number of leaves of $\bar{a}^{\ell}_{\{g_1,\ldots,g_m\}}$, so we have:

$$|\bar{a}^{\ell}_{\{g_1,\ldots,g_m\}}| = |\mathrm{leaves}(\bar{a}^{\ell}_{\{g_1,\ldots,g_m\}})| = |\{a_{g_1}, \ldots, a_{g_m}\}| = m \ .$$

As a notational convenience, let $\mathbf{b}_{\{g_1\ldots,g_m\}}$ denote the unordered set of boxes $\{\mathbf{b}_{g_1}, \ldots, \mathbf{b}_{g_m}\}$. Similarly, if I is a set of integers, let $\mathbf{b}_I$ be the unordered set of boxes indexed by the members of I.

As another notational convenience, we shall extend the subtree notation to the vectors of geometric characteristics of all nodes in the tree, so that $\mathrm{leaves}(\bar{\mathbf{b}}^{\ell}_{\{g_1,\ldots,g_m\}})$ refers to the set of vectors of geometric characteristics of the leaves of the subtree rooted at the node a_ℓ:

$$\mathrm{leaves}(\bar{\mathbf{b}}^{\ell}_{\{g_1,\ldots,g_m\}}) = \mathbf{b}_{\{g_1\ldots,g_m\}} = \{\mathbf{b}_{g_1}, \ldots, \mathbf{b}_{g_m}\} \ .$$

Finally, we let $C^{i}_{\{g_1,\ldots,g_m\}}$ denote a subtree $\bar{a}^{\ell}_{\{g_1,\ldots,g_m\}}$ for which $C_\ell = C^i$. This lets us index subtrees using the index i of the class C^i in the PGG rather than the index ℓ of the node a_ℓ in the tree. (This notation also corresponds most closely to the subtree notation N^i_{pq} used in standard PCFGs, as described in Section A.1.2, as the internal nodes in PCFG parse trees are simply nonterminal symbols.)

[1]Note the distinction between a *tree fragment*, defined in Section 3.1.1 and denoted $\hat{a}^{\ell}_{gh}$ and a *subtree* as defined here, which is denoted $\bar{a}^{\ell}_{\{g_1,\ldots,g_m\}}$. A tree fragment consists of an internal node and its children only, while a subtree is defined for every node in the tree and consists of the node and all its descendents (possibly none). Furthermore, we made a simplifying assumption in our definition of tree fragments that the nodes of the tree are indexed according to a breadth-first traversal, but there is no such assumption here—the indices of the leaf nodes are explicitly included in the unordered set $\{g_1, \ldots, g_m\}$.

4.1.4 The Bounding Box: An Approximate Maximum Likelihood

The inside algorithm operates in a bottom-up manner. In the adapted version for PGGs, therefore, an important step will be the construction of new internal nodes from partially constructed subtrees.

In Section 2.6, we showed how to calculate the geometric likelihood of a parse tree t. Consider, for the moment, a single tree fragment of t:

$$\begin{array}{ccc} & a_\ell & \\ / & | & \backslash \\ a_{g_1} & \ldots & a_{g_n} \end{array}$$

where the nodes $a_{g_1} \ldots a_{g_n}$ may be primitive or composite.

During parsing, we need to actually *build* a tree of this form, rather than simply calculate its likelihood. Thus, we need to construct the new internal node a_ℓ, where $C_\ell = C^i$, that covers the nodes $a_{g_1}, \ldots, a_{g_n}$:

$$C^i_{\{g_1,\ldots,g_n\}} \ .$$

We must consider each rule r^{ij} of the form

$$C^i = s^{ij1}_{c_1}, \ldots s^{ijn}_{c_n}$$

in the PGG G. We assume for simplicity in this section that the number of children in the subtree and the number of rule parts on the right side of r^{ij} are the same, and that we are only considering a single one-to-one assignment of object parts to rule parts.

In order to sum over or choose between rules, we need to calculate the geometric likelihood of the children of subtree according to the assignment to rule parts in r^{ij}:

$$\mathrm{p}(\mathbf{b}_{g_1}, \ldots, \mathbf{b}_{g_n} \,|\, G) \ .$$

And, to calculate this quantity, we must know the value of the geometric characteristics of the new composite part a_ℓ that we want to construct:

$$\mathrm{p}(\mathbf{b}_{g_1}, \ldots, \mathbf{b}_{g_n}, \mathbf{b}_\ell \,|\, G) = \mathrm{p}(\mathbf{b}_{g_1}, \ldots, \mathbf{b}_{g_n} \,|\, \mathbf{b}_\ell, G)\, \mathrm{p}(\mathbf{b}_\ell \,|\, G)$$

(where the equality is due to the chain rule of probability). In parsing, however, we are given only primitive parts to begin, so where do the composite parts, such as $\mathbf{b}_\ell$, come from?

In fact, we never see direct evidence of composite parts in real world instances—the concept of a "chair base" is, in some sense, an imaginary construct of human language and thought. The vector of geometric characteristics $\mathbf{b}_\ell$ for each composite part a_ℓ is actually a hidden random variable in the model. This random variable can take on an arbitrary vector in $\mathbb{B}$ as a value, so we should "sum over" it—i.e., sum the values of $\mathrm{p}(\mathbf{b}_{g_1}, \ldots, \mathbf{b}_{g_n}, \mathbf{b}_\ell \,|\, G)$ for all possible values of the root composite part $\mathbf{b}_\ell$.

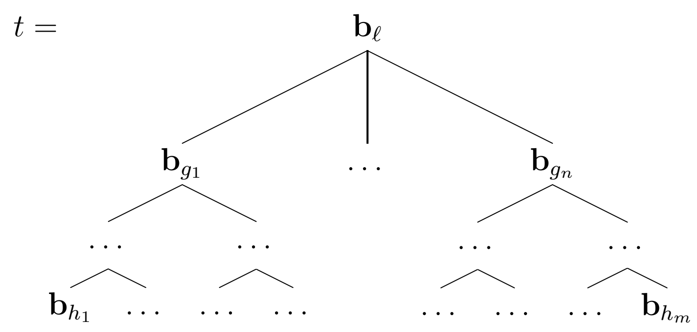


Figure 4-1: If $B = \{\mathbf{b}_{g_1}, \ldots, \mathbf{b}_{g_n}\}$, then $\square(B)$ is defined to be the minimum volume bounding box of the parts at the leaves of the tree, $\text{leaves}(t) = \{\mathbf{b}_{h_1}, \ldots, \mathbf{b}_{h_m}\}$, rather than the actual parts in B.

Then we can make a standard assumption that the resulting distribution over values of $\mathbf{b}_\ell$ is quite peaked, so therefore it is a sufficient approximation to simply take the max rather than the sum. We can write these two steps as:

$$\begin{aligned} \mathrm{p}(\mathbf{b}_{g_1}, \ldots, \mathbf{b}_{g_n} \,|\, G) &= \sum_{\mathbf{b}_\ell \in \mathbb{B}} \mathrm{p}(\mathbf{b}_{g_1}, \ldots, \mathbf{b}_{g_n} \mathbf{b}_\ell \,|\, G) \\ &\approx \max_{\mathbf{b}_\ell \in \mathbb{B}} \mathrm{p}(\mathbf{b}_{g_1}, \ldots, \mathbf{b}_{g_n} \mathbf{b}_\ell \,|\, G) \quad . \end{aligned}$$

This still seems impossible to calculate, though—regardless of whether we take the max or the sum we must still consider all values of $\mathbf{b}_\ell$ in $\mathbb{B}$.

This is where summarization comes in. As we discussed in Section 2.2, we expect any composite part represented by the left hand side variable of a rule to have geometric characteristics that "summarize" the geometric characteristics of all the parts represented by the right hand side variables. So, it is reasonable to assume that the maximum likelihood value for $\mathbf{b}_\ell$ might be one which summarizes the component parts $\{\mathbf{b}_{g_1}, \ldots, \mathbf{b}_{g_n}\}$. In particular, the minimum volume bounding box of the component parts seems like a good approximation to the maximum likelihood value of $\mathbf{b}_\ell$.

This is the gist of the method for constructing composite parts in the inside algorithm for PGGs. However, because of the dynamic programming assumption, we must ensure that the value of a composite part is identical for a fixed set of primitive parts *regardless of the internal tree structure*. And, since we are using an approximation algorithm to compute the minimum volume bounding box (as described in Section A.4, the easiest way to ensure this is to construct each composite part as the bounding box of all the *leaves* of the subtree rooted at that composite part. Thus, if $B = \{\mathbf{b}_{g_1}, \ldots, \mathbf{b}_{g_n}\}$ is a set of primitive or composite parts, we let $\square(B)$ denote the minimum volume bounding box of the *primitive* 3D boxes that constitute the parts in B. For a visual example, see Figure 4-1.

Thus we have:

$$
\begin{aligned}
\mathrm{p}(\mathbf{b}_{g_1}, \ldots, \mathbf{b}_{g_n} \,|\, G) &= \sum_{\mathbf{b}_\ell \in \mathbb{B}} \mathrm{p}(\mathbf{b}_{g_1}, \ldots, \mathbf{b}_{g_n} \mathbf{b}_\ell \,|\, G) \\
&\approx \max_{\mathbf{b}_\ell \in \mathbb{B}} \mathrm{p}(\mathbf{b}_{g_1}, \ldots, \mathbf{b}_{g_n} \mathbf{b}_\ell \,|\, G) \\
&\approx \mathrm{p}(\mathbf{b}_{g_1}, \ldots, \mathbf{b}_{g_n}, \Box(\mathbf{b}_{\{g_1,\ldots,g_n\}} \,|\, G) \\
&= \mathrm{p}(\mathbf{b}_{g_1}, \ldots, \mathbf{b}_{g_n} \,|\, \Box(\mathbf{b}_{\{g_1,\ldots,g_n\}}, G) \, \mathrm{p}(\Box(\mathbf{b}_{\{g_1,\ldots,g_n\}} \,|\, G) \ .
\end{aligned}
\tag{4.1}
$$

Using a summarization function is a reasonable approach in practice as well. This is precisely because composite parts are hidden variables, and therefore never have measurable values in the world, so we must estimate their values even when gathering and labeling training data. Hence, if we make an effort to train the PGG using data that has been labeled with a summarization function similar to the one used by the parsing algorithm, the summarization function will actually produce values very close or identical to the maximum likelihood values for composite parts.

4.1.5 A Penalty For Clutter Parts

One of the inherent weaknesses of PCFGs is their strong bias towards smaller parse trees. This bias exists because each internal node in a parse tree introduces a new expansion probability to the expression for the likelihood of the tree, and expansion probabilities are discrete and guaranteed to be between zero and one. In a PCFG parse tree, the result is that a parse tree with more internal nodes will *always* have a lower likelihood than a tree with fewer internal nodes—even in cases in which the larger tree is actually the "correct" answer.

PGGs inherit this problem from PCFGs. Consider again the expression for the likelihood of a parse tree in Equation (2.18). The γ_ℓ terms multiply to produce a product that is always smaller for a greater number of internal nodes a_ℓ in t.[2] Unless we compensate for this weakness in the model, the bias towards smaller trees could have undesirable consequences. For example, the parsing process might always prefer to ignore the arms of a chair, if the grammar says they are optional, in order to avoid the additional probabilistic terms that would be accrued by including the arms in a parse tree.

At the root of this problem is the fact that the likelihoods of two probabilistic expressions can only be compared if they cover the same random variables. This means that it is incorrect to compare the likelihoods of two parse trees unless the trees cover the same set of primitive parts. If we are choosing between two interpretations

[2]However, it is not actually the case that the entire likelihood of the parse tree is necessarily smaller for a greater number of internal nodes. This is because the geometric scores of nodes are continuous probability *densities*, and therefore can actually be arbitrary positive values, possibly much greater than one. This may seem quite unintuitive. To understand how a Gaussian distribution can produce densities that are greater than one, consider a simple univariate "bell curve". A Gaussian is normalized such that the volume under the curve integrates to one, but this means that, if the distribution is quite peaked and the average "width" of the bell curve is much less than one, the average "height" of the curve could be much greater than one.

of a set of parts, we must account for, and provide labels for, *all* of the parts in both interpretations—even if some of the labels are actually "clutter".

The ideal way to approach this problem would be to learn a geometric model for clutter boxes and have a formal clutter class in the PGG. However, this approach seems problematic because, in many cases, it is not the geometric properties of the boxes themselves that makes them clutter, but rather a global property of the scene and the other parts in the scene. If we were considering richer representations of shape, there might be parts that we know we can safely label as clutter simply on the basis of their geometric properties. With our simple box representation for parts, however, we will ignore this possibility for now.

Instead, a reasonable approach might be to propose a low "uniform" distribution over the space of boxes for the clutter model, so that the parsing process would prefer to assign boxes to real model parts when it can, but it cannot simply ignore them altogether. In practice, this is roughly equivalent to a applying a constant "penalty" for each box we label as clutter. Thus, we introduce a constant clutter penalty κ to a PGG G, which shall represent the likelihood of a single unassigned clutter part. In our implementations, we choose a fixed value for κ which works well empirically, but theoretically we could use more formal learning methods. We shall discuss how the clutter penalty is applied in the next section.

4.2 The Inside Algorithm for PGGs

Now, we are ready to present our modified version of the inside algorithm for PGGs. First, we consider how to calculate the likelihood of a set of primitive object parts, and then how to find the maximum likelihood parse tree of the primitive parts.

4.2.1 Calculating the Likelihood of a Set of Object Parts

Using All Object Parts

The *inside likelihood* $\beta_i(\{g_1, \ldots, g_m\})$ is the total likelihood of generating the set of primitive object parts $\{\mathbf{b}_{g_1}, \ldots, \mathbf{b}_{g_m}\}$ in some order, given that we begin with a root class C^i and that all m boxes are used. We define the inside likelihood β of a scene of boxes $\{\mathbf{b}_1, \ldots, \mathbf{b}_M\}$ as:

$$\begin{aligned}
\mathrm{p}(\mathbf{b}_{\{1,\ldots,M\}} \,|\, G) &= \mathrm{p}(C^1 \stackrel{*}{\Rightarrow} \mathbf{b}_{\{1,\ldots,M\}}, G) \\
&= \mathrm{p}(\mathbf{b}_{\{1,\ldots,M\}} \,|\, C^1_{\{1,\ldots,M\}}, G) \\
&= \beta_1(\{1, \ldots, M\}) \ .
\end{aligned}$$

Now, we calculate the inside likelihood by induction on the set of covered boxes:

Base case: We want to find $\beta_i(\{g\})$. This is simply one if the class C^i is primitive, and zero otherwise, since in our model, any primitive unlabeled part might be an instance of any primitive class:

$$\beta_i\{g\} = \mathrm{p}(\{\mathbf{b}_g\} \mid C^i_{\{g\}}, G)$$
$$= \begin{cases} \mathrm{p}(\mathbf{b}_g \mid \rho^i) & \text{if } i = 1 \text{ and } C^i \text{ is primitive} \\ 1 & \text{if } i \neq 1 \text{ and } C^i \text{ is primitive} \\ 0 & \text{otherwise.} \end{cases}$$

Induction: We want to find $\beta_i(\{g_1, \ldots, g_m\})$. We do not constrain our grammars to be in Chomsky Normal Form, so we must consider rules of the general form

$$C^i \mapsto s^{ij1}_{c_1} \ldots s^{ijn}_{c_n} \quad .$$

for $n \leq m$. In addition, we must consider all assignments of the primitive parts to leaves of the subtrees rooted at the parts of the rule, so we must search over all sets of n partitions of the set of m primitive parts, and all assignments of the sets in each partition to the rule parts.

Let $\mathrm{parts}(\{g_1, \ldots, g_m\}, n)$ denote the set of all partitions of the set of indices $\{g_1, \ldots, g_m\}$ into n subsets, so that each member $\{I_1, \ldots, I_n\}$ is a set of mutually exclusive yet collectively exhaustive subsets of the original set. Also, let $\mathrm{perms}(\{1, \ldots, n\})$ be the set of all permutations of the set of integers $\{1, \ldots, n\}$, such that each member f maps an integer between 1 and n to its new position between 1 and n. Then we have:

$$\begin{aligned} \beta_i(\{g_1, \ldots, g_m\}) &= \mathrm{p}(\{\mathbf{b}_{g_1}, \ldots, \mathbf{b}_{g_m}\} \mid C^i_{\{g_1,\ldots,g_m\}}, G) \\ &= \sum_{\substack{[C^i \mapsto s^{ij1}_{c_1}, \ldots, s^{ijn}_{c_n}] \in G \\ \text{for } n \leq m}} \; \sum_{\substack{\{I_1,\ldots,I_n\} \in \\ \mathrm{parts}(\{g_1,\ldots,g_m\},n)}} \; \sum_{\substack{f \in \\ \mathrm{perms}(\{1,\ldots,n\})}} \\ &\qquad \mathrm{p}\big(\mathbf{b}_{I_{f(1)}}, C^{c_1}_{I_{f(1)}}, \ldots, \mathbf{b}_{I_{f(n)}}, C^{c_n}_{I_{f(n)}} \mid C^i_{\{g_1,\ldots,g_m\}}, G\big) \quad . \end{aligned} \tag{4.2}$$

Then we apply the chain rule of probability to break up the expression:

$$\begin{aligned} \beta_i(\{g_1, \ldots, g_m\}) = &\sum_{\substack{[C^i \mapsto s^{ij1}_{c_1}, \ldots, s^{ijn}_{c_n}] \in G \\ \text{for } n \leq m}} \; \sum_{\substack{\{I_1,\ldots,I_n\} \in \\ \mathrm{parts}(\{g_1,\ldots,g_m\},n)}} \; \sum_{\substack{f \in \\ \mathrm{perms}(\{1,\ldots,n\})}} \\ &\quad \mathrm{p}\big(\mathbf{b}_{I_{f(1)}}, C^{c_1}_{I_{f(1)}}, \ldots, \mathbf{b}_{I_{f(n)}}, C^{c_n}_{I_{f(n)}} \mid C^i_{\{g_1,\ldots,g_m\}}, G\big) \\ = &\sum_{\substack{[C^i \mapsto s^{ij1}_{c_1}, \ldots, s^{ijn}_{c_n}] \in G \\ \text{for } n \leq m}} \; \sum_{\substack{\{I_1,\ldots,I_n\} \in \\ \mathrm{parts}(\{g_1,\ldots,g_m\},n)}} \; \sum_{\substack{f \in \\ \mathrm{perms}(\{1,\ldots,n\})}} \\ &\quad \mathrm{p}\big(C^{c_1}_{I_{f(1)}}, \ldots, C^{c_n}_{I_{f(n)}} \mid C^i_{\{g_1,\ldots,g_m\}}, G\big) \\ &\qquad \times \mathrm{p}\big(\mathbf{b}_{I_{f(1)}} \mid C^{c_1}_{I_{f(1)}}, \ldots, C^{c_n}_{I_{f(n)}}, C^i_{\{g_1,\ldots,g_m\}}, G\big) \\ &\qquad \times \cdots \times \mathrm{p}\big(\mathbf{b}_{I_{f(n)}} \mid \mathbf{b}_{I_{f(1)}}, \ldots, \mathbf{b}_{I_{f(n-1)}}, C^{c_1}_{I_{f(1)}}, \ldots, C^{c_n}_{I_{f(n)}}, C^i_{\{g_1,\ldots,g_m\}}, G\big) \quad . \end{aligned} \tag{4.3}$$

Now, we apply the conditional independence assumptions of the PGG framework, first to simplify the results of the chain rule by using the fact that a node is independent of all non-descendents given its parent:

$$
\begin{aligned}
\beta_i(\{g_1,\dots,g_m\}) = & \sum_{\substack{[C^i \mapsto s_{c_1}^{ij1},\dots,s_{c_n}^{ijn}] \in G \\ \text{for } n \le m}} \; \sum_{\substack{\{I_1,\dots,I_n\} \in \\ \text{parts}(\{g_1,\dots,g_m\},n)}} \; \sum_{\substack{f \in \\ \text{perms}(\{1,\dots,n\})}} \\
& \quad \mathrm{p}\big(C_{I_{f(1)}}^{c_1},\dots,C_{I_{f(n)}}^{c_n} \,|\, C_{\{g_1,\dots,g_m\}}^{i}, G\big) \\
& \quad\quad \times \mathrm{p}\big(\mathbf{b}_{I_{f(1)}} \,|\, C_{I_{f(1)}}^{c_1},\dots,C_{I_{f(n)}}^{c_n}, C_{\{g_1,\dots,g_m\}}^{i}, G\big) \\
& \quad\quad \times \cdots \times \mathrm{p}\big(\mathbf{b}_{I_{f(n)}} \,|\, \mathbf{b}_{I_{f(1)}},\dots,\mathbf{b}_{I_{f(n-1)}}, C_{I_{f(1)}}^{c_1},\dots,C_{I_{f(n)}}^{c_n}, C_{\{g_1,\dots,g_m\}}^{i}, G\big) \\
= & \sum_{\substack{[C^i \mapsto s_{c_1}^{ij1},\dots,s_{c_n}^{ijn}] \in G \\ \text{for } n \le m}} \; \sum_{\substack{\{I_1,\dots,I_n\} \in \\ \text{parts}(\{g_1,\dots,g_m\},n)}} \; \sum_{\substack{f \in \\ \text{perms}(\{1,\dots,n\})}} \\
& \quad \mathrm{p}\big(C_{I_{f(1)}}^{c_1},\dots,C_{I_{f(n)}}^{c_n} \,|\, C_{\{g_1,\dots,g_m\}}^{i}, G\big) \\
& \quad\quad \times \mathrm{p}\big(\mathbf{b}_{I_{f(1)}} \,|\, C_{I_{f(1)}}^{c_1}, G\big) \times \cdots \times \mathrm{p}\big(\mathbf{b}_{I_{f(n)}} \,|\, C_{I_{f(n)}}^{c_n}, G\big)
\end{aligned}
\tag{4.4}
$$

and then to separate the structural and geometric likelihoods of the tree fragment, and apply the conditional independence among sibling parts given the parent:

$$
\begin{aligned}
\beta_i(\{g_1,\dots,g_m\}) = & \sum_{\substack{[C^i \mapsto s_{c_1}^{ij1},\dots,s_{c_n}^{ijn}] \in G \\ \text{for } n \le m}} \; \sum_{\substack{\{I_1,\dots,I_n\} \in \\ \text{parts}(\{g_1,\dots,g_m\},n)}} \; \sum_{\substack{f \in \\ \text{perms}(\{1,\dots,n\})}} \\
& \quad \mathrm{p}\big(C_{I_{f(1)}}^{c_1},\dots,C_{I_{f(n)}}^{c_n} \,|\, C_{\{g_1,\dots,g_m\}}^{i}, G\big) \\
& \quad\quad \times \mathrm{p}\big(\mathbf{b}_{I_{f(1)}} \,|\, C_{I_{f(1)}}^{c_1}, G\big) \times \cdots \times \mathrm{p}\big(\mathbf{b}_{I_{f(n)}} \,|\, C_{I_{f(n)}}^{c_n}, G\big) \\
= & \sum_{\substack{[C^i \mapsto s_{c_1}^{ij1},\dots,s_{c_n}^{ijn}] \in G \\ \text{for } n \le m}} \; \sum_{\substack{\{I_1,\dots,I_n\} \in \\ \text{parts}(\{g_1,\dots,g_m\},n)}} \; \sum_{\substack{f \in \\ \text{perms}(\{1,\dots,n\})}} \\
& \quad \mathrm{p}\big(C^{c_1},\dots,C^{c_n}, \,|\, C^{i}, G\big) \\
& \quad\quad \times \mathrm{p}\big(\mathbf{b}_{I_{f(1)}},\dots,\mathbf{b}_{I_{f(n)}} \,|\, \mathbf{b}_{\{g_1,\dots,g_m\}}^{i}, \phi^{ij1},\dots,\phi^{ijn}\big) \\
& \quad\quad \times \mathrm{p}\big(\mathbf{b}_{I_{f(1)}} \,|\, C_{I_{f(1)}}^{c_1}, G\big) \times \cdots \times \mathrm{p}\big(\mathbf{b}_{I_{f(n)}} \,|\, C_{I_{f(n)}}^{c_n}, G\big) \\
= & \sum_{\substack{[C^i \mapsto s_{c_1}^{ij1},\dots,s_{c_n}^{ijn}] \in G \\ \text{for } n \le m}} \; \sum_{\substack{\{I_1,\dots,I_n\} \in \\ \text{parts}(\{g_1,\dots,g_m\},n)}} \; \sum_{\substack{f \in \\ \text{perms}(\{1,\dots,n\})}} \\
& \quad \mathrm{p}\big(C^{c_1},\dots,C^{c_n}, \,|\, C^{i}, G\big) \\
& \quad\quad \times \mathrm{p}\big(\mathbf{b}_{I_{f(1)}} \,|\, \mathbf{b}_{\{g_1,\dots,g_m\}}^{i}, \phi^{ij1}\big) \\
& \quad\quad \times \cdots \times \mathrm{p}\big(\mathbf{b}_{I_{f(n)}} \,|\, \mathbf{b}_{\{g_1,\dots,g_m\}}^{i}, \phi^{ijn}\big) \\
& \quad\quad \times \mathrm{p}\big(\mathbf{b}_{I_{f(1)}} \,|\, C_{I_{f(1)}}^{c_1}, G\big) \times \cdots \times \mathrm{p}\big(\mathbf{b}_{I_{f(n)}} \,|\, C_{I_{f(n)}}^{c_n}, G\big) \quad .
\end{aligned}
\tag{4.5}
$$

These steps are very similar to the derivation presented in Section 2.6.

And finally, we use the results from Equation (4.1) above to replace the geometric characteristics of the parent node with the bounding box of the children, and then make the inductive step:

$$
\begin{aligned}
\beta_i(\{g_1,\ldots,g_m\}) = & \sum_{\substack{[C^i \mapsto s^{ij1}_{c_1},\ldots,s^{ijn}_{c_n}]\in G\\ \text{for } n\le m}} \sum_{\substack{\{I_1,\ldots,I_n\}\in\\ \text{parts}(\{g_1,\ldots,g_m\},n)}} \sum_{\substack{f\in\\ \text{perms}(\{1,\ldots,n\})}} \\
& \quad \mathrm{p}\big(C^{c_1},\ldots,C^{c_n},\,|\,C^i,G\big) \\
& \qquad \times \mathrm{p}\big(\mathbf{b}_{I_{f(1)}}\,|\,\mathbf{b}^i_{\{g_1,\ldots,g_m\}},\phi^{ij1}\big) \\
& \qquad \times\cdots\times \mathrm{p}\big(\mathbf{b}_{I_{f(n)}}\,|\,\mathbf{b}^i_{\{g_1,\ldots,g_m\}},\phi^{ijn}\big) \\
& \qquad \times \mathrm{p}\big(\mathbf{b}_{I_{f(1)}}\,|\,C^{c_1}_{I_{f(1)}},G\big)\times\cdots\times \mathrm{p}\big(\mathbf{b}_{I_{f(n)}}\,|\,C^{c_n}_{I_{f(n)}},G\big) \\
= & \sum_{\substack{[C^i \mapsto s^{ij1}_{c_1},\ldots,s^{ijn}_{c_n}]\in G\\ \text{for } n\le m}} \sum_{\substack{\{I_1,\ldots,I_n\}\in\\ \text{parts}(\{g_1,\ldots,g_m\},n)}} \sum_{\substack{f\in\\ \text{perms}(\{1,\ldots,n\})}} \\
& \quad \mathrm{p}\big(C^{c_1},\ldots,C^{c_n},\,|\,C^i,G\big) \\
& \qquad \times \mathrm{p}\big(\mathbf{b}_{I_{f(1)}}\,|\,\square(\mathbf{b}_{\{g_1,\ldots,g_m\}}),\phi^{ij1}\big) \\
& \qquad \times\cdots\times \mathrm{p}\big(\mathbf{b}_{I_{f(n)}}\,|\,\square(\mathbf{b}_{\{g_1,\ldots,g_m\}}),\phi^{ijn}\big) \\
& \qquad \times \mathrm{p}\big(\mathbf{b}_{I_{f(1)}}\,|\,C^{c_1}_{I_{f(1)}},G\big)\times\cdots\times \mathrm{p}\big(\mathbf{b}_{I_{f(n)}}\,|\,C^{c_n}_{I_{f(n)}},G\big) \\
= & \sum_{\substack{[C^i \mapsto s^{ij1}_{c_1},\ldots,s^{ijn}_{c_n}]\in G\\ \text{for } n\le m}} \sum_{\substack{\{I_1,\ldots,I_n\}\in\\ \text{parts}(\{g_1,\ldots,g_m\},n)}} \sum_{\substack{f\in\\ \text{perms}(\{1,\ldots,n\})}} \\
& \quad \left(\gamma^{ij}\prod_{k=1}^{n}\Big(\mathrm{p}\big(\mathbf{b}_{I_{f(k)}}\,|\,\square(\mathbf{b}_{\{g_1,\ldots,g_m\}}),\phi^{ijk}\big)\beta_{c_k}(I_{f(k)})\Big)\right) .
\end{aligned}
\tag{4.6}
$$

Although this derivation looks a little terrifying, it represents a straightforward extension of the inductive step in the original inside algorithm presented in Section A.1.2, supplemented by the ideas presented in Section 2.6.

Finally, note that if $i = 1$, we must also incorporate the root geometric score on the geometric characteristics of the entire scene, which is only included once for the entire tree. Thus we can summarize the result of the inductive step as:

$$
\beta_i(\{g_1,\ldots,g_m\}) = \begin{cases}
\sum_{\substack{[C^i \mapsto s^{ij1}_{c_1},\ldots,s^{ijn}_{c_n}]\in G\\ \text{for } n\le m}} \sum_{\substack{\{I_1,\ldots,I_n\}\in\\ \text{parts}(\{g_1,\ldots,g_m\},n)}} \sum_{\substack{f\in\\ \text{perms}(\{1,\ldots,n\})}} & \\
\quad \Big(\mathrm{p}\big(\square(\mathbf{b}_{\{g_1,\ldots,g_m\}})\,|\,\rho^i\big) & \\
\qquad \times\gamma^{ij}\prod_{k=1}^{n}\Big(\mathrm{p}\big(\mathbf{b}_{I_{f(k)}}\,|\,\square(\mathbf{b}_{\{g_1,\ldots,g_m\}}),\phi^{ijk}\big)\beta_{c_k}(I_{f(k)})\Big)\Big) & \text{if } i=1 \\
\sum_{\substack{[C^i \mapsto s^{ij1}_{c_1},\ldots,s^{ijn}_{c_n}]\in G\\ \text{for } n\le m}} \sum_{\substack{\{I_1,\ldots,I_n\}\in\\ \text{parts}(\{g_1,\ldots,g_m\},n)}} \sum_{\substack{f\in\\ \text{perms}(\{1,\ldots,n\})}} & \\
\quad \Big(\gamma^{ij}\prod_{k=1}^{n}\Big(\mathrm{p}\big(\mathbf{b}_{I_{f(k)}}\,|\,\square(\mathbf{b}_{\{g_1,\ldots,g_m\}}),\phi^{ijk}\big)\beta_{c_k}(I_{f(k)})\Big)\Big) & \text{otherwise.}
\end{cases}
\tag{4.7}
$$

Using All Subsets of Object Parts

The above derivation assumes that all m primitive parts are used in any parse tree. However, this is a limiting requirement for an object recognition system, as there will often be clutter elements in a scene that we would prefer to ignore.

As we discussed above, the parsing process is allowed to assign the "clutter" label to a primitive part in a scene (and therefore not include the part in the resulting parse tree), if it pays a constant penalty for each clutter part. The clutter penalty κ represents the approximate likelihood of a single unassigned clutter part; if a tree t covers only m of the M primitive parts in the scene, we simply multiply the likelihood of the tree by $\kappa^{(M-m)}$.

Nonetheless, the parsing process still must find the parse trees in the first place—even if they do not cover all the primitive parts. Currently, our rather primitive approach to addressing this problem is simply to run the above algorithm on all subsets of m primitive parts in the scene, and then sum the (appropriately penalized) likelihoods of the results. Clever memoization of intermediate values—partitions and permutations of index sets, and bounding boxes of subsets of primitive parts—can help to offset some of the inefficiency of the approach. Clearly, however, this approach is still intractable for large values of M.

4.2.2 Finding the Most Likely Parse of a Set of Object Parts

Using All Object Parts

As in the original inside algorithm for PCFGs, we can find the most likely parse tree of a set of object parts by simply taking maxes rather than sums in Equation (4.7), and then recording the rule, partition, and permutation associated with each max. In this way, the most likely tree can be constructed at the end.

To implement the dynamic programming, we define an accumulator $\delta_i(\{g_1, \ldots, g_m\})$, which stores the highest inside likelihood of a subtree $C^i_{\{g_1,\ldots,g_m\}}$. We also use a back-trace $\psi_i(\{g_1, \ldots, g_m\})$ to store the rule index j, part index partition $\{I_1, \ldots, I_n\}$, and assignment f of rule parts to index subsets that corresponds to the highest likelihood subtree at any step. Given these variables, the algorithm proceeds inductively much like above:

Base case:

$$\delta_i(\{g\}) = \begin{cases} \mathrm{p}(\mathbf{b}_g \mid \rho^i) & \text{if } i = 1 \text{ and } C^i \text{ is primitive} \\ 1 & \text{if } i \neq 1 \text{ and } C^i \text{ is primitive} \\ 0 & \text{otherwise.} \end{cases}$$

Induction:

$$\delta_i(\{g_1,\ldots,g_m\}) = \begin{cases} \max_{\substack{[C^i \mapsto s^{ij1}_{c_1},\ldots,s^{ijn}_{c_n}]\in G \\ \text{for } n\le m}} \max_{\substack{\{I_1,\ldots,I_n\}\in \\ \text{parts}(\{g_1,\ldots,g_m\},n)}} \max_{\substack{f\in \\ \text{perms}(\{1,\ldots,n\})}} \\ \quad \Big(\mathrm{p}\big(\square(\mathbf{b}_{\{g_1,\ldots,g_m\}})\,|\,\rho^i\big) \\ \quad\quad \times\gamma^{ij}\prod_{k=1}^n \Big(\mathrm{p}\big(\mathbf{b}_{I_{f(k)}}\,|\,\square(\mathbf{b}_{\{g_1,\ldots,g_m\}}),\phi^{ijk}\big)\beta_{c_k}(I_{f(k)})\Big)\Big) & \text{if } i=1 \\ \max_{\substack{[C^i \mapsto s^{ij1}_{c_1},\ldots,s^{ijn}_{c_n}]\in G \\ \text{for } n\le m}} \max_{\substack{\{I_1,\ldots,I_n\}\in \\ \text{parts}(\{g_1,\ldots,g_m\},n)}} \max_{\substack{f\in \\ \text{perms}(\{1,\ldots,n\})}} \\ \quad \Big(\gamma^{ij}\prod_{k=1}^n \Big(\mathrm{p}\big(\mathbf{b}_{I_{f(k)}}\,|\,\square(\mathbf{b}_{\{g_1,\ldots,g_m\}}),\phi^{ijk}\big)\beta_{c_k}(I_{f(k)})\Big)\Big) & \text{otherwise.} \end{cases} \tag{4.8}$$

Store the backtrace:

$$\psi_i(\{g_1,\ldots,g_m\}) = \operatorname*{argmax}_{(j,\{I_1,\ldots,I_n\},f)} \delta_i(\{g_1,\ldots,g_m\})$$

Tree readout: By construction, we know that the likelihood of the most likely parse tree t of the set of primitive parts rooted at C^1 and covering the primitive object parts $\{\mathbf{b}_1,\ldots,\mathbf{b}_M\}$ is given by:

$$\mathrm{p}(t\,|\,G) = \delta_1(\{1,\ldots,M\}) \;\; .$$

We know that the root node of t must be $a_1 = C^1_{\{1,\ldots,M\}}$. Then we use the backtrace to recursively construct the best set of child nodes of each internal node. If we have an internal node $a_\ell = C^i_{\{g_1,\ldots,g_m\}}$, and we have

$$\psi_i(\{g_1,\ldots,g_m\}) = (j,\{I_1,\ldots,I_n\},f)$$

in the backtrace, and

$$r^{ij} = [\gamma^{ij} \;\; C^i \mapsto s^{ij1}_{c_1}\ldots s^{ijn}_{c_n}]$$

in the PGG, then:

$$\begin{aligned} \mathrm{child}(a_\ell,1) &= C^{c_1}_{I_{f(1)}} \\ &\vdots \\ \mathrm{child}(a_\ell,n) &= C^{c_n}_{I_{f(n)}} \;\; . \end{aligned}$$

The readout process continues recursively until all the leaf nodes are primitive; then the most likely parse tree t is complete.

Using All Subsets of Object Parts

Again, our naïve approach to allowing the algorithm to ignore boxes in the scene is simply to find the most likely tree that covers each subset of m boxes in the scene, where there are M total primitive parts in the scene. Then we choose the tree with

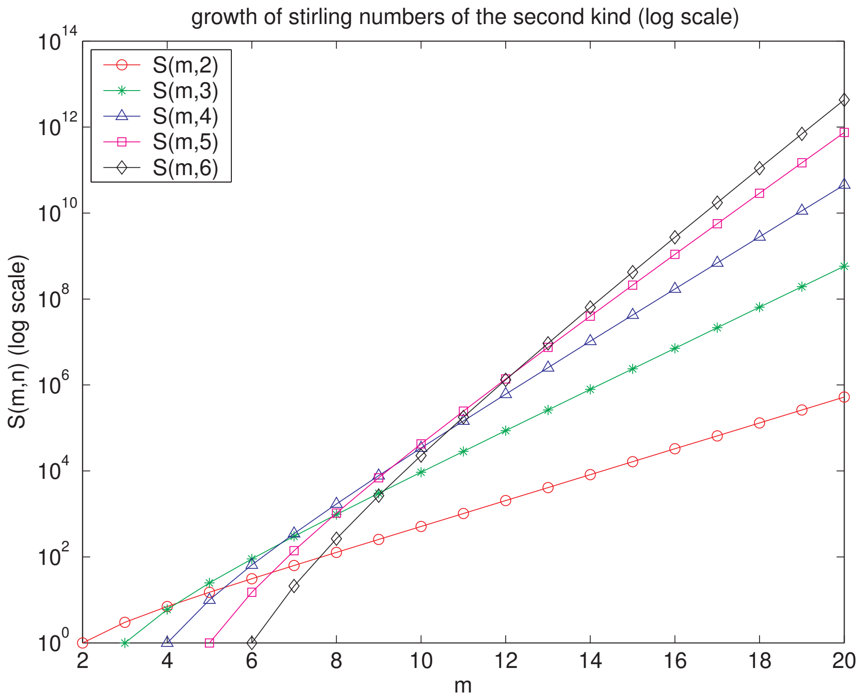


Figure 4-2: The growth of the Stirling numbers of the second kind $S(m,n)$.

the maximum likelihood, once the clutter penalty $\kappa^{(M-m)}$ has been applied to each tree.

4.2.3 Log Likelihoods

All the algorithms presented in this chapter, and in Section 2.6, are actually represented in their log form. The log likelihood is preferable in general to the straight likelihood, because the probabilities associated with complex models can often become so small that numeric underflow is a serious risk. Thus, the products described in these equations are actually sums of logs. Furthermore, the sums in the original equations are computed using the well known "logsumexp" procedure for summing a set of values that are represented in their log form, while avoiding numerical underflow.

4.3 Complexity Concerns

We have seen that, in the modified inside algorithm for PGGs, we must consider all possible assignments of object parts to rule parts at each level of the dynamic programming. This is debilitating: the number of possible ways to assign a set of object parts to the parts on the right side of a rule is exponential in the number of object parts.

Specifically, if there are n parts in a rule and m object parts in the scene, we must consider, and compute the summarization function for, $O((n!)S(m,n))$ possible assignments. The factorial term $(n!)$ counts the number of permutations of a set of n elements. The $S(m,n)$ term is a *Stirling number of the second kind*, defined as:

$$S(m,n) = \frac{1}{n!}\sum_{i=1}^{n-1}(-1)^i\binom{n}{i}(n-i)^m$$

which returns the number of ways of partitioning a set of m elements into n nonempty sets [38]. See Figure 4-2.

So, although the dynamic programming approach of the inside algorithm saves us from explicitly searching over an exponential number of parse trees, we are still left with an algorithm that is exponential in the number of object parts in the scene.

A fundamental issue is that the inside algorithm for PGGs offers rotational invariance—the same object can be arbitrarily rotated in 3D space, and the algorithm will parse it with identical accuracy according to the model. This flexibility is a valuable quality for a recognition algorithm to have, but perhaps the price of full rotational invariance is higher than we are willing to pay. In particular, many possible approaches to constraining the search over assignments of object parts to rule parts—for example, imposing an arbitrary ordering or adjacency constraints on the object parts in the scene—will compromise some aspect of rotational invariance.[3]

Although we do not implement or test any approximate parsing algorithms in this thesis, we discuss several approaches for further work in Chapter 6.

[3]Pollak et al. also modify the inside-outside parsing algorithm to apply it to 2D object recognition in [27]. However, to avoid exponential complexity, the group imposes a left-to-right and top-to-bottom order on the parts in images, thus sacrificing even two-dimensional rotational invariance.

Chapter 5

Experimental Results

In this chapter, we describe a set of experiments conducted to test the choices made in designing the PGG framework. The current system is limited to purely three-dimensional representation and recognition. The system also assumes that all parts are simple 3D boxes, represented as a vector of three half-dimensions, a three-element position vector, and a unit quaternion rotation. Due to these constraints, we use synthetic data to approximate real data, and compare the performance of the PGG framework against the performances of baseline models on the synthetic data.

The chapter proceeds as follows:

Section 5.1 describes the hypothesis that the experiments are designed to test, and discusses our general approach and performance metrics.

Section 5.2 describes our implementation of the PGG framework.

Section 5.3 discusses the design of the experiments, including the form of the baseline models, the object classes, the synthetic data, and the training and testing procedure.

Section 5.4 presents and discusses the results of the experiments.

5.1 Hypothesis and Approach

We have argued in this thesis that the use of context-free grammars to model object classes enables effective parameter learning from fewer examples and better generalization of the learned models to unseen instances. The motivation for this hypothesis lies in two distinguishable aspects of the PGG framework:

- the conditional independence of each object part from all other parts, given its parent and children; and
- the sharing of geometric information between models for object classes with similar subparts.

Both of these aspects contribute to the fact that a PGG requires relatively few parameters to model object classes.

However, any claim that uses comparatives such as "fewer" and "better" must be evaluated in comparison to a baseline. We choose two baseline models which attempt to separate the two distinguishable aspects above. Thus, if the PGG framework in fact offers better performance, we might understand to which property of the model to attribute the improvement. The baseline models are described in detail in Section 5.3.1.

We are interested in testing the expressive power and rate of learning of the PGG framework. Thus, our experiments do not explicitly test the accuracy of the parsing algorithms we presented in Chapter 4. Instead, the training *and* test sets contain fully parsed and labeled geometric instances (described in Section 5.3.3). We measure the performance of each model by finding the total log likelihood it assigns to the set of held-out test examples.[1] Finally, we measure the rate of learning by varying the number of training examples the models are given, and comparing the relative values of the log likelihood score given by each model to the test data.

We mentioned that these experiments do not explicitly test parsing accuracy under a variety of circumstances or clutter environments. However, we did perform sufficient testing to ensure that the parsing algorithm performs accurately on simple cases. In fact, on the synthetic data sets we describe below, it achieves almost flawless parsing accuracy. When presented with 20 single objects from each of 12 ground object classes, the algorithm produces the correct parse on 237 out of 240 test instances, for a parsing accuracy of 98.75%. Any errors appear to be due to an insufficient hand-chosen clutter penalty (-15.0 log likelihood), resulting in the omission of optional parts such as chair arms, or the re-ordering of similar object parts with ambiguous geometric characteristics, such as mixing up the legs of the chair.

Furthermore, as we would expect, the log likelihood that the parsing algorithm assigns to the resulting parse tree is (almost always) identical to the likelihood returned by a simple calculation of the log likelihood of the tree according to Equation (2.18); the log likelihoods differ on 11 out of the 237 correct parses, or 4.64%. When differences occur, they are small relative to the magnitudes of the log likelihoods being compared, and are due to rounding errors and resulting slight discrepancies in the choice of unit quaternions to represent rotation by the approximate bounding box algorithm. Based on these results, we can assume that the log likelihood of a hand-parsed test instance is equivalent to the log likelihood of the tree that would be produced by explicitly parsing the unlabeled primitives of the test tree.

This experimental approach is beneficial because the exponential complexity of the parsing algorithm means that parsing objects with a large number of parts can be prohibitively time consuming. Parsing a simple chair with four legs and no arms takes a few seconds, while parsing a chair with five wheel legs and two arms takes over two hours! The experimental methodology we describe here lets us perform some

[1] As mentioned in Chapter 4, the log likelihood is preferable in general to the straight likelihood, because the probabilities associated with complex models can often become so small that numeric underflow is a serious risk. Therefore, all likelihoods in the implementation are actually represented in their log form.

testing of the model and learning algorithms, even though our parsing algorithm is not yet tractable.

5.2 PGG Implementation

The PGG framework was implemented in the Java 1.4.2 programming language, supplemented by the Java 3D 1.3.1 package. All elements of the system were implemented by the author, with two exceptions. First, some core functionality of the computation of approximate minimum volume bounding boxes was provided by the implementations described in Section A.4 and [21] and [24]. Second, Michael Ross provided his implementation of the algorithm for Singular Value Decomposition (SVD), as given by *Numerical Recipes in C* [28].

Documentation and an API for the PGG software system may be found at

`http://www.csail.mit.edu/~aycinena/obj_rec/docs/`

and source code is available from the author upon request.

5.3 Experimental Setup

In this section, we describe the design and set up of the experiments we conducted. First, we describe the baseline models against which we compared the performance of the PGG algorithm. Then, we describe the object classes and the form of the synthetic data used for training and testing. Finally, we present the training and testing procedure.

5.3.1 Baseline Models

We shall consider two baseline models for comparison in these experiments: a *fully connected model*, and a *Bayesian network* (Bayes net).

A fully connected model specifies a large joint Gaussian distribution over the geometric characteristics of every primitive and composite part in a test parse tree. It is called *fully connected* because the geometric characteristics of any part in the tree can depend arbitrarily on any other part in the tree—therefore the "tree" is actually a fully connected graph. (Note, however, that a fully connected model still includes nodes for the composite parts of test trees, as well as the primitive parts, in order to define a distribution that is comparable to the tree-based models.)

A Bayes net model, in contrast, is a tree model. A Bayes net specifies a set of geometric conditional Gaussian distributions at each internal node, and a single geometric unconditional Gaussian distribution at the root node. The geometric characteristics of each part in a test parse tree are therefore conditionally independent from those of all other parts, given its parents and children. Note that the geometric distribution defined by a Bayes net model is identical to the geometric distribution specified by a PGG over object trees with a *fixed structure*. (Indeed, if a PGG and a

Bayes net are trained on a examples from a single ground class with a fixed structure, they exhibit identical performance; i.e., they assign identical log likelihoods to test examples.)

Any single instance of either of these models can represent only a single "ground" object class, because there is no choice between rules as there is in a grammar.[2] Therefore, in order to test on multiple ground object classes, we must actually compare a single PGG model against:

- an enumerated set of fully connected models; and
- an enumerated set of Bayes nets;

where there is a model in each enumerated set that corresponds to each structural parse tree that the PGG can produce. See Appendix B for examples.

Furthermore, both the fully connected models and the Bayes nets only define geometric distributions, because their structures are fixed. To make the comparisons valid, we must assign a prior probability to each model in each enumerated set, which corresponds to the *structural likelihood* of the corresponding parse tree produced by the PGG.

Note that neither of these types of baseline models can share geometric information between models for different ground object classes. And, while the Bayes net has conditional independence assumptions that are identical to those of a PGG, the fully connected model does not. Thus we have succeeded in separating these two aspects of the PGG framework.

The baseline models were implemented in Java.

5.3.2 Object Classes

The experiments were conducted on the following twelve ground object classes:

chair with legs	bench
chair with legs and arms	bench with arms
chair with 3 wheels	stool
chair with 3 wheels and arms	table
chair with 5 wheels	coffee table
chair with 5 wheels and arms	lamp

Some of the ground classes have significant common substructure with other classes (e.g., the chair and bench classes), while other classes are largely independent (e.g., the stool, table, coffee table, and lamp classes).

The PGG for these object classes is shown in Figures B-1 and B-2 in Appendix B. The grammatical structure of the PGG was specified by hand, but the expansion probabilities and geometric models were learned from the training data, according to

[2]We will use the terms *ground object class* or *ground class* to refer to an object class with a single fixed structure, such as "chair with four legs and two arms". This contrasts with our use of *object class* to refer to general basic semantic classes such as "chair" or "lamp".

the process we will describe below. Note which part classes are shared among object classes.

The equivalent sets of fully connected models and enumerated Bayes nets for the ground classes are shown in Figures B-3, B-4, and B-5. Again, the structure of the models was specified by hand and the prior probabilities on each model and the geometric models were learned from the training data.

5.3.3 Synthetic Data

One of the goals of our experiments is to quantify the tradeoff between representational power and speed of learning that is implied by the conditional independence assumptions of the PGG framework. Therefore, the synthetic data used in the experiments was constructed to have as many correlations among the geometric characteristics of object parts as possible.

Five hundred example objects were generated for each of the twelve ground object classes listed above. To create each example object in each ground class (except the lamp class), three random dimensions were chosen. The dimensions of each object part were then set, to either exactly the randomly chosen dimensions or some constant multiple of them. For each example in the lamp class, a single random dimension was chosen. The position and rotation of object parts were generated according to a noisy Gaussian distribution, with hand-chosen parameters inspired by measurements of real world objects. Other aspects of the data generation process included a random displacement between the seat and back of chairs, and a random skew of chair legs outward from the base.

Each example object was then converted into a parse tree using a hard-coded parsing process (i.e. simple deterministic application of the rules, rather than search), performed according to a grammatical structure identical to that of the PGG in Figures B-1 and B-2.

Finally, each set of 500 examples for a ground object class was divided into a pool of 300 possible training examples and 200 held-out test examples. In the experiments, training sets were always chosen from the 300 training examples for each object class, while test sets were always chosen from the 200 held-out examples for each object class.

5.3.4 Training and Testing Procedure

The training and testing procedure is defined as follows:

1. For each ground class, choose a fixed set of 10 test examples in advance from the held-out test set for that class. (Thus the entire test set for this procedure contains 120 examples.)

2. For each increasing even value of n, for $n = 2, 4, 6, \ldots, 50$:

   (a) Perform the following trial 30 times:

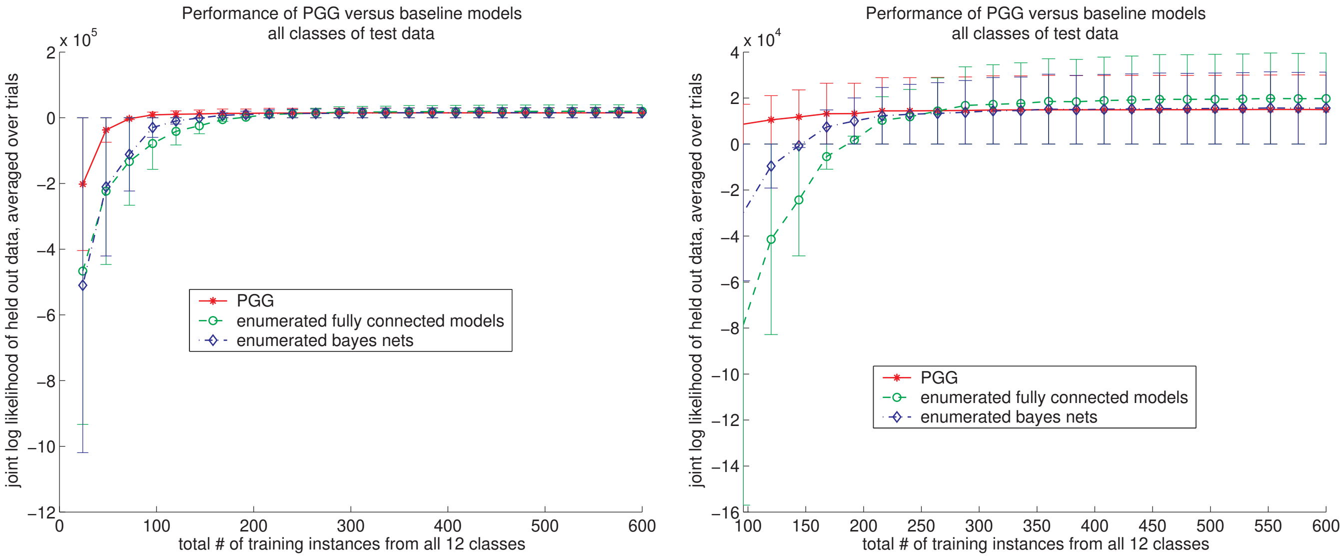


Figure 5-1: Performance of the PGG framework and the baseline models on all 12 ground classes, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance.

i. Randomly choose n examples from the training set for each class, so that the training set for this trial contains $12 \times n$ training examples.

ii. Train the PGG and each set of baseline models on the training set for this trial.

iii. For each type of model, calculate the log likelihood of each example in the test set according to the model, and sum the results over all 120 examples. This produces the joint log likelihood of the test data given the model.

(b) Take the "logmeanexp" of the set of 30 results (one from each trial).[3] Plot this value at the nth point on the x-axis of a graph. Thus, the mean (and standard deviation, for error bars) is calculated correctly for the likelihood rather than the log likelihood, although the mean log likelihood is actually plotted.

5.4 Results and Discussion

We ran the training and testing procedure on the data sets described in the previous section. The results are shown in Figures 5-1 through 5-13. In each figure, the graph on the left shows the results for increasing numbers of training examples, with

[3]The "logmeanexp" is similar to the well known "logsumexp" procedure for summing a set of values that are represented in their log form, while avoiding numerical underflow. First, the maximum value in the set is subtracted from each value in the set. Then, the log of the mean of the exponent of the values is computed, and finally the original maximum value is added back onto the mean.

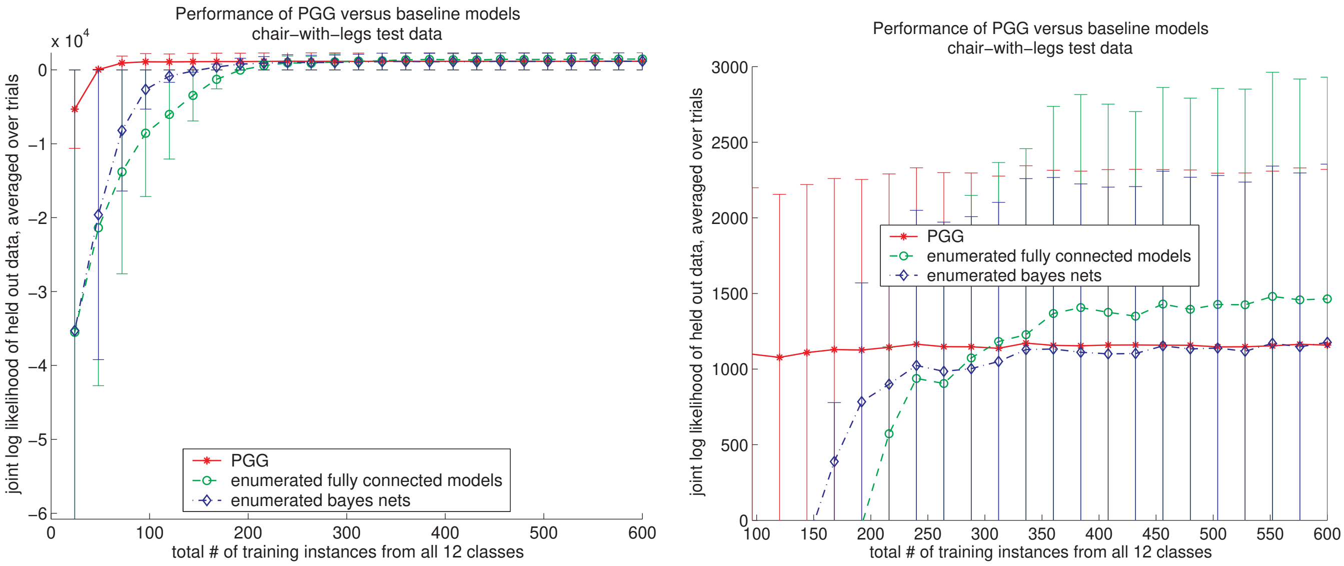


Figure 5-2: Performance of the PGG framework and the baseline models on the chair-with-legs ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance.

numbers varying from 12×2 to 12×50. The graph on the right shows a zoomed-in version of the left graph—here the number of training examples from all classes starts at 96, allowing the viewer to better compare the asymptotic performances of the models.

To see the varying performance on different ground classes, Figures 5-2 through 5-13 depict the performance of the models on test examples from a single group class (although the models are always trained on examples from all twelve classes).

First, we consider the results on test examples from ground classes that share information with other ground classes—Figures 5-2 through 5-9.

A first observation we can make from these figures is that the PGG clearly produces markedly better performance than either baseline model for fewer numbers of training examples. Second, the Bayes net models perform better than the fully connected models for fewer numbers of training examples.

These observations verify the expectation that using a grammar to model object classes enables effective parameter learning from fewer examples and better generalization of the learned models to unseen instances. We can attribute some success here to the conditional independence assumptions—the assumptions allow a model to learn more quickly because there are fewer parameters to adjust.

We can attribute more of the success, however, to the sharing of geometric information between models for object classes with similar subparts—a strategy that allows, for example, the PGG to use the instances of chairs-with-legs-and-arms that it has seen, to help it make decisions about instances of chairs-with-wheels-and-arms. It is this kind of generalization of information beyond a single ground class which explicitly demonstrates the power of a geometric grammar.

Now we consider the results on test examples from ground classes that do not

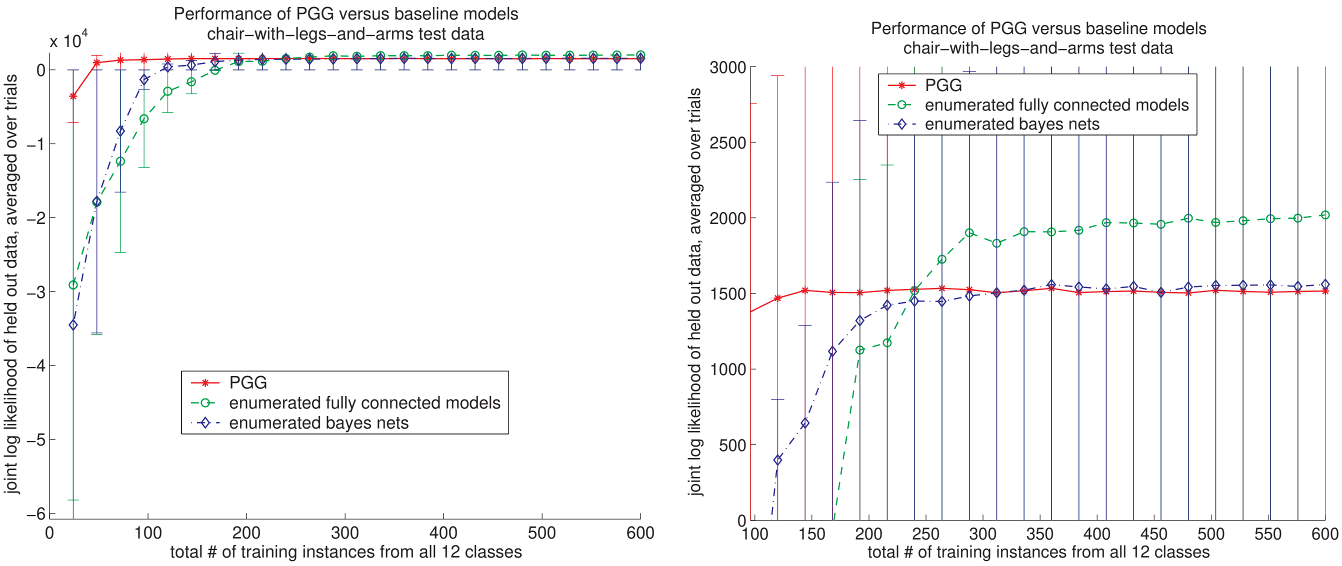


Figure 5-3: Performance of the PGG framework and the baseline models on the chair-with-legs-and-arms ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to show asymptotic performance.

share information between ground classes—Figures 5-10 through 5-13. Comparing these results to the earlier results on the "information-sharing" classes offers yet another way to see the importance of sharing between ground classes with similar substructure. On the non-information-sharing ground classes, the advantage of the PGG over the baseline models when trained with fewer examples is limited, and in fact there may be no advantage at all.

What about our goal of quantifying the tradeoff between representational power and speed of learning implied by the conditional independence assumptions? The synthetic data was explicitly generated to have the high levels of correlation among the parts necessary to test the tradeoff. Clearly, the fully connected models must have more expressive power than models with aggressive conditional independence assumptions.

Indeed, as the number of training examples increases, the fully connected model outperforms the other two types of models on all twelve classes. However, the difference in asymptotic performance between the fully connected models and the others is small, particularly when compared with the advantage of the conditionally independent models over the fully connected models when trained fewer training examples and tested on "information-sharing" ground classes.

It is also interesting to note that, for "oddly shaped" classes, such as benches or lamps, the PGG performs slightly worse in the limit than either baseline model. This is due to the root geometric model, which attempts to model a prior distribution over the geometric characteristics of *any* object, including characteristics such as scale, aspect ratio, and position. Thus, since a full half of the training examples on each iteration are chairs, we would expect a lower score for objects that have a different

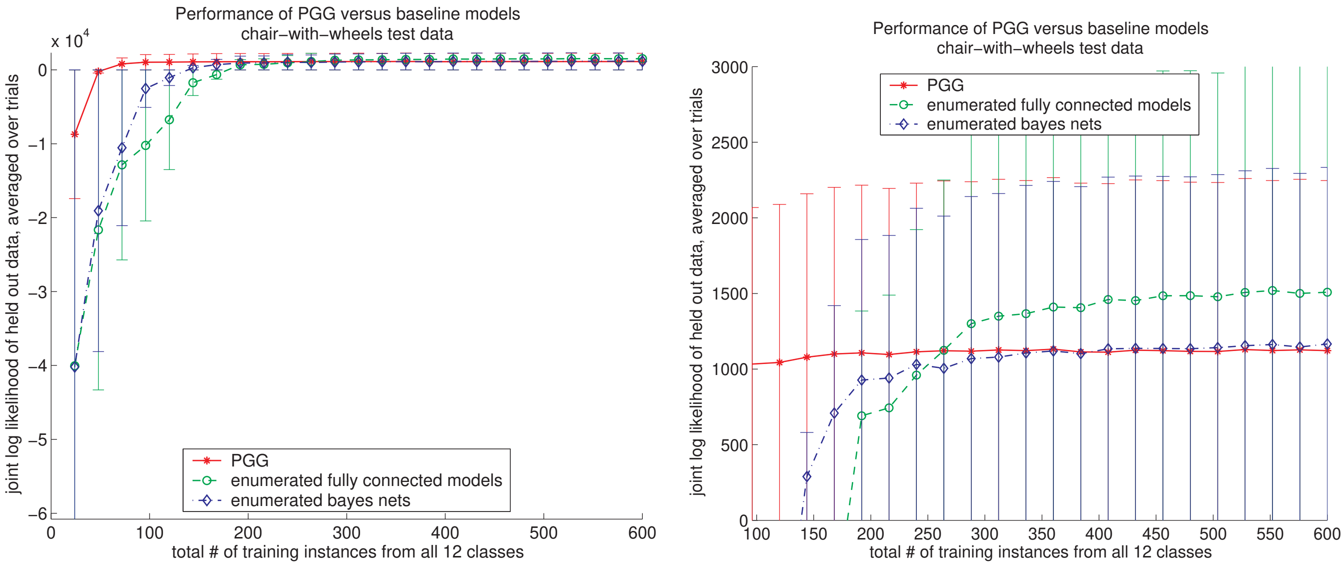


Figure 5-4: Performance of the PGG framework and the baseline models on the chair-with-3-wheels ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance.

general shape or size than chairs. In the future, we probably would prefer to have a "null" root geometric model for the scene class, and instead consider the *object class* node (e.g. chair, bench, etc.) as the root, for the purposes of calculating the geometric score.

Finally, to understand the significance of these results, consider the importance of learning from fewer examples in an object recognition system. Large numbers of high quality labeled training examples, especially in 2.5 or 3D, are extremely expensive to obtain. In order to adequately scale to recognizing instances from many general classes of objects, a system must be able to exploit information from previously seen objects—even if the new objects are not absolutely identical in structure to those the system has seen before.

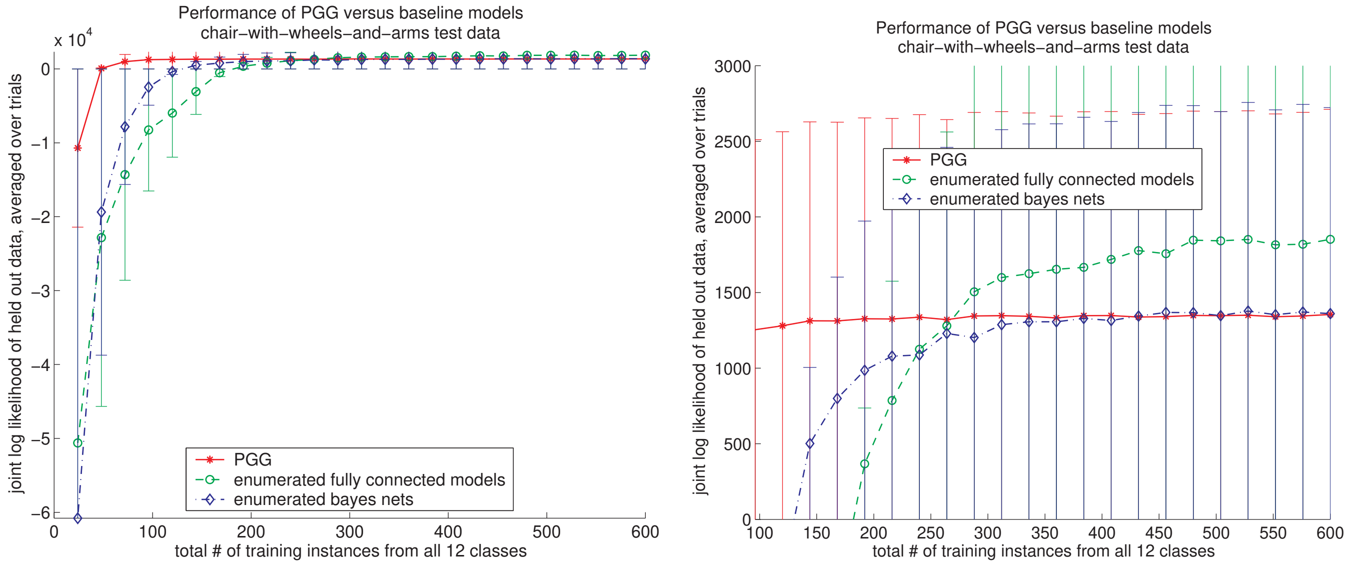


Figure 5-5: Performance of the PGG framework and the baseline models on the chair-with-3-wheels-and-arms ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance.

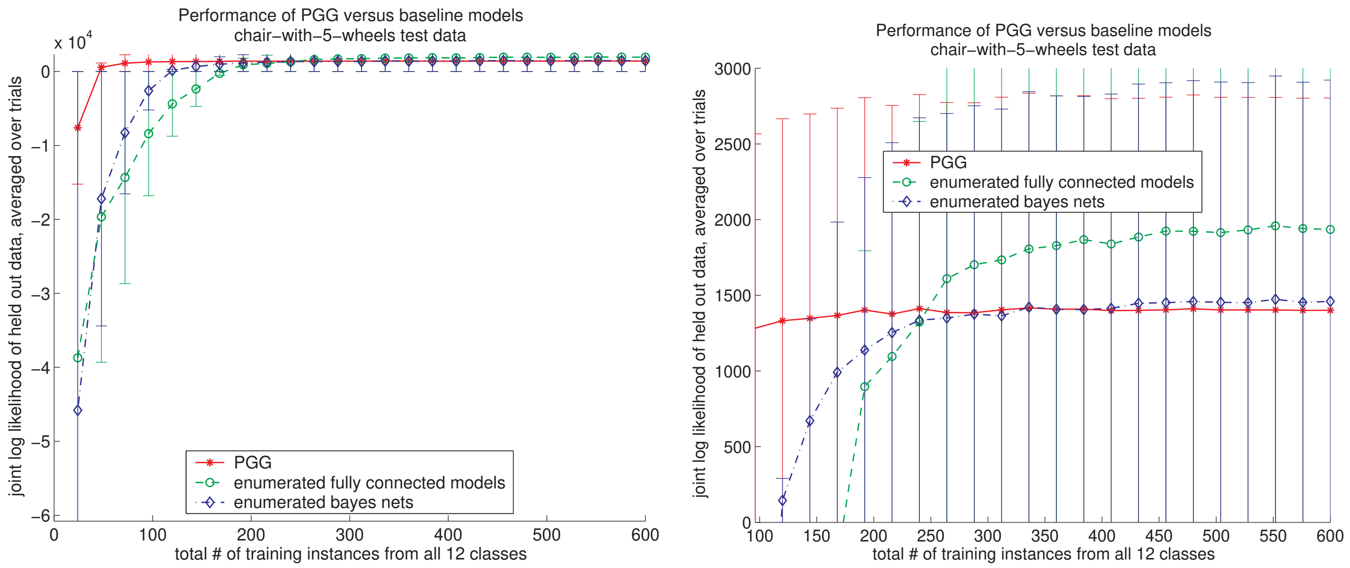


Figure 5-6: Performance of the PGG framework and the baseline models on the chair-with-5-wheels ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance.

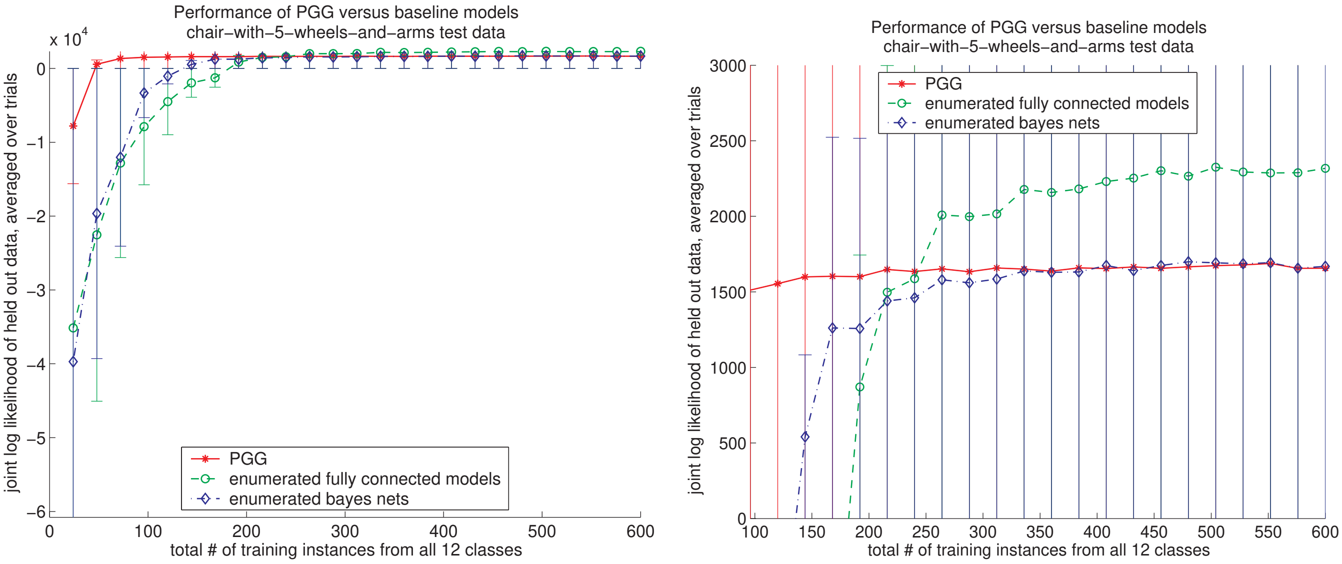


Figure 5-7: Performance of the PGG framework and the baseline models on the chair-with-5-wheels-and-arms ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance.

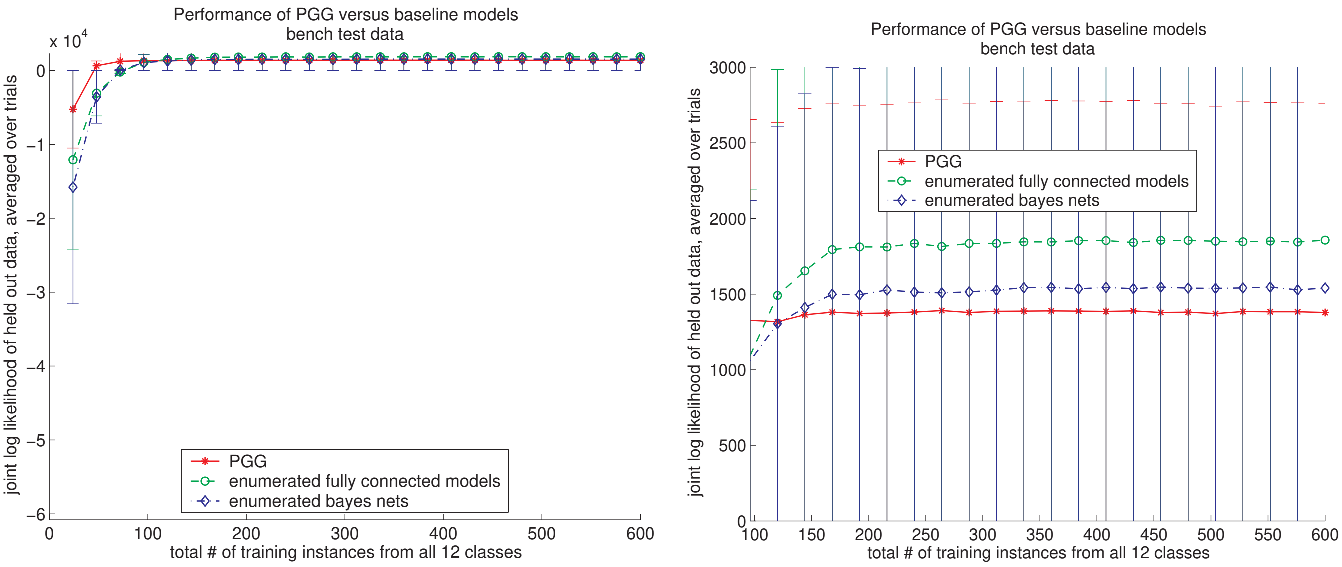


Figure 5-8: Performance of the PGG framework and the baseline models on the bench ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance.

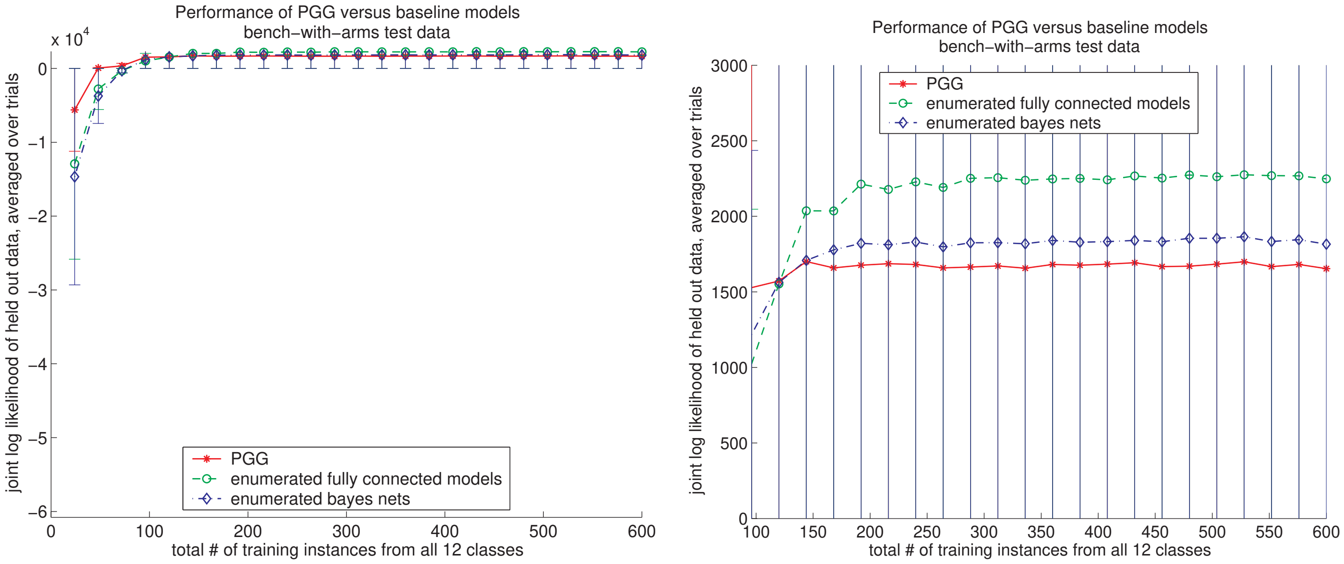


Figure 5-9: Performance of the PGG framework and the baseline models on the bench-with-arms ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance.

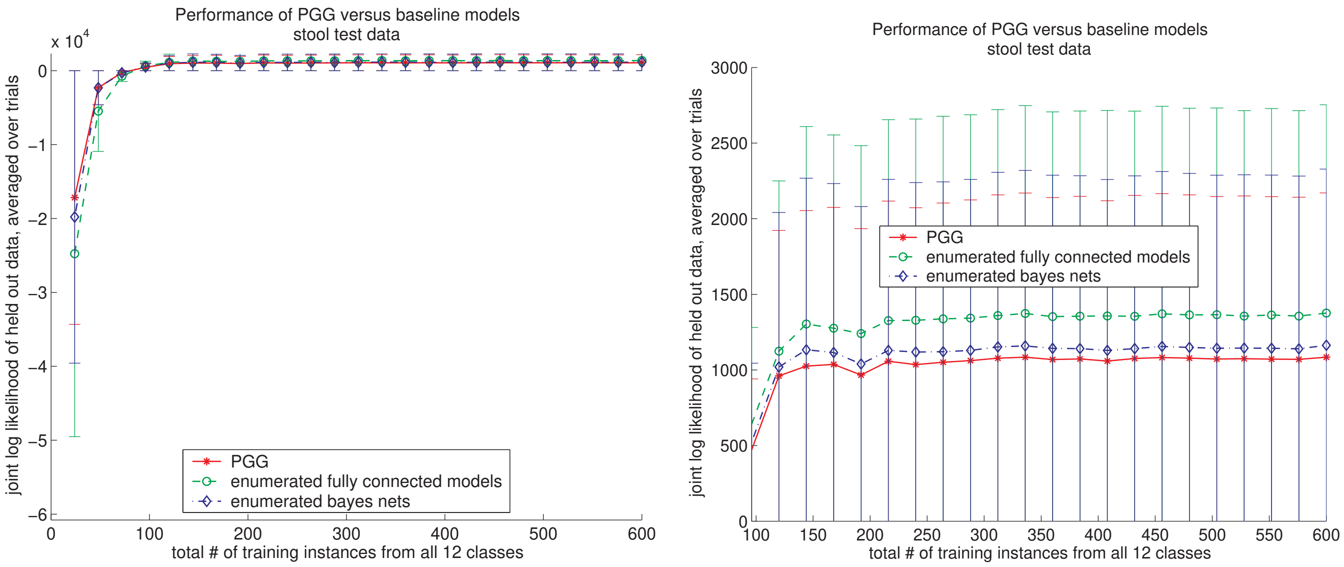


Figure 5-10: Performance of the PGG framework and the baseline models on the stool ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance.

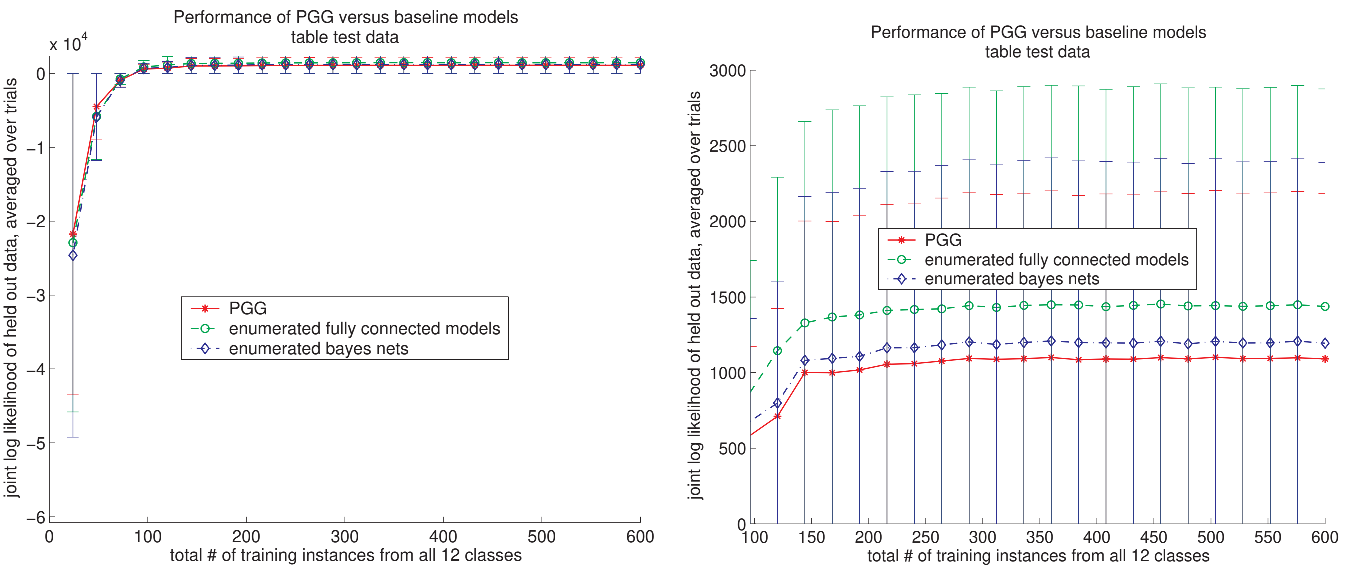


Figure 5-11: Performance of the PGG framework and the baseline models on the table ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance.

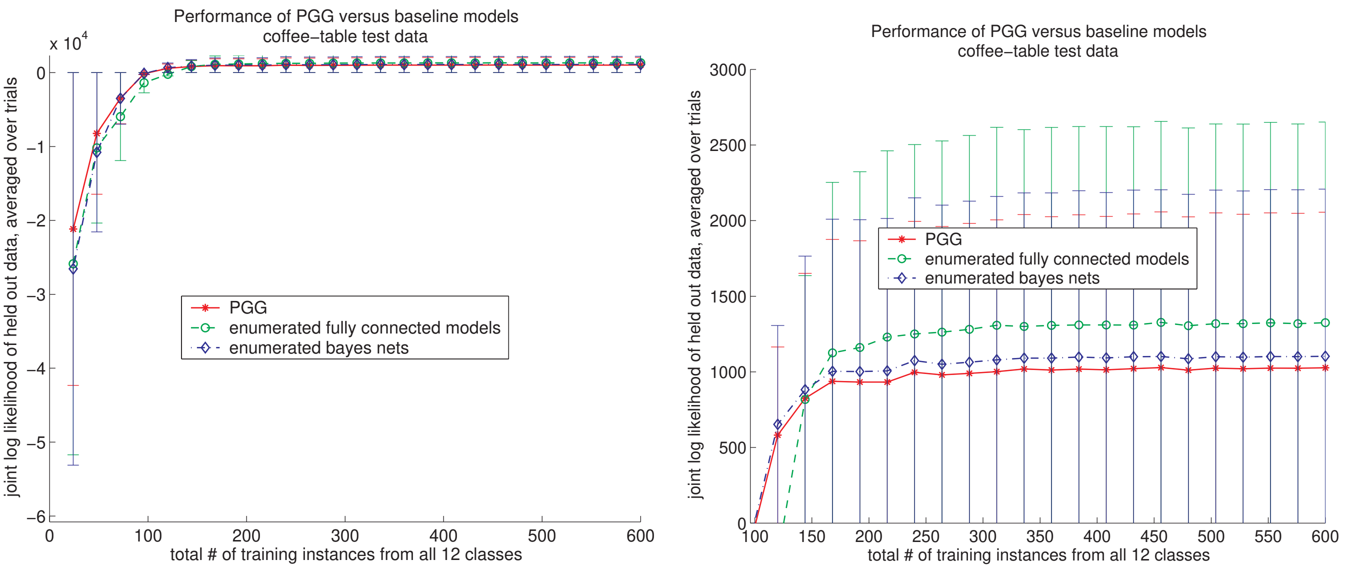


Figure 5-12: Performance of the PGG framework and the baseline models on the coffee-table ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance.

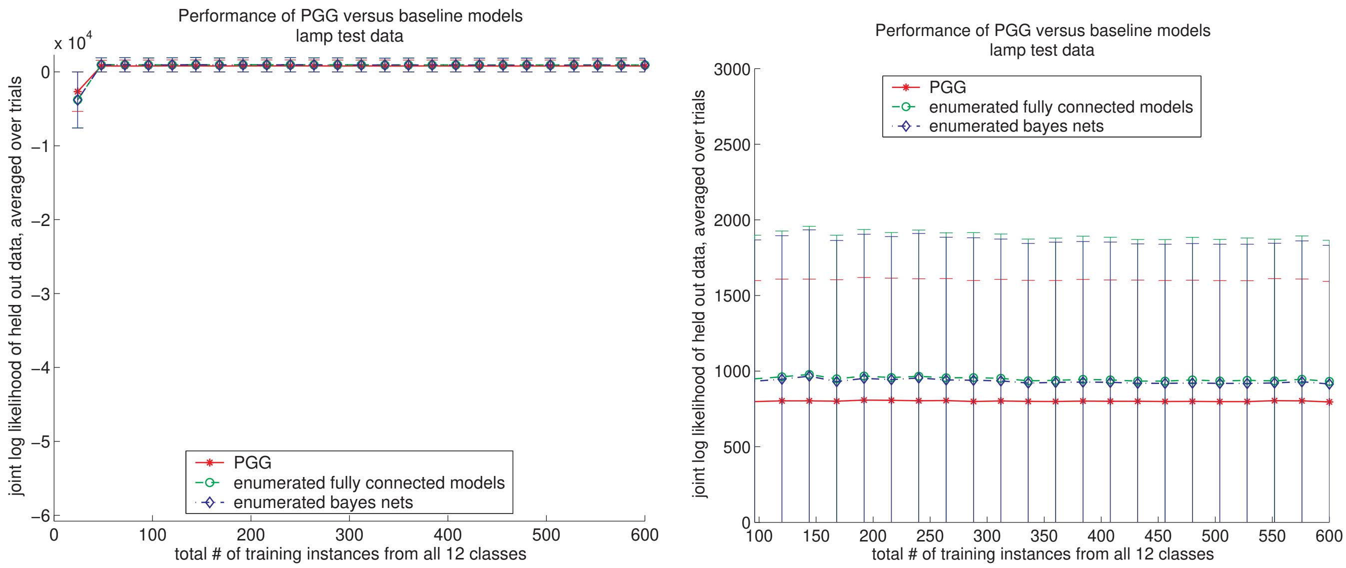


Figure 5-13: Performance of the PGG framework and the baseline models on the lamp ground class, for increasing amounts of training data. The graph on the right is a zoomed-in version of that on the left, to better show asymptotic performance.

Chapter 6

Conclusion

In this chapter, we discuss avenues for further research in the PGG framework, and then conclude.

6.1 Future Work

6.1.1 Recognition

A Tractable Parsing Algorithm

As we described in Chapter 4, a heuristic approach to parsing in the PGG framework is essential in order to achieve tractability. Therefore, our first priority in future work is the development of a polynomial time parsing algorithm. There are several possible approaches we might consider.

One possibility is to use the part geometric scores to aggressively prune unlikely assignments or subtrees, so that they are not considered at higher levels of the dynamic programming. However, recall that the summarization process means that there is no way to compute the geometric likelihood of a single part given its parent without committing to the assignments of the other sibling parts in the tree fragment. It is not clear whether this aspect of the PGG framework will make pruning approaches using the part geometric models more difficult. As we suggested in Chapter 4, imposing a limit on the number of rule parts in each rule might address this issue, should it prove to be problematic.

Alternatively, a pruning strategy could use the root geometric models at lower levels of the tree to eliminate unlikely assignments or subtrees. This would be purely a heuristic approach—although the conditional independence assumptions of the PGG framework are aggressive, a part's geometric characteristics are still assumed to be very much dependent on those of its parent. However, the root geometric models might be extremely useful as heuristics, especially for certain kinds of information, such as the aspect ratio of parts.

Both of these pruning strategies require learning a likelihood threshold to define "unlikely" subtrees. Each geometric model would probably require its own threshold, but the limited scope of each threshold might make its value easier to calibrate. The threshold could even be a function of some aspect of the geometric model itself, such as the likelihood value some number of standard deviations away from the mean of the Gaussian.

The unordered rotationally invariant problem is so hard, however, that it is probably computationally infeasible to even consider all possibilities and combinations of groupings and assignments. Therefore, we must consider approaches that can eliminate possibilities without actually looking at them. A simple ordering constraint, such as the left-to-right and top-to-bottom ordering imposed by Pollak et al. in [27], would achieve this, but would sacrifice any hope of rotational invariance.

A slightly more sophisticated approach might be to limit the size of the problem that the algorithm is allowed to consider at any time. For example, we could create an adjacency graph, where the primitive parts in the scene are nodes in the graph, and parts that are "close enough" to one another are connected with an edge. Then, we could set a threshold on the maximum length of the path that can connect any set of nodes that are combined using a rule. We would then have to learn or calibrate parameters to determine the meaning of "close enough" and the threshold on path length in the adjacency graph.

Testing Parsing Accuracy

There are a number of experiments beyond those presented in this thesis that we would like to try, to test various aspects of the PGG framework and its parsing abilities. In most cases, these tests would be quite difficult without a tractable parsing algorithm.

First, we would like to parse full scenes with multiple objects. A grammar could nicely describe arrangements of furniture just as it describes arrangements of object parts, and we would like to test this possibility.

Second, we would like to test the robustness of the framework by testing its resistance to distracting clutter. Extensive development on scenes with clutter would also allow us to better tune the clutter penalty of the PGG.

Third, we would like to parse "real" data, such as segmented CAD objects or 3D scans of actual objects. The logistics of obtaining such data aside, this would let us determine to what degree the conditional independence assumptions of the framework apply to the geometric characteristics of actual objects.

Finally, the assumption that "real" data would be segmented is a strong one. Furthermore, it seems like any successful segmentation approach would need a fair amount of top down information, so it might be best to approach the segmentation and parsing problems in parallel, or at least with significant feedback between the two processes.

Beyond Three Dimensions

As stated in the introduction, this thesis is focused on recognition and learning given three-dimensional input, but eventually recognition and learning must occur from two-dimensional images. It is not yet clear how best to approach this problem.

One possibility is to perform recognition of 2D or 2.5D images, given a 3D model, by searching over viewpoints and matching the projection of the model against the image. Clever use of feature, part, or region detectors at the 2D image level could guide the search over viewpoints.

However, how should parsing occur in 2D recognition? It could happen as an outer loop at the level of the 3D model, where searching over projections occurs for a fixed structure of the object. An alternative would be to perform the parsing in projected space. Although this seems mathematically and computationally daunting, it seems worth investigation.

6.1.2 Representation

The PGG framework, as presented in this thesis, limits primitive and composite object parts to simple three-dimensional boxes. A natural next step would be to consider richer representations of shape, such as generalized cylinders or geons [13].

Another aspect of the framework we might reconsider is the aggressiveness of the conditional independence assumptions. It might be beneficial to consider ways to allow the sharing of information among parts that cannot be directly specified in a tree format. This would allow us to say explicitly, for example, that the length of the arms of a chair should be the same as the length of the legs, which is not directly possible in the current framework. We might look to the concepts of augmented grammars or head phrase structure grammars in natural language processing for inspiration [1]. However, it is possible that the richer expressive power offered by these additions might not be worth the requisite computational complexity for supporting them.

Finally, we might consider supplementing the 3D model of a PGG with some view-based representation or caching system, to aid in 2D recognition or learning.

6.1.3 Learning

The learning algorithms we presented in Chapter 3 are not particularly sophisticated, so there are a number of ways in which we could extend research in the area of learning.

A natural first step might be to learn expansion probabilities and geometric models from unparsed but labeled examples. An algorithm to address this problem could be based on the EM approach used by the natural language processing to learn the probabilities on rules in a PCFG from an unparsed training corpus, in which parse trees are considered a hidden variable. The next step after that would be learning from unparsed *and* unlabeled examples.

We discussed the recognition problem on "real" data above; the analogous problem exists for learning. Additional work might include learning geometric models from segmented or unsegmented CAD or 3D object scan examples, range data, reconstructed-from-video models, or from 2D or 2.5D images.

Also, we discussed briefly in Chapter 3 the issue of incremental learning. As the system recognizes new instances, these examples might be added to the training set in order to refine the rule and geometric probability models. Thus, an important area of future work is the modification the learning algorithms in Chapter 3 to operate incrementally, so that the set of all training examples would not need to be maintained indefinitely.

Finally, there is the problem of learning the structure of the grammar itself. We mentioned in Chapter 3 that grammar induction is known to be a quite difficult problem in natural language processing. However, it would be interesting to consider how the structural learning problem might be approached with the addition of geometric information. For example, the variance or entropy—the "flatness"—of the geometric models might help us decide when to split a grammar class into subclasses.

6.2 In Conclusion

This thesis has presented the probabilistic geometric grammar framework, which is a generative parts-based three-dimensional representation and recognition framework for classes of objects. We have explained the ways in which a PGG is similar to, and differs from, a PCFG, and described and derived the form of the geometric models. We have presented algorithms for learning and parsing in the framework. We have also shown empirically that using context-free grammars to model object classes allows effective parameter learning from fewer examples and better generalization of the learned models to unseen instances than baseline models. Finally, having demonstrated that the PGG framework warrants further research, we have proposed possible avenues for such future work.

Appendix A

Background

This appendix presents the fundamentals upon which the PGG framework is built. A review of these concepts is essential to a full understanding of this thesis.

The appendix is organized as follows:

Section A.1 describes basic probabilistic context-free grammars (PCFGs) for language, and presents the inside algorithm for efficient parsing in PCFGs.

Section A.2 motivates the choice of the quaternion representation for rotation, describes basic quaternion algebra, discusses how unit quaternions are used to represent rotation, and shows how to model and estimate Gaussian distributions over unit quaternions.

Section A.3 derives the process of factoring a multivariate joint Gaussian over $\mathbb{R}^n$ into marginal and conditional Gaussian distributions.

Section A.4 describes an algorithm for estimating the minimal bounding box of a set of 3D boxes.

This material is presented as part of the thesis to serve as convenient background material for the reader, because an understanding of this material is necessary for the comprehension of my work. I owe much of my own understanding and explanation in these areas to the sources I cite. There is a brief description of these sources and related work at the end of each section, to which I refer interested readers for additional information on each topic.

A.1 Probabilistic Context-Free Grammars

A probabilistic geometric grammar (PGG) is a probabilistic context-free grammar (PCFG) augmented with geometric information. Therefore, readers must be familiar with PCFGs in order to understand the PGG framework.

In Chapter 1, we briefly introduced context-free grammars (CFGs) and their probabilistic equivalents. In this section we will give a more formal introduction to PCFGs as used in linguistics and natural language processing, and then present the inside algorithm for efficient parsing in PCFGs.

0.3	NP	$\mapsto$	ART NP	1.0	ART	$\mapsto$	*the*
0.3	NP	$\mapsto$	ADJ NP	0.5	ADJ	$\mapsto$	*big*
0.4	NP	$\mapsto$	N	0.5	ADJ	$\mapsto$	*red*
				0.2	N	$\mapsto$	*barn*
				0.8	N	$\mapsto$	*ball*

Figure A-1: A probabilistic context-free grammar for a tiny subset of English noun phrases.

A.1.1 An Introduction to PCFGs for Language

In Figure 1-2 we gave an example of a very simple context-free grammar for English noun phrases. In Figure A-1, we add probabilities to the rules of the same grammar to produce a PCFG.

Formal Definition

Formally, a PCFG G consists of:

- a set of terminals $V = \{w^1, \ldots, w^{|V|}\}$;
- a set of nonterminals $\{N^1, \ldots, N^n\}$;
- a set of rules $\{r^{ij}\}$ for each nonterminal N^i; and,
- a designated start symbol N^1.

A rule r^{ij} maps the head nonterminal N^i to a sequence of terminals and nonterminals ξ^{ij} with a probability γ^{ij}, and is written in the form:

$$\gamma^{ij} \;\; N^i \mapsto \xi^{ij} \quad .$$

The expansion probability γ^{ij} of a rule r^{ij} is the likelihood $\mathrm{P}(r^{ij} \,|\, N^i)$ that the rule is chosen given the head nonterminal N^i. Thus, the expansion probabilities for all the rules for a nonterminal must sum to one:

$$\forall N^i \in G \; \sum_j \gamma^{ij} = 1 \quad .$$

As the terms suggest, a nonterminal can be expanded, or "rewritten", using its rules, but a terminal is a primitive symbol which cannot be further rewritten. We also use the term *preterminal* for a nonterminal which can only be rewritten into single nonterminals using unary rules.

For example, in the grammar in Figure A-1, the start symbol is NP; the nonterminals are NP, ART, ADJ, and N; the preterminals are ART, ADJ, and N; and the terminals are *the*, *big*, *red*, *barn*, and *ball*.

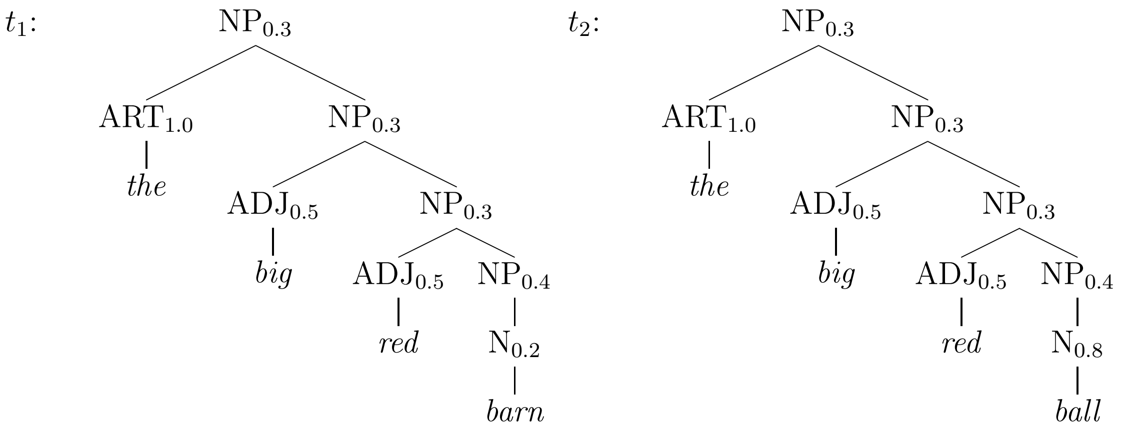


Figure A-2: Probabilistic parse trees for the English noun phrases *the big red barn* and *the big red ball*.

Chomsky Normal Form

A context-free grammar is in *Chomsky Normal Form* (CNF) if each rule in the grammar takes one of two forms:

$$N^i \mapsto N^j N^k \qquad \text{or} \qquad N^i \mapsto w^j$$

i.e., if the right hand side of the rule consists of exactly two nonterminals or exactly one terminal. It can be proved that any CFG can be written as a weakly equivalent CFG in Chomsky Normal Form.[1]

Parse Trees

A PCFG is a generative model for a language. To generate new strings with a PCFG, first write down the designated start symbol N^1. Then recursively rewrite each nonterminal using a rule randomly chosen according to the expansion probabilities on the rules of that nonterminal. The string is complete when only terminals remain. A string of terminals is written $w_1 \dots w_m$ and abbreviated w_{1m}.

The recursive rewriting of nonterminals can be represented structurally as a parse tree t, in which the internal nodes are nonterminals and the leaves are terminals. There is also some "bookkeeping" information stored in the tree during the generation or parsing process. In particular, each nonterminal in the tree is subscripted with the probability of the rule that was chosen to expand it. Figure A-2 shows two examples of parse trees that could have been generated by the grammar in Figure A-1.

We say that a tree *yields* the string of terminals at its leaves. We also say that each internal nonterminal N^i *covers* the terminals w_{jk} at the leaves of the subtree of which it is root; this is written as $N^i \overset{*}{\Rightarrow} w_{jk}$.

[1] Two grammars G_1 and G_2 are *weakly equivalent* if they both generate the same language, (with the same probabilities on strings for stochastic equivalence). Two grammars are *strongly equivalent* if they also assign strings the same tree structures (with the same probabilities, for the stochastic case). [23]

A PCFG defines at least one tree structure over every string in the language. Furthermore, a PCFG defines a probability distribution over all possible parse trees. The probability of a tree is calculated by taking the product over the expansion probabilities of all rules used—essentially, the probabilities on all internal nodes in the tree. A simple product is possible because of the aggressive context-free assumptions of PCFGs, which enforce the conditional independence of a nonterminal from all ancestors and siblings given its parent, and from all descendents given its children.

For example, in Figure A-2 the probabilities of t_1 and t_2 can be calculated as:

$$
\begin{aligned}
\mathrm{P}(t_1) &= 0.3 \times 1.0 \times 0.3 \times 0.5 \times 0.3 \times 0.5 \times 0.4 \times 0.2 \\
&= 0.00054 \\
\mathrm{P}(t_2) &= 0.3 \times 1.0 \times 0.3 \times 0.5 \times 0.3 \times 0.5 \times 0.4 \times 0.8 \\
&= 0.00216 \ .
\end{aligned}
$$

Note that we computed $\mathrm{P}(t_1)$ rather than P(*the big red barn*). Often a PCFG will be *ambiguous*, meaning that it defines more than one valid tree structure for a single string. Thus the total probability of a string w_{1m} must be calculated as the sum over the probabilities of all the possible parse trees t that yield that string:

$$
\begin{aligned}
\mathrm{P}(w_{1m}) &= \sum_t \mathrm{P}(w_{1m}, t) \\
&= \sum_{\{t:\mathrm{yield}(t)=w_{1m}\}} \mathrm{P}(t) \ .
\end{aligned}
$$

A.1.2 Parsing in PCFGs: The Inside Algorithm

Given a PCFG G and a string w_{1m}, we naturally might want to find:

- the total probability of w_{1m} given G; and
- the most likely parse tree of w_{1m} given G.

Naïvely, we might attempt both tasks by explicitly searching over all possible parse trees that yield the string w_{1m}, and then either summing or maximizing over the associated probabilities. However, in general the number of possible parse trees for an arbitrary string will be exponential in the length of the string, so this approach is intractable.

Instead, we present a well-known dynamic programming method called the *inside algorithm*, which efficiently calculates the probability of a string or finds the most likely parse without explicitly searching over trees. Rather than taking exponential time, the inside algorithm takes only $O(m^3n^3)$ time, where m is the length of the string and n is the number of nonterminals in the grammar.

In this section, we assume all PCFGs are in Chomsky Normal Form. Also, given a parse tree that covers a string w_{1m}, we use the notation N^j_{pq} to denote a subtree of the tree rooted by the nonterminal N^j that covers the substring w_{pq}.

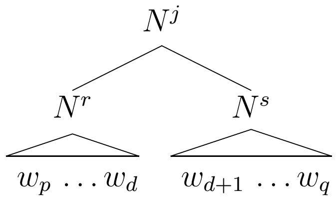


Figure A-3: A depiction of an inductive step in the calculation of the inside probability of a substring w_{pq}.

Calculating the Probability of a String

The *inside probability* $\beta_j(p, q)$ is the total probability of generating the substring $w_p \dots w_q$ given that we begin with the nonterminal N^j. We define the inside probability β of a string w_{1m} as:[2]

$$\begin{aligned} \mathrm{P}(w_{1m} \mid G) &= \mathrm{P}(N^1 \stackrel{*}{\Rightarrow} w_{1m} \mid G) \\ &= \mathrm{P}(w_{1m} \mid N^1_{1m}, G) \\ &= \beta_1(1, m) \ . \end{aligned}$$

We calculate the inside probability of a substring by induction on the length of the substring:

Base case: We want to find $\beta_j(k, k)$, which is simply the probability that a rule of the form $N^j \mapsto w_k$ exists:

$$\begin{aligned} \beta_j(k, k) &= \mathrm{P}(w_k \mid N^j_{kk}, G) \\ &= \mathrm{P}(N^j \mapsto w_k \mid G) \ . \end{aligned}$$

Induction: We want to find $\beta_j(p, q)$ for $p < q$. Because we are assuming that G is in Chomsky Normal Form, we know the first rule must be of the form $N^j \mapsto N^r N^s$, so we can simply search over the possible ways to divide the string w_{pq} into two substrings w_{pd} and $w_{(d+1)q}$, and then sum the recursive result of the algorithm on each substring. A visual summary of the inductive step is given in Figure A-3.

For all nonterminals N^j, and for all possible substrings w_{pq} where $1 \leq p < q \leq m$, we calculate the inside probability $\beta_j(p, q)$ at this level of the induction. First, we search over possible rules to expand the N^j and possible division indices d for the substring w_{pq}. We then use the chain rule for probability to break up the large joint expression into smaller conditional expressions. Finally, we exploit the context-free

[2] The inside probability of a string is calculated bottom up. There is also the concept of an outside probability $\alpha_j(p, q)$, which is calculated top down and is defined as the total probability of beginning with the start symbol N^1 and generating the nonterminal N^j_{pq} and all the words outside $w_p \dots w_q$. For the purposes of this thesis, we shall be focusing on bottom-up parsing, so inside probabilities will be sufficient.

assumptions of PCFGs to simplify these expressions and perform the inductive step:

$$\begin{aligned}
\beta_j(p,q) &= \mathrm{P}(w_{pq} \,|\, N^j_{pq}, G) \\
&= \sum_{r,s} \sum_{d=p}^{q-1} \mathrm{P}(w_{pd}, N^r_{pd}, w_{(d+1)q}, N^s_{(d+1)q} \,|\, N^j_{pq}, G) \\
&= \sum_{r,s} \sum_{d=p}^{q-1} \mathrm{P}(N^r_{pd}, N^s_{(d+1)q} \,|\, N^j_{pq}, G)\, \mathrm{P}(w_{pd} \,|\, N^j_{pq}, N^r_{pd}, N^s_{(d+1)q}, G) \\
&\qquad\qquad \times \mathrm{P}(w_{(d+1)q} \,|\, N^j_{pq}, N^r_{pd}, N^s_{(d+1)q}, w_{pd}, G) \\
&= \sum_{r,s} \sum_{d=p}^{q-1} \mathrm{P}(N^r_{pd}, N^s_{(d+1)q} \,|\, N^j_{pq}, G)\, \mathrm{P}(w_{pd} \,|\, N^r_{pd}, G) \\
&\qquad\qquad \times \mathrm{P}(w_{(d+1)q} \,|\, N^s_{(d+1)q}, G) \\
&= \sum_{r,s} \sum_{d=p}^{q-1} \mathrm{P}(N^j \mapsto N^r N^s)\beta_r(p,d)\beta_s(d+1,q) \quad .
\end{aligned}$$

Thus the inside probability of a string can be efficiently calculated bottom up, starting with the base case and then recursively applying the inductive step.

Finding the Most Likely Parse

Intuitively, we can find the most likely parse tree for a string with a few simple extensions to the inside algorithm. First, we take the max rather than the sum over rules and division indices. Second, we record the rule and division index associated with each max and use these to reconstruct the most likely tree at the end.[3]

To implement the dynamic programming, we define an *accumulator* $\delta_i(p,q)$, which stores the highest inside probability of a subtree N^i_{pq}. We also use a *backtrace* $\psi_i(p,q)$ to store the rule and division index corresponding to the highest probability subtree at any step. Then the algorithm proceeds inductively much like above:

Base case:

$$\delta_i(p,p) = \mathrm{P}(N^i \mapsto w_p)$$

Induction:

$$\delta_i(p,q) = \max_{\substack{1 \le j,k \le n \\ p \le r < q}} \mathrm{P}(N^i \mapsto N^j N^k)\delta_j(p,r)\delta_k(r+1,q)$$

Store the backtrace:

$$\psi_i(p,q) = \operatorname*{argmax}_{(j,k,r)} \mathrm{P}(N^i \mapsto N^j N^k)\delta_j(p,r)\delta_k(r+1,q)$$

Tree readout: By construction, we know that the most likely parse tree of the string

[3]The resulting algorithm is similar to the Viterbi algorithm for finding the most likely path through a Hidden Markov Model (HMM).

w_{1m} rooted at the start symbol N^1 is given by $\mathrm{P}(\hat{t}) = \delta_1(1, m)$, and that the root node of $\hat{t}$ must be N^1_{1m}. Then we use the backtrace to recursively construct the left and right child nodes of each internal node, in a top down manner. If we would like to find the children nodes of a node N^i_{pq}, and we have $\psi_i(p, q) = (j, k, r)$, then:

$$\begin{aligned} \text{leftchild}(N^i_{pq}) &= N^j_{pr} \\ \text{rightchild}(N^i_{pq}) &= N^k_{(r+1)q} \quad . \end{aligned}$$

The readout process continues recursively until all the leaf nodes are nonterminals, when the most likely parse tree is complete.

A.1.3 Sources and Related Work

There are a number of excellent references on PCFGs and parsing in natural language processing. James Allen's book *Natural Language Understanding* [1] covers CFGs, PCFGs, and several approaches to parsing, including top-down, bottom-up, chart parsing, and best-first parsing. Jurafsky and Martin, in *Speech and Language Processing* [19], also present CFGs and PCFGs, and discuss the Earley algorithm for parsing with dynamic programming in CFGs and the efficient CYK algorithm for parsing in PCFGs.

Finally, Manning and Schütze's book *Foundations of Statistical Natural Language Processing* [23] offers extensive coverage of PCFGs, and a thorough treatment of the inside algorithm we presented in this section. They also discuss general issues of parsing in PCFGs, and evaluation of parsing methods.

A.2 Representing Rotation with Quaternions

In Chapter 1 we described the motivation for a three-dimensional approach to object recognition. However, three-dimensional models require a method for handling three-dimensional rotations. The PGG framework uses unit quaternions to represent rotation, but there are a variety of options, so the choice must be considered thoroughly.

In this section we will motivate the use of unit quaternions, and offer a brief introduction to the mathematics of quaternions for rotation. We will then show how to define and learn a Gaussian distribution over rotations represented as unit quaternions, and finally describe how to sample uniformly from the space of unit quaternions.

A.2.1 Representation Choices for Rotation

Euler's theorem states that:

> Every displacement (or orientation with respect to a fixed frame) of a rigid body can be described as a rotation by some angle θ around some fixed axis $\hat{\mathbf{n}}$. [17]

Although this offers a simple and elegant mathematical representation for rotations, it does not necessarily suggest the best representation from a computational point of view.

There are four standard choices for the computational representation of rotations in three-dimensional Euclidean space $\mathbb{R}^3$:

- the **coordinate matrix** representation, in which a rotation is represented as a member of $\mathbf{SO}(3)$, the group of *special orthogonal* 3 by 3 matrices;

- the **axis-angle** representation, in which a rotation is represented as a pair of the form $(\hat{\mathbf{n}}, \theta)$ which directly corresponds to Euler's theorem above;

- the **Euler angles** representation, in which a rotation is represented as a sequence of three rotations $(\theta_1, \theta_2, \theta_3)$ around the principal orthogonal axes of the coordinate frame $(\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}})$; and

- the **quaternion** representation, in which a rotation is represented as a point on the four-dimensional unit hypersphere, or equivalently as a complex number with one real and three imaginary components $q = w + x\hat{\mathbf{i}} + y\hat{\mathbf{j}} + z\hat{\mathbf{k}}$.

Each of these representations has advantages and disadvantages.

Coordinate Matrix

Coordinate matrices leverage basic knowledge about linear algebra and are used extensively throughout computer vision and graphics; hence the matrix representation has familiarity on its side. The representation also makes it easy to map a point from one coordinate frame to another, using simple matrix multiplication. Yet another benefit in using this representation is the one-to-one mapping between members of the space of all rotations and members of the space of coordinate matrices. In fact, the mathematical group $\mathbf{SO}(3)$ is used to refer to both spaces.

However, coordinate matrices can be difficult to work with in a computational setting because they are inefficient and redundant (nine parameters are required to represent three degrees of freedom), and because a rotation is hard to visualize when written in matrix form. The most important drawback, however, is that it is very difficult to define a distribution over the space of coordinate matrices because $\mathbf{SO}(3)$ is not a vector space. This means that linear superpositions of elements are not necessarily closed under the space (see Johnson [17] for a more formal definition), so many familiar mathematical operations, such as basic linear interpolation, are not valid without modification.[4]

[4] Of course, since we have stated that $\mathbf{SO}(3)$ is actually the group of all rotations in three-dimensional Euclidean space itself, the fact that it is not a vector space is relevant in all these representations. Quaternions are the preferred choice of representation precisely because they offer a reasonable way of dealing with the non-vectorness of $\mathbf{SO}(3)$.

Axis-Angle

The axis-angle representation seems appealing because it directly corresponds to Euler's theorem, but as we have already suggested, it is mathematically elegant but computationally quite inconvenient. There are several types of inherent redundancy in the formalism; for example, the identity rotation can be represented by a rotation of zero around *any* axis $\hat{\mathbf{n}}$.

In order to compensate for some of this redundancy, the axis $\hat{\mathbf{n}}$ is often constrained to be of unit length, and the angle θ limited to the range $-\pi$ to π or 0 to 2π. This in turn, however, introduces discontinuities into the representation that make interpolation (and thus defining distributions) extremely difficult. As Johnson [17] says:

> If we choose to keep the angle in a fixed range, the interpolation cannot be continuous—at some point it needs to "jump" through the boundary from $-\pi$ to π or from 0 to 2π. These discontinuities wreak havoc on most interpolation and numerical integration schemes that are unaware of them. [17]

Euler Angles

Euler angles have historically been used in physics and animation because they are intuitively easy for a human user to visualize and define. They also appear to be efficient because they require only a minimal three parameters to specify three degrees of freedom. However, the minimal representation means they have a major inherent singularity—one that actually exists in any rotation representation with only three parameters. If, during the three rotations around the axes, one of the three axes (e.g., $\hat{\mathbf{x}}$) aligns with the original direction of either of the other two axes (e.g., $\hat{\mathbf{z}}$), any configuration produced by rotating around the axis pointing in this direction ($\hat{\mathbf{x}}$) could have produced by simply rotating around the original axis that pointed in that direction ($\hat{\mathbf{z}}$). Thus a degree of freedom has been lost. This singularity (which is a singularity of the representation, not of the space it represents) is often called *gimbal lock*, and interpolation through this singularity is very difficult.

Another way to think about this singularity is as a "factorization of $\mathbf{SO}(3)$". Again, Johnson describes this well:

> If you rotate something around $\hat{\mathbf{x}}$ and then around $\hat{\mathbf{y}}$, there will always be a component of $\hat{\mathbf{z}}$ in the result. [17]

Euler angles are an attempt to factor 3D rotations into three independent 1D rotations, but in fact this is not a mathematically accurate representation and thus will *always* have problems.

If an application can afford to maintain special cases for the inherent singularities, then Euler angles provide an adequate representation. But for the purposes of this research, the singularities of the Euler angle representation are problematic. Thus we turn to our fourth and best option for the representation of rotations: quaternions.

A.2.2 An Introduction to Quaternions

The Irish mathematician Sir William Rowan Hamilton discovered quaternions while walking home from work in Dublin on October 16, 1843. He had been searching for a way to extend ordinary complex numbers of the form $a + bi$ to higher dimensions (e.g., $a + bi + cj$), but he could not understand how to create a consistent number system with only one real term and two imaginary terms.

Complex numbers, when written in polar form, can be thought of as a point in a two-dimensional plane, represented as a length r and an angle θ, and thus multiplication of complex numbers acts as a scaling and rotation around the origin of that plane. In three dimensions, however, two numbers are required to specify an axis (of unit length) around which to rotate, one is required to specify the angle of rotation, and a *fourth* is required to specify the scaling factor. [7]

While walking that day, Hamilton realized that four dimensions were required, instead of three, and was so excited that he inscribed the discovery into a rock. The site of his discovery can still be seen in Dublin today.[5]

Basic Quaternion Algebra

A quaternion is a "hypercomplex" number—it has one real component and three imaginary components, while standard complex numbers have only one of each. A quaternion q can be written as:[6]

$$q = w + xi + yj + zk$$

such that the coefficients are real:

$$w, x, y, z \in \mathbb{R} \quad .$$

and the imaginary numbers can be combined as with normal complex numbers:

$$i^2 = j^2 = k^2 = ijk = -1$$

and

$$\begin{aligned} ij &= -ji = k \\ jk &= -kj = i \\ ki &= -ik = j \quad . \end{aligned}$$

[5] Lest the reader be confused—as the author was—that four numbers should be required to specify a rotation that has only three degrees of freedom, note that only unit quaternions are necessary to represent rotation, and therefore only three degrees of freedom are actually being expressed. This is explained in further detail below.

[6] Although it is standard with ordinary complex numbers as well as quaternions, the use of a + to denote the combination of real and imaginary terms is misleading—it would be more accurate to write them as an ordered tuple of the real and imaginary components. However, we use the standard notation for consistency.

A quaternion can also be thought of as a pair of a real scalar w and a real vector $\mathbf{v} \in \mathbb{R}^3$:

$$q = w + \mathbf{v}$$

or expanded out on the imaginary axes:

$$q = w + x\hat{\mathbf{i}} + y\hat{\mathbf{j}} + z\hat{\mathbf{k}} \quad .$$

As Johnson does, we use $\mathbb{H}$ to denote the quaternion group.

The conjugate of a quaternion is formed by negating the imaginary part:

$$q^* = w - \mathbf{v} \quad .$$

The magnitude (also called "norm") of a quaternion is found by multiplying it with its conjugate:

$$\begin{aligned} |q| &= qq^* \\ &= q^*q \\ &= w^2 + \mathbf{v} \cdot \mathbf{v} \quad . \end{aligned}$$

A unit quaternion has a magnitude of one, and is denoted with a hat, $\hat{q}$. The subgroup of unit quaternions is written as $\hat{\mathbb{H}}$.

Just as with ordinary complex numbers, addition of quaternions is performed by summing corresponding components. The quaternion group $\mathbb{H}$ is closed under addition, but the subgroup of unit quaternions $\hat{\mathbb{H}}$ is not.

Multiplication of quaternions is performed with a Cartesian product of the quaternions as if they are polynomials in terms of the imaginary numbers:

$$\begin{aligned} q_1 q_2 = \big(&(w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2) + \\ &(y_1 z_2 - y_2 z_1 + w_1 x_2 + w_2 x_1) i + \\ &(x_2 z_1 - x_1 z_2 + w_1 y_2 + w_2 y_1) j + \\ &(x_1 y_2 - x_2 y_1 + w_1 z_2 + w_2 z_1) k \big) \end{aligned}$$

or in the vector notation:

$$q_1 q_2 = (w_1 w_2 - \mathbf{v}_1 \cdot \mathbf{v}_2, \mathbf{v}_1 \times \mathbf{v}_2 + w_1 \mathbf{v}_2 + w_2 \mathbf{v}_1) \quad .$$

Quaternion multiplication is associative but *not* commutative, which we would expect for a formalism that can represent rotations.[7]

The quaternion inverse is taken as:

$$q^{-1} = \frac{1}{|q|} q^* \quad .$$

[7] The non-commutativity of quaternion multiplication is important in Section 2.4, when we derive a conditional multivariate Gaussian distribution over quaternions.

The inverse of a unit quaternion is just its conjugate:

$$\hat{q}^{-1} = \hat{q}^* \quad .$$

Representing Rotation: The Polar Form

A quaternion can be written in its polar form as:

$$q = re^{\hat{\mathbf{n}}\frac{\theta}{2}} = r\left[\cos\frac{\theta}{2} + \hat{\mathbf{n}}\sin\frac{\theta}{2}\right]$$

where r is called the magnitude, $\frac{\theta}{2}$ is called the angle, and $\hat{\mathbf{n}}$ is called the axis and is a 3-vector of unit length, or equivalently, a "pure" unit quaternion (one that has a zero real component).

For a unit quaternion the magnitude r is simply one. Thus a unit quaternion written as

$$\hat{q} = e^{\hat{\mathbf{n}}\frac{\theta}{2}} = \cos\frac{\theta}{2} + \hat{\mathbf{n}}\sin\frac{\theta}{2}$$

expresses a rotation in three-dimensional space of the angle θ around the unit axis $\hat{\mathbf{n}}$.

To rotate a vector $\mathbf{x} \in \mathbb{R}^3$ by a unit quaternion $\hat{q}$, pretend that $\mathbf{x}$ is actually a pure quaternion, and perform the following quaternion product:

$$\boxed{\mathbf{x}' = \hat{q}\mathbf{x}\hat{q}^{-1}}$$

The product $\mathbf{x}'$ will always be a pure quaternion (and thus a 3-vector in $\mathbb{R}^3$) if $\mathbf{x}$ is pure and $\hat{q}$ is unit. And, most importantly for our purposes, $\mathbf{x}'$ *will be the result of rotating* $\mathbf{x}$ *by* θ *radians around the axis* $\hat{\mathbf{n}}$.

A rotation of θ radians around the axis $\hat{\mathbf{n}}$ is equivalent to a rotation of $-\theta$ radians around the axis $-\hat{\mathbf{n}}$. Thus a unit quaternion $\hat{q}$ and its negation $-\hat{q}$ represent the same rotation, and the quaternion group provides "double coverage" of the space of possible rotations in three dimensions. The resulting symmetry of the space is called *antipodal symmetry.*

Just as with coordinate matrices, multiplying two unit quaternions that each represent a rotation produces a new unit quaternion that represents the composition of the two rotations. We can also find the shortest rotation that will rotate an orientation $\hat{q}_1$ into another orientation $\hat{q}_2$ by taking the product $\hat{q}_1^*\hat{q}_2$; this is somewhat like a (non-commutative) "subtraction" operation for rotations, so we shall refer to the resulting quaternion as the *rotational difference* between $\hat{q}_1$ and $\hat{q}_2$.

Tangent Spaces on the Hypersphere

As we said above, a unit quaternion $\hat{q} \in \hat{\mathbb{H}}$ can be thought of a 4-vector with unit magnitude. Thus the unit quaternion group can be represented as the surface of a four-dimensional unit hypersphere. The surface of the hypersphere has three degrees of freedom, although it is embedded in four-dimensional space, so the hypersphere is often written as S^3. Thus, this representation makes clear the fact that, although quaternions have four parameters, unit quaternions use only three degrees of freedom to represent rotations in three dimensions. It also makes it easy to visualize the antipodal symmetry of the unit quaternion space—points on opposite sides of the hypersphere (i.e., the two points where a line through the origin intersects the surface) represent the same rotation.

The hypersphere representation will also be helpful computationally—thinking of unit quaternions as points on the surface of a unit four-dimensional hypersphere provides a geometric way to visualize and analyze algorithms using quaternions. In particular, two important operations, the exponential map and the logarithmic map, are much easier to visualize using the hypersphere representation for $\hat{\mathbb{H}}$.

The exponential map provides a way to convert any arbitrary 3-vector in $\mathbb{R}^3$ into a unit quaternion:

$$\hat{q} = e^{\mathbf{x}} = e^{\frac{\theta}{2}\hat{\mathbf{n}}} \quad .$$

Its inverse operation, the logarithmic map, maps any unit quaternion into a 3-vector in $\mathbb{R}^3$:

$$\boxed{\ln \hat{q} = \ln e^{\frac{\theta}{2}\hat{\mathbf{n}}} = \frac{\theta}{2}\hat{\mathbf{n}}}$$

This can be implemented computationally using the "rectangular", rather than polar, form of quaternion $\hat{q} = w + \mathbf{v}$ as:

$$\ln \hat{q} = \frac{\mathbf{v}}{\operatorname{sinc}\left(\frac{\theta}{2}\right)}$$

where

$$\frac{\theta}{2} = \arccos(w)$$

and sinc is the "sink" function $\sin(x)/x$ whose limit at $x = 0$ exists.

We can think of the logmap $\ln \hat{q}$ of the unit quaternion $\hat{q}$ as living in the 3-dimensional tangent space at the identity of the quaternion hypersphere S^3. We can choose an arbitrary location $\hat{p}$ on the hypersphere for this tangent space by rotating the sphere to align the chosen location with the identity before taking the log:

$$\ln_{\hat{p}}(\hat{q}) = \ln(\hat{p}^* \hat{q}) \quad .$$

The subscript $\hat{p}$ denotes the "tangent point"—the location on the hypersphere for the tangent space. We will write the tangent space at a point $\hat{p}$ as $\tan_{\hat{p}}(S^3)$; we will omit the subscript when the tangent point is the identity.

The logarithmic mapping is invertible, such that:

$$\exp_{\hat{p}}\left(\frac{\theta}{2}\hat{\mathbf{n}}\right) = \hat{p} e^{\frac{\theta}{2}\hat{\mathbf{n}}} \quad .$$

The ability to perform the logmap at an arbitrary point on the hypersphere is important since the logmap is effectively a local linearization, so it is best nearest the center of the map.

Thus we have an important result: the tangent space of the quaternion hyper-

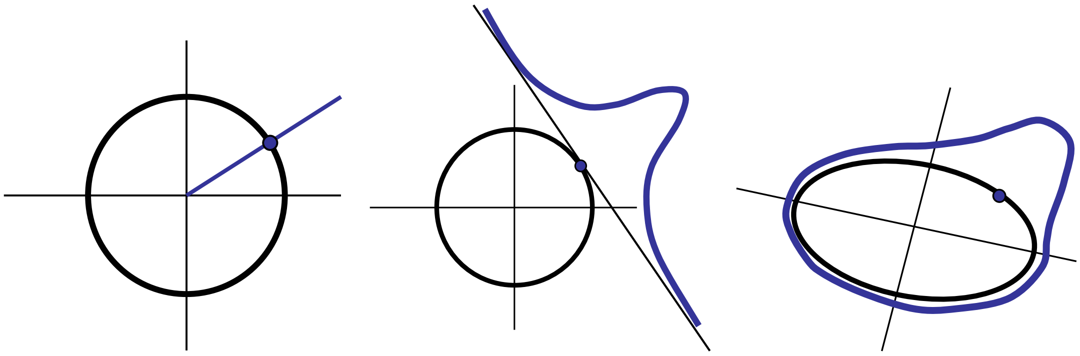


Figure A-4: In a Gaussian distribution over quaternions, the distribution is defined in the three-dimensional tangent space to the four-dimensional unit hypersphere at the mean, and then the tails of the distribution are "wrapped" back onto the hypersphere to produce a spherical distribution. Here, for illustrative purposes, the 4D hypersphere is depicted as a 2D circle and the 3D tangent space as a 1D tangent line. We also ignore the other peak of the bimodal distribution. [22]

sphere is in $\mathbb{R}^3$ and, crucially, it is a vector space. The tangent space also has the following useful properties:

1. The distance from a point to the identity on the hypersphere is preserved in the tangent space; and

2. The angle between two quaternions and the identity is preserved as the angle between the mapped vectors in the tangent space.

This means that an ellipse around the tangent point on the hypersphere will be preserved as an ellipsoid in the tangent space. We will exploit these facts below in order to define a Gaussian distribution over quaternions.

A.2.3 Gaussian Distributions Over Unit Quaternions

In Chapter 2, we show how the geometric models of a PGG take the form of multivariate Gaussian distributions over the space of object parts. In this section, we will lay the first building block for PGG geometric models by discussing Gaussian distributions over rotations that are represented as quaternions.

As before, we refer to Johnson [17] for help with quaternions. He describes how to model and estimate Gaussian distributions over unit quaternions, in a manner that intuitively parallels Gaussians over real numbers.

Intuitive Approach

Here, thinking of quaternions as points on the four-dimensional unit hypersphere S^3 is important. We define the Gaussian distribution in the tangent space at the mean $\tan_{\hat{\mu}}(S^3)$, and then "wrap" the tails of the distribution onto the quaternion hypersphere, using the exponential map, to produce a spherical distribution. See Figure A-4 for a depiction of this concept, where the 4D hypersphere is simplified to a 2D circle.

To understand why this approach produces a reasonable result, recall that it is an important property of tangent spaces of the quaternion hypersphere that ellipses around the tangent point on the hypersphere are preserved as ellipsoids in the tangent space. This means that the isocontours of a distribution we define in the tangent space will be preserved when mapped onto the hypersphere, so the distribution will provide meaningful density values for query points.

Because of the antipodal symmetry of the quaternion hypersphere, learning a wrapped Gaussian distribution over quaternions will actually produce a *bimodal* distribution, with peaks at each pole of the hypersphere (where the mean quaternion and its negation serve as the "north" and "south" poles). To handle this symmetry, we learn a wrapped Gaussian distribution over only one hemisphere of the hypersphere. We then flip any query or example quaternion to the appropriate hemisphere before using the distribution.

Formal Definition

Now, we consider how to mathematically express and calculate a Gaussian over quaternions. Recall that the probability density function for Gaussian distributed multivariate vectors is written as:[8]

$$\mathrm{p}(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{n/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\} \tag{A.1}$$

where $\mathbf{x}$ is the query vector, $\boldsymbol{\mu}$ is the mean vector, $\boldsymbol{\Sigma}$ is the covariance matrix, and n is the dimensionality of the vector space. We can define a Gaussian distribution on unit quaternions in terms of this basic definition on multivariate vectors by using the exponential map.

The mean is simply a unit quaternion, $\hat{\mu}$. The covariance matrix $\boldsymbol{\Sigma}$ is defined in the tangent space at the mean, because as we discussed above, $\tan_{\hat{\mu}}(S^3)$ is a vector space, so vector multiplication is well defined. The dimensions of $\boldsymbol{\Sigma}$ are 3×3 for the univariate quaternion case, because the logarithmic map converts a single quaternion to a 3-vector in $\mathbb{R}^3$. Like all covariance matrices, $\boldsymbol{\Sigma}$ is symmetric and positive definite.

The deviation quantity $(\mathbf{x} - \boldsymbol{\mu})$ in the original function should be the "distance" of the query quaternion $\hat{q}$ from the mean quaternion $\hat{\mu}$. It can be computed on the hypersphere as the rotational difference between the two quaternions, and then projected into the tangent space using the logarithmic map. This allows the tangent space to be located at the mean, so that the covariance is for a zero-mean normal distribution and distortion is minimized. Following Johnson, we will call the result of

[8] We use a lowercase p to denote a continuous probability density, rather than the uppercase P for discrete probabilities such as those we encountered in Section A.1.

projecting the deviation into the tangent space at the mean the *mode-tangent* vector:

$$\mathbf{m} = \ln(\hat{\mu}^* \hat{q}) \ .$$

The quantity $\dfrac{1}{(2\pi)^{n/2}|\mathbf{\Sigma}|^{1/2}}$ is a normalizing constant, necessary to ensure that the density function integrates to one over S^3. It is computed entirely in the tangent space, and, just as with standard multivariate Gaussians, we can calculate it in advance because it does not refer to the query quaternion $\hat{q}$ at all. For the case of a Gaussian over single quaternions, we have that $n = 3$.

Thus we can define a Gaussian distribution over unit quaternions in terms of the mode-tangent as:

$$\mathrm{p}(\hat{q}) = \mathcal{N}(\hat{q}; \hat{\mu}, \mathbf{\Sigma}) = \frac{1}{(2\pi)^{3/2}|\mathbf{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}\mathbf{m}^T \mathbf{\Sigma}^{-1} \mathbf{m}\right\}$$

or using quaternion algebra:

$$\boxed{\mathrm{p}(\hat{q}) = \mathcal{N}(\hat{q}; \hat{\mu}, \mathbf{\Sigma}) = \frac{1}{(2\pi)^{3/2}|\mathbf{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}\ln(\hat{\mu}^* \hat{q})^T \mathbf{\Sigma}^{-1} \ln(\hat{\mu}^* \hat{q})\right\}} \ . \tag{A.2}$$

A.2.4 Parameter Estimation in Gaussians Over Unit Quaternions

One of the benefits of using a Gaussian distribution is the ability to learn its parameters from example data in an efficient closed-form manner. In this section we discuss how to estimate the maximum likelihood parameters $\hat{\mu}$ and $\mathbf{\Sigma}$ of a Gaussian distribution over quaternions from a set of sample quaternions.

Estimating the Mean

Before we show how to estimate the quaternion mean of a set of quaternions, consider why we cannot use the familiar sample mean:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \ .$$

First, the sample mean does not take into account the antipodal symmetry of the quaternion hypersphere at all. This is clearly problematic, as we would like two opposite quaternions $\hat{q}$ and $-\hat{q}$ to be considered equal, but this equation will average them to zero. Second, the sample mean produced by this equation for an arbitrary set of quaternions will not necessarily lie on the hypersphere, and renormalizing the resulting quaternion to unit length will not solve the first problem of antipodal quaternions.

Instead, Johnson proposes finding the unit quaternion that minimizes the squared distances between it and all the quaternion data points [17]. The distance function we will minimize is defined between two quaternions, and gives the distance from the first argument to the closer of the two possible signs of the second, according to the "rotational difference" distance metric we defined above; i.e.,

$$\operatorname*{dist}_{|S^3|}(\hat{p}, \hat{q}) = \min_{\hat{a}=\hat{q},-\hat{q}} \operatorname*{dist}_{S^3}(\hat{p}, \hat{a})$$

where

$$\operatorname*{dist}_{S^3}(\hat{q}_i, \hat{q}_j) = \|\ln(\hat{q}_i^* \hat{q}_j)\| \ .$$

Say that our data is a $4 \times N$ matrix $\mathbf{Q}$ of unit quaternions $\hat{q}_i$ (which form the columns of $\mathbf{Q}$). Then the mean $\hat{\mu}$ is defined to be the unit quaternion $\hat{p}$ that minimizes the squared distance from $\hat{p}$ to all the data points:

$$\hat{\mu} = \operatorname*{argmin}_{\hat{p}} \sum_{i=1}^{N} \operatorname*{dist}_{|S^3|}(\hat{p}, \hat{q}_i)^2 \ .$$

Unfortunately, this function is non-linear, so a closed-form minimization is difficult. Fortunately, Johnson shows (pages 115-116, [17]) that minimizing the above expression is equivalent to minimizing:

$$\hat{\mu} = \operatorname*{argmax}_{\hat{a} \in S^3} \sum_{i=1}^{N} (\hat{a} \cdot \hat{q}_i)^2 \ .$$

Johnson presents a formal argument as well, but intuitively, this equation says that we would like to maximize the directional match between the unit quaternions, which is what the dot product does, and the square allows the sign to be ignored.

However, we also need to constrain $\hat{a}$ to be a unit quaternion, so we add a Lagrange multiplier:

$$E(\hat{p}) = \sum_{i=1}^{N} (\hat{p} \cdot \hat{q}_i)^2 + \lambda\big((\hat{p} \cdot \hat{p})^2 - 1\big) \ .$$

Switching to vector notation in $\mathbb{R}^4$, we have:

$$\begin{aligned} E(\mathbf{p}) &= \sum_{i=1}^{N} (\mathbf{p}^T \mathbf{q}_i)^2 + \lambda(\mathbf{p}^T \mathbf{p} - 1) \\ &= \mathbf{p}^T \Big(\sum_{i=1}^{N} \mathbf{q}_i^T \mathbf{q}_i \Big) \mathbf{p} + \lambda(\mathbf{p}^T \mathbf{p} - 1) \ . \end{aligned}$$

We defined $\mathbf{Q}$ to be the $4 \times N$ data matrix, so we can say:

$$E(\mathbf{p}) = \mathbf{p}^T \mathbf{Q} \mathbf{Q}^T \mathbf{p} + \lambda(\mathbf{p}^T \mathbf{p} - 1) \ .$$

Then to minimize this expression, we take the derivative and set it equal to zero:

$$\frac{d}{d\mathbf{p}} E(\mathbf{p}) = 2\mathbf{Q}\mathbf{Q}^T \mathbf{p} + 2\lambda \mathbf{p} = 0 \ .$$

Now, it is simply an eigenvector problem:

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \quad .$$

where $\mathbf{x} = \mathbf{p}$ and $\mathbf{A} = \mathbf{Q}\mathbf{Q}^T$, with dimensions 4×4. The eigenvectors and eigenvalues can be found using Singular Value Decomposition (SVD), and the maximum value $\mathbf{p}$ of the expression is then the eigenvector (or possibly its negation) associated with the maximum eigenvalue of $\mathbf{A}$. Because $\mathbf{A}$ has constant dimensions of 4×4, SVD is not computationally expensive.

Johnson summarizes the mean estimation algorithm as follows [17]:

1. Let $\mathbf{q}_i$ be the column vector representation of the unit quaternion sample $\hat{q}_i$.
2. Let $\mathbf{Q}$ be the $4 \times N$ data matrix, where the ith column $\mathbf{q}_i$ is the ith sample.
3. Let $\mathbf{A} = \mathbf{Q}\mathbf{Q}^T$.
4. Let $\mathbf{e}_i$ be an eigenvector of $\mathbf{A}$ with real eigenvalue a_i.
5. Choose one of the two eigenvectors $\pm\mathbf{e}_*$ associated with the maximal eigenvalue a_* as the estimate of the mean $\hat{\mu}$.

Hemispherization

We have mentioned several times that, because of the double coverage of rotations in $\mathbf{SO}(3)$ by unit quaternions on the hypersphere, a learned Gaussian distribution will actually be bimodal. For simplicity, we define the distribution to be valid over only one of the two hemispheres of the hypersphere. The choice of hemisphere is defined by the mean quaternion $\hat{\mu}$, as argued by Johnson (pages 118-119, [17]).

The mean quaternion is estimated according to the above algorithm, and either $\hat{\mu}$ or its negation $-\hat{\mu}$ is a valid choice. But, once that choice has been made, we must *hemispherize*, or flip, all the data points $\hat{q}_i$ to the appropriate hemisphere before the covariance is estimated. The choice of $\hat{q}_i$ or $-\hat{q}_i$ is made according to which one minimizes the directional match with the mean.

Thus the hemispherization algorithm is written as:

1. Let $\hat{\mu}$ be the quaternion mean of the example data.
2. For each example $\hat{q}_i$:
   (a) If $\hat{\mu} \cdot \hat{q}_i < 0$, then $\hat{q}_i \leftarrow -\hat{q}_i$.

Furthermore, any query quaternion that is encountered must also be hemispherized before its density can be calculated according to the distribution. In this way, we ensure that the distribution is in fact defined over only one of the hemispheres.

Estimating the Covariance

Once the quaternion mean has been estimated and the examples have been hemispherized according to that mean, estimating the covariance is straightforward. The estimated covariance matrix $\mathbf{\Sigma}$ is defined to be the normalized outer product of the data matrix, according to the standard maximum likelihood estimation approach for multivariate Gaussians. Note, however, that the data matrix will consists of data quaternions that have first been hemispherized, then rotated by the quaternion mean, and finally projected into the tangent space at the mean to be "linearized", using the logmap.

The covariance estimation algorithm is summarized as follows:

1. Let $\hat{\mu}$ be the quaternion mean of the example data.
2. Hemispherize the example data points, and project them into the tangent space at the mean. For each example $\hat{q}_i$:
   (a) If $\hat{\mu} \cdot \hat{q}_i < 0$, then $\hat{q}_i \leftarrow -\hat{q}_i$.
   (b) Let $\mathbf{w}_i = \ln(\hat{\mu}^* \hat{q}_i)$.
3. Let $\mathbf{W}$ be the data matrix with $\mathbf{w}_i$ as the ith column.
4. Let the estimated covariance matrix be calculated as $\mathbf{\Sigma} = \frac{1}{N-1}\mathbf{W}\mathbf{W}^T$, where N is the number of data points.

A.2.5 Sources and Related Work

Berthold Horn briefly describes quaternions for rotation in *Robot Vision* [14]. Ken Shoemake presents the algebra and calculus of quaternions and how they relate to rotations in [33]. David Eberly summarizes these ideas in [10]. Andrew Burbanks describes how the problem of extending complex numbers led Hamilton to the discovery of quaternions and then presents the fundamentals of quaternion algebra and a C++ implementation in [7].

These contributions notwithstanding, my most important source is Michael Johnson's 2003 dissertation [17]. In that document, Johnson gives an excellent and extensive description of the intuition and formal mathematics of each of the rotation representations described above and their computational advantages and disadvantages. He describes quaternions and gives a thorough motivation for their use to represent rotation. He also presents the approach to defining and learning a Gaussian distribution over quaternions for rotation that I described in this section. I am greatly indebted to Johnson's work for my understanding of this material—indeed, much of the explanation in this section is based directly upon his—and I refer the interested reader to it for more information.

A.3 Conditional Multivariate Gaussian Distributions

In this section, we discuss another important building block that we use when we develop the geometric models of a PGG in Chapter 2: conditional multivariate Gaussian distributions over $\mathbb{R}^n$. We will factor a multivariate joint Gaussian into marginal and conditional distributions, but first we must show how to block diagonalize and take the inverse of a partitioned matrix.

A.3.1 Partitioned Matrices

Consider a partitioned matrix $\mathbf{M}$:

$$\mathbf{M} = \begin{bmatrix} \mathbf{E} & \mathbf{F} \\ \mathbf{G} & \mathbf{H} \end{bmatrix}$$

where $\mathbf{E}$ and $\mathbf{H}$ are invertible. We would like to find an expression for the inverse of $\mathbf{M}$ in terms of its blocks.

We start by finding a way to *block diagonalize* the matrix $\mathbf{M}$; in other words, manipulate the partitions of $\mathbf{M}$ such that we replace $\mathbf{F}$ and $\mathbf{G}$ each with a block of zeros. Jordan [18] demonstrates that we can do this by pre- and post-multiplying the matrix by two other matrices, each which serves to zero out either $\mathbf{F}$ or $\mathbf{G}$, but which miraculously do not interfere with each other:

$$\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{G}\mathbf{E}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{E} & \mathbf{F} \\ \mathbf{G} & \mathbf{H} \end{bmatrix} \begin{bmatrix} \mathbf{I} & -\mathbf{E}^{-1}\mathbf{F} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} = \begin{bmatrix} \mathbf{E} & \mathbf{0} \\ \mathbf{0} & \mathbf{H} - \mathbf{G}\mathbf{E}^{-1}\mathbf{F} \end{bmatrix} \tag{A.3}$$

The term in the lower right of the block diagonal matrix is called the *Schur complement* of $\mathbf{M}$ with respect to $\mathbf{E}$. It can be shown to be invertible, and is written $\mathbf{M}/\mathbf{E}$:

$$\mathbf{M}/\mathbf{E} = \mathbf{H} - \mathbf{G}\mathbf{E}^{-1}\mathbf{F} \ .$$

Recall that if we have a matrix expression of the form $\mathbf{ABC} = \mathbf{D}$ and invert both sides, we obtain $\mathbf{B}^{-1} = \mathbf{C}\mathbf{D}^{-1}\mathbf{A}$. Recall also that the inverse of a block diagonal matrix is simply the diagonal matrix of the inverse of its blocks.

Now, we apply these facts to take the inverse of both sides of Equation (A.3) for the block diagonal of $\mathbf{M}$:

$$\begin{bmatrix} \mathbf{E} & \mathbf{F} \\ \mathbf{G} & \mathbf{H} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{I} & -\mathbf{E}^{-1}\mathbf{F} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{E}^{-1} & \mathbf{0} \\ \mathbf{0} & (\mathbf{M}/\mathbf{E})^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{G}\mathbf{E}^{-1} & \mathbf{I} \end{bmatrix} \tag{A.4a}$$

$$= \begin{bmatrix} \mathbf{E}^{-1} + \mathbf{E}^{-1}\mathbf{F}(\mathbf{M}/\mathbf{E})^{-1}\mathbf{G}\mathbf{E}^{-1} & -\mathbf{E}^{-1}\mathbf{F}(\mathbf{M}/\mathbf{E})^{-1} \\ -(\mathbf{M}/\mathbf{E})^{-1}\mathbf{G}\mathbf{E}^{-1} & \mathbf{M}/\mathbf{E}^{-1} \end{bmatrix} \tag{A.4b}$$

Thus we have an expression for $\mathbf{M}^{-1}$ in terms of its blocks, as desired.

If instead we take the determinant of both sides of Equation (A.3), we find:

$$|\mathbf{M}| = |\mathbf{M}/\mathbf{E}||\mathbf{E}| \tag{A.5}$$

which, as Jordan points out, makes the notation for the Schur complement very appropriate [18].

A.3.2 Marginalizing and Conditioning

We have a vector $\mathbf{x}$ of length n, and we would like to partition it into two subvectors $\mathbf{x}_1$ and $\mathbf{x}_2$, of lengths n_1 and n_2 respectively such that $n_1 + n_2 = n$:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} \quad .$$

If we have a multivariate Gaussian distribution for $\mathrm{p}(\mathbf{x})$, it is natural to ask whether we can factor it into the marginal distribution $\mathrm{p}(\mathbf{x}_1)$ and the conditional distribution $\mathrm{p}(\mathbf{x}_2 \,|\, \mathbf{x}_1)$, to exploit the fact that:

$$\begin{aligned} \mathrm{p}(\mathbf{x}) &= \mathrm{p}(\mathbf{x}_1, \mathbf{x}_2) \\ &= \mathrm{p}(\mathbf{x}_2 \,|\, \mathbf{x}_1)\, \mathrm{p}(\mathbf{x}_1) \quad . \end{aligned}$$

We begin by partitioning the mean and covariance parameters of the joint Gaussian in the same way we partitioned $\mathbf{x}$:

$$\boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix} \qquad\qquad \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix} \quad .$$

This allows us to write a joint Gaussian distribution for $\mathrm{p}(\mathbf{x}) = \mathrm{p}(\mathbf{x}_1, \mathbf{x}_2)$ in terms of the partitioned forms of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$:

$$\begin{aligned} \mathrm{p}(\mathbf{x}) &= \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) \\ &= \frac{1}{(2\pi)^{(n_1+n_2)/2} |\boldsymbol{\Sigma}|^{1/2}} \exp\left\{ -\frac{1}{2} \begin{bmatrix} \mathbf{x}_1 - \boldsymbol{\mu}_1 \\ \mathbf{x}_2 - \boldsymbol{\mu}_2 \end{bmatrix}^T \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{x}_1 - \boldsymbol{\mu}_1 \\ \mathbf{x}_2 - \boldsymbol{\mu}_2 \end{bmatrix} \right\} \end{aligned} \tag{A.6}$$

Focusing on the exponential factor first, we apply Equation (A.4a) to write out the inverse covariance matrix in terms of its blocks:

$$\begin{aligned} &\exp\left\{ -\frac{1}{2} \begin{bmatrix} \mathbf{x}_1 - \boldsymbol{\mu}_1 \\ \mathbf{x}_2 - \boldsymbol{\mu}_2 \end{bmatrix}^T \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{x}_1 - \boldsymbol{\mu}_1 \\ \mathbf{x}_2 - \boldsymbol{\mu}_2 \end{bmatrix} \right\} \\ &= \exp\left\{ -\frac{1}{2} \begin{bmatrix} \mathbf{x}_1 - \boldsymbol{\mu}_1 \\ \mathbf{x}_2 - \boldsymbol{\mu}_2 \end{bmatrix}^T \begin{bmatrix} \mathbf{I} & -\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{\Sigma}_{11}^{-1} & \mathbf{0} \\ \mathbf{0} & (\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{11})^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1} & \mathbf{I} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 - \boldsymbol{\mu}_1 \\ \mathbf{x}_2 - \boldsymbol{\mu}_2 \end{bmatrix} \right\} \\ &= \exp\left\{ -\frac{1}{2} (\mathbf{x}_1 - \boldsymbol{\mu}_1)^T \boldsymbol{\Sigma}_{11}^{-1} (\mathbf{x}_1 - \boldsymbol{\mu}_1) \right\} \\ &\quad \times \exp\left\{ -\frac{1}{2} \Big(\mathbf{x}_2 - \boldsymbol{\mu}_2 - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}(\mathbf{x}_1 - \boldsymbol{\mu}_1) \Big)^T (\boldsymbol{\Sigma}/\boldsymbol{\Sigma}_{11})^{-1} \Big(\mathbf{x}_2 - \boldsymbol{\mu}_2 - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}(\mathbf{x}_1 - \boldsymbol{\mu}_1) \Big) \right\} \end{aligned}$$

where the last step uses the fact that $\mathbf{\Sigma}_{12}^T = \mathbf{\Sigma}_{21}$, so:

$$(\mathbf{x}_1 - \boldsymbol{\mu}_1)^T \mathbf{\Sigma}_{11}^{-1} \mathbf{\Sigma}_{12} = \Big(\mathbf{\Sigma}_{21} \mathbf{\Sigma}_{11}^{-1} (\mathbf{x}_1 - \boldsymbol{\mu}_1)\Big)^T \quad .$$

Then, we use Equation (A.5) to similarly factor the normalizing term:

$$\begin{aligned}
\frac{1}{(2\pi)^{(n_1+n_2)/2} |\mathbf{\Sigma}|^{1/2}} &= \frac{1}{(2\pi)^{(n_1+n_2)/2} \big(|\mathbf{\Sigma}/\mathbf{\Sigma}_{11}||\mathbf{\Sigma}_{11}|\big)^{1/2}} \\
&= \left(\frac{1}{(2\pi)^{n_1/2} |\mathbf{\Sigma}_{11}|^{1/2}}\right)\left(\frac{1}{(2\pi)^{n_2/2} |\mathbf{\Sigma}/\mathbf{\Sigma}_{11}|^{1/2}}\right) \quad .
\end{aligned}$$

And now we can write the desired expressions for the marginal $\mathrm{p}(\mathbf{x}_1)$ and conditional $\mathrm{p}(\mathbf{x}_2 \,|\, \mathbf{x}_1)$ using the first and second factors of the exponential and normalizing terms:

$$\begin{aligned}
\mathrm{p}(\mathbf{x}_1) &= \frac{1}{(2\pi)^{n_1/2} |\mathbf{\Sigma}_{11}|^{1/2}} \exp\left\{ -\frac{1}{2} (\mathbf{x}_1 - \boldsymbol{\mu}_1)^T \mathbf{\Sigma}_{11}^{-1} (\mathbf{x}_1 - \boldsymbol{\mu}_1) \right\} \\
\mathrm{p}(\mathbf{x}_2 \,|\, \mathbf{x}_1) &= \frac{1}{(2\pi)^{n_2/2} |\mathbf{\Sigma}/\mathbf{\Sigma}_{11}|^{1/2}} \\
&\quad \times \exp\left\{ -\frac{1}{2} \Big(\mathbf{x}_2 - \boldsymbol{\mu}_2 - \mathbf{\Sigma}_{21}\mathbf{\Sigma}_{11}^{-1}(\mathbf{x}_1 - \boldsymbol{\mu}_1)\Big)^T (\mathbf{\Sigma}/\mathbf{\Sigma}_{11})^{-1} \right. \\
&\qquad \left. \times \Big(\mathbf{x}_2 - \boldsymbol{\mu}_2 - \mathbf{\Sigma}_{21}\mathbf{\Sigma}_{11}^{-1}(\mathbf{x}_1 - \boldsymbol{\mu}_1)\Big) \right\} \quad .
\end{aligned}$$

Say that $\langle \boldsymbol{\mu}_1^m, \mathbf{\Sigma}_1^m \rangle$ are the parameters of the marginal distribution of $\mathbf{x}_1$, and that $\langle \boldsymbol{\mu}_{2|1}^c, \mathbf{\Sigma}_{2|1}^c \rangle$ are the parameters of the conditional distribution of $\mathbf{x}_2$ given $\mathbf{x}_1$. We can then write expressions for these parameters in terms of the partitioned parts of $\boldsymbol{\mu}$ and $\mathbf{\Sigma}$ as:

Marginal:

$$\boldsymbol{\mu}_1^m = \boldsymbol{\mu}_1 \qquad \text{(A.7a)}$$

$$\mathbf{\Sigma}_1^m = \mathbf{\Sigma}_{11} \qquad \text{(A.7b)}$$

and

Conditional:

$$\boldsymbol{\mu}_{2|1}^c = \boldsymbol{\mu}_2 + \mathbf{\Sigma}_{21}\mathbf{\Sigma}_{11}^{-1}(\mathbf{x}_1 - \boldsymbol{\mu}_1) \qquad \text{(A.8a)}$$

$$\mathbf{\Sigma}_{2|1}^c = \mathbf{\Sigma}_{22} - \mathbf{\Sigma}_{21}\mathbf{\Sigma}_{11}^{-1}\mathbf{\Sigma}_{12} \qquad \text{(A.8b)}$$

A.3.3 Sources and Related Work

My primary source for multivariate Gaussian distributions is Chapter 13 of Michael Jordan's *An Introduction to Probabilistic Graphical Models* [18]. Jordan derives block diagonalization results for a partitioned matrix, and marginalization and conditioning results for a partitioned multivariate Gaussian, but he chooses to factor $p(\mathbf{x}_1, \mathbf{x}_2)$ into $p(\mathbf{x}_1 \mid \mathbf{x}_2)\, p(\mathbf{x}_2)$. The alternative factorization $p(\mathbf{x}_2 \mid \mathbf{x}_1)\, p(\mathbf{x}_1)$ is equally valid, and is more intuitive and useful for the PGG framework. Thus the results in this section are rederived from Jordan's results in order to obtain this alternative factorization.

A.4 Estimating The Minimum Volume Bounding Box

In Chapter 2, we show that a geometric "summarization function" is necessary in order to extend context-free grammars with geometric information. We will use the bounding box as a summarization function in this work for a variety of reasons that are discussed in subsequent chapters; a primary reason is that it allows a consistent interface and representation for all primitive and composite parts.

It is a nontrivial task to determine the optimal minimum volume bounding box of a set of 3D boxes; however, approximate optimality is sufficient for our purposes.

A.4.1 An Approximation Algorithm

This is an algorithm which returns an approximation to the minimum volume bounding box enclosing a set of 3D boxes:

1. Let $\mathbf{H}$ be the convex hull of the union of the vertices of the input boxes $B = \mathbf{b}_1 \ldots \mathbf{b}_n$.
2. Let $\mathbf{M}$ be the moment tensor and $\mathbf{p}$ be the center of mass of $\mathbf{H}$.
3. Let $\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3]$ be the eigenvectors of the moment tensor matrix $\mathbf{M}$.
4. Reorder the columns of $\mathbf{A}$ to produce a right-hand coordinate frame $\mathbf{R} = [\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3]$; i.e., one in which $\mathbf{r}_1 \times \mathbf{r}_2 = \mathbf{r}_3$.
5. Let $\mathbf{d}$ be the extent of the vertices of B in the frame $\mathbf{R}$.
6. Let $\mathbf{b} = \langle \mathbf{d}, \mathbf{p}, \mathbf{R} \rangle$ be the estimated minimum volume bounding box of B.

It remains to be proved why this algorithm works well, but in practice it seems to offer reasonable performance nonetheless.

A.4.2 Sources and Implementation

In order to find the convex hull of a set of points, we use John Lloyd's 2004 Java implementation [21] of Barber, Dobkin, and Huhdanpaa's Quickhull algorithm for convex hulls [2]. We then use Brian Mirtich's algorithm and C++ implementation for finding the moment tensor of the convex hull [24].

Appendix B

Experiment Models

```
scene(s)[0]#.
0.5  scene(s)[0] :- chair(c)[0]%.
0.0833  scene(s)[1] :- stool(o)[0]%.
0.1667  scene(s)[2] :- bench(b)[0]%.
0.0833  scene(s)[3] :- table(t)[0]%.
0.0833  scene(s)[4] :- coffee_table(f)[0]%.
0.0833  scene(s)[5] :- lamp(p)[0]%.

chair(c)[1]#.
1  chair(c)[0] :- chair_top(ct)[0]%, chair_base(cb)[1]%.

chair_top(ct)[2]#.
0.5  chair_top(ct)[0] :- chair_back(ck)[0]%, chair_seat(cs)[1]%.
0.5  chair_top(ct)[1] :- chair_back(ck)[0]%, chair_seat(cs)[1]%,
                         chair_arm(ca1)[2]%, chair_arm(ca2)[3]%.

chair_base(cb)[3]#.
0.3333  chair_base(cb)[0] :- chair_leg(cl1)[0]%, chair_leg(cl2)[1]%,
                             chair_leg(cl3)[2]%, chair_leg(cl4)[3]%.
0.3333  chair_base(cb)[1] :- chair_axle(cx)[0]%, chair_wheel_leg(cw1)[1]%,
                             chair_wheel_leg(cw2)[2]%, chair_wheel_leg(cw3)[3]%.
0.3333  chair_base(cb)[2] :- chair_axle(cx)[0]%, chair_wheel_leg(cw1)[1]%,
                             chair_wheel_leg(cw2)[2]%, chair_wheel_leg(cw3)[3]%,
                             chair_wheel_leg(cw4)[4]%, chair_wheel_leg(cw5)[5]%.

stool(o)[4]#.
1  stool(o)[0] :- chair_seat(cs)[0]%, stool_base(ob)[1]%.

stool_base(ob)[5]#.
1  stool_base(ob)[0] :- chair_leg(cl1)[0]%, chair_leg(cl2)[1]%,
                        chair_leg(cl3)[2]%, chair_leg(cl4)[3]%.
```

Figure B-1: The PGG used in the experiments (continued in next figure). The learned expansion probabilities are shown. The presence of a learned root geometric model is denoted with a #, and the presence of a learned part geometric model with a %.

```
chair_back(ck)[6]#.
chair_seat(cs)[7]#.
chair_arm(ca)[8]#.
chair_leg(cl)[9]#.
chair_axle(cx)[10]#.
chair_wheel_leg(cw)[11]#.

bench(b)[12]#.
1  bench(b)[0] :- bench_top(bt)[0]%, bench_base(bb)[1]%.

bench_top(bt)[13]#.
0.5  bench_top(bt)[0] :- bench_back(bk)[0]%, bench_seat(bs)[1]%.
0.5  bench_top(bt)[1] :- bench_back(bk)[0]%, bench_seat(bs)[1]%,
                         bench_arm(ba1)[2]%, bench_arm(ba2)[3]%.

bench_base(bb)[14]#.
1  bench_base(bb)[0] :- bench_leg(bl1)[0]%, bench_leg(bl2)[1]%,
                        bench_leg(bl3)[2]%, bench_leg(bl4)[3]%.

bench_back(bk)[15]#.
bench_seat(bs)[16]#.
bench_arm(ba)[17]#.
bench_leg(bl)[18]#.

table(t)[19]#.
1  table(t)[0] :- table_top(tt)[0]%, table_base(tb)[1]%.

table_base(tb)[20]#.
1  table_base(tb)[0] :- table_leg(tl1)[0]%, table_leg(tl2)[1]%,
                        table_leg(tl3)[2]%, table_leg(tl4)[3]%.

table_top(tt)[21]#.
table_leg(tl)[22]#.

coffee_table(f)[23]#.
1  coffee_table(f)[0] :- coffee_table_top(ft)[0]%, coffee_table_base(fb)[1]%.

coffee_table_base(fb)[24]#.
1  coffee_table_base(fb)[0] :- coffee_table_leg(fl1)[0]%, coffee_table_leg(fl2)[1]%,
                               coffee_table_leg(fl3)[2]%, coffee_table_leg(fl4)[3]%.

coffee_table_top(ft)[25]#.
coffee_table_leg(fl)[26]#.

lamp(p)[27]#.
1  lamp(p)[0] :- lamp_post(pp)[0]%, lamp_shade(ps)[1]%, lamp_base(pb)[2]%.

lamp_post(pp)[28]#.
lamp_shade(ps)[29]#.
lamp_base(pb)[30]#.

Clutter penalty: 15.0
```

Figure B-2: The PGG used in the experiments (continued).

```
0.0833   chair(V0)# :- scene(V1), chair(V2), chair_top(V3), chair_back(v0),
                       chair_seat(v1), chair_base(V4), chair_leg(v2), chair_leg(v3),
                       chair_leg(v4), chair_leg(v5).
0.0833   chair(V0)# :- scene(V1), chair(V2), chair_top(V3), chair_back(v0),
                       chair_seat(v1), chair_arm(v2), chair_arm(v3), chair_base(V4),
                       chair_leg(v4), chair_leg(v5), chair_leg(v6), chair_leg(v7).
0.0833   chair(V0)# :- scene(V1), chair(V2), chair_top(V3), chair_back(v0),
                       chair_seat(v1), chair_base(V4), chair_axle(v2),
                       chair_wheel_leg(v3), chair_wheel_leg(v4), chair_wheel_leg(v5).
0.0833   chair(V0)# :- scene(V1), chair(V2), chair_top(V3), chair_back(v0),
                       chair_seat(v1), chair_arm(v2), chair_arm(v3), chair_base(V4),
                       chair_axle(v4), chair_wheel_leg(v5), chair_wheel_leg(v6),
                       chair_wheel_leg(v7).
0.0833   chair(V0)# :- scene(V1), chair(V2), chair_top(V3), chair_back(v0),
                       chair_seat(v1), chair_base(V4), chair_axle(v2),
                       chair_wheel_leg(v3), chair_wheel_leg(v4), chair_wheel_leg(v5),
                       chair_wheel_leg(v6), chair_wheel_leg(v7).
0.0833   chair(V0)# :- scene(V1), chair(V2), chair_top(V3), chair_back(v0),
                       chair_seat(v1), chair_arm(v2), chair_arm(v3), chair_base(V4),
                       chair_axle(v4), chair_wheel_leg(v5), chair_wheel_leg(v6),
                       chair_wheel_leg(v7), chair_wheel_leg(v8), chair_wheel_leg(v9).
0.0833   stool(V0)# :- scene(V1), stool(V2), chair_seat(v0), stool_base(V3),
                       chair_leg(v1), chair_leg(v2), chair_leg(v3), chair_leg(v4).
0.0833   bench(V0)# :- scene(V1), bench(V2), bench_top(V3), bench_back(v0),
                       bench_seat(v1), bench_base(V4), bench_leg(v2), bench_leg(v3),
                       bench_leg(v4), bench_leg(v5).
0.0833   bench(V0)# :- scene(V1), bench(V2), bench_top(V3), bench_back(v0),
                       bench_seat(v1), bench_arm(v2), bench_arm(v3), bench_base(V4),
                       bench_leg(v4), bench_leg(v5), bench_leg(v6), bench_leg(v7).
0.0833   table(V0)# :- scene(V1), table(V2), table_top(v0), table_base(V3),
                       table_leg(v1), table_leg(v2), table_leg(v3), table_leg(v4).
0.0833   coffee_table(V0)# :- scene(V1), coffee_table(V2), coffee_table_top(v0),
                              coffee_table_base(V3), coffee_table_leg(v1),
                              coffee_table_leg(v2), coffee_table_leg(v3),
                              coffee_table_leg(v4).
0.0833   lamp(V0)# :- scene(V1), lamp(V2), lamp_post(v0), lamp_shade(v1),
                      lamp_base(v2).
```

Figure B-3: The fully connected models used in the experiments. The learned prior structural probabilities over models are shown. The presence of a learned geometric model is denoted with a #.

```
0.0833   chair(V0)
+--+ scene(V1)#
   +--+ chair(V2)%
      +--+ chair_top(V3)%
      |  +--+ chair_back(v0)%
      |  +--+ chair_seat(v1)%
      +--+ chair_base(V4)%
         +--+ chair_leg(v2)%
         +--+ chair_leg(v3)%
         +--+ chair_leg(v4)%
         +--+ chair_leg(v5)%

0.0833   chair(V0)
+--+ scene(V1)#
   +--+ chair(V2)%
      +--+ chair_top(V3)%
      |  +--+ chair_back(v0)%
      |  +--+ chair_seat(v1)%
      |  +--+ chair_arm(v2)%
      |  +--+ chair_arm(v3)%
      +--+ chair_base(V4)%
         +--+ chair_leg(v4)%
         +--+ chair_leg(v5)%
         +--+ chair_leg(v6)%
         +--+ chair_leg(v7)%

0.0833   chair(V0)
+--+ scene(V1)#
   +--+ chair(V2)%
      +--+ chair_top(V3)%
      |  +--+ chair_back(v0)%
      |  +--+ chair_seat(v1)%
      +--+ chair_base(V4)%
         +--+ chair_axle(v2)%
         +--+ chair_wheel_leg(v3)%
         +--+ chair_wheel_leg(v4)%
         +--+ chair_wheel_leg(v5)%

0.0833   chair(V0)
+--+ scene(V1)#
   +--+ chair(V2)%
      +--+ chair_top(V3)%
      |  +--+ chair_back(v0)%
      |  +--+ chair_seat(v1)%
      |  +--+ chair_arm(v2)%
      |  +--+ chair_arm(v3)%
      +--+ chair_base(V4)%
         +--+ chair_axle(v4)%
         +--+ chair_wheel_leg(v5)%
         +--+ chair_wheel_leg(v6)%
         +--+ chair_wheel_leg(v7)%

0.0833   chair(V0)
+--+ scene(V1)#
   +--+ chair(V2)%
      +--+ chair_top(V3)%
      |  +--+ chair_back(v0)%
      |  +--+ chair_seat(v1)%
      +--+ chair_base(V4)%
         +--+ chair_axle(v2)%
         +--+ chair_wheel_leg(v3)%
         +--+ chair_wheel_leg(v4)%
         +--+ chair_wheel_leg(v5)%
         +--+ chair_wheel_leg(v6)%
         +--+ chair_wheel_leg(v7)%

0.0833   chair(V0)
+--+ scene(V1)#
   +--+ chair(V2)%
      +--+ chair_top(V3)%
      |  +--+ chair_back(v0)%
      |  +--+ chair_seat(v1)%
      |  +--+ chair_arm(v2)%
      |  +--+ chair_arm(v3)%
      +--+ chair_base(V4)%
         +--+ chair_axle(v4)%
         +--+ chair_wheel_leg(v5)%
         +--+ chair_wheel_leg(v6)%
         +--+ chair_wheel_leg(v7)%
         +--+ chair_wheel_leg(v8)%
         +--+ chair_wheel_leg(v9)%
```

Figure B-4: The Bayes net models used in the experiments (continued in next figure). The learned prior structural probabilities over models are shown. The presence of a learned root geometric model is denoted with a #, and the presence of a learned part geometric model with a %.

```
0.0833   stool(V0)
+--+ scene(V1)#
   +--+ stool(V2)%
      +--+ chair_seat(v0)%
      +--+ stool_base(V3)%
         +--+ chair_leg(v1)%
         +--+ chair_leg(v2)%
         +--+ chair_leg(v3)%
         +--+ chair_leg(v4)%

0.0833   bench(V0)
+--+ scene(V1)#
   +--+ bench(V2)%
      +--+ bench_top(V3)%
      |  +--+ bench_back(v0)%
      |  +--+ bench_seat(v1)%
      +--+ bench_base(V4)%
         +--+ bench_leg(v2)%
         +--+ bench_leg(v3)%
         +--+ bench_leg(v4)%
         +--+ bench_leg(v5)%

0.0833   bench(V0)
+--+ scene(V1)#
   +--+ bench(V2)%
      +--+ bench_top(V3)%
      |  +--+ bench_back(v0)%
      |  +--+ bench_seat(v1)%
      |  +--+ bench_arm(v2)%
      |  +--+ bench_arm(v3)%
      +--+ bench_base(V4)%
         +--+ bench_leg(v4)%
         +--+ bench_leg(v5)%
         +--+ bench_leg(v6)%
         +--+ bench_leg(v7)%
```

```
0.0833   table(V0)
+--+ scene(V1)#
   +--+ table(V2)%
      +--+ table_top(v0)%
      +--+ table_base(V3)%
         +--+ table_leg(v1)%
         +--+ table_leg(v2)%
         +--+ table_leg(v3)%
         +--+ table_leg(v4)%

0.0833   coffee_table(V0)
+--+ scene(V1)#
   +--+ coffee_table(V2)%
      +--+ coffee_table_top(v0)%
      +--+ coffee_table_base(V3)%
         +--+ coffee_table_leg(v1)%
         +--+ coffee_table_leg(v2)%
         +--+ coffee_table_leg(v3)%
         +--+ coffee_table_leg(v4)%

0.0833   lamp(V0)
+--+ scene(V1)#
   +--+ lamp(V2)%
      +--+ lamp_post(v0)%
      +--+ lamp_shade(v1)%
      +--+ lamp_base(v2)%
```

Figure B-5: The Bayes net models used in the experiments (continued).

Bibliography

[1] James Allen. *Natural Language Understanding*. The Benjamin/Cummings Publishing Company, Inc., 1995.

[2] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The Quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4), December 1996.

[3] Jeffrey S. Beis and David G. Lowe. Learning indexing functions for 3-D model-based object recognition. In *Proc. AAAI Fall Symposium: Machine Learning in Computer Vision*, pages 275–280, 1993.

[4] Irving Biederman and Moshe Bar. Differing views on views: Response to Hayward and Tarr (2000). *Vision Res.*, 40(28):3901–3905, 2000.

[5] Tom Binford. Visual perception by computer. In *Proc. IEEE Conf. on Systems and Control*, 1971.

[6] V. Blanz, B. Scholkopf, H. Bulthoff, C. Burges, V. Vapnik, and T. Vetter. Comparison of view-based object recognition algorithms using realistic 3D models. In *Proc. Artifical Neural Networks, ICANN*, pages 251–256, 1996.

[7] Andrew Burbanks. Quaternions in C++. November 1, 1996.

[8] Dave Coverly. *Speed Bump Cartoons*. 2003. http://www.speedbump.com/.

[9] David Crandall, Pedro Felzenszwalb, and Daniel Huttenlocher. Spatial priors for part-based recognition using statistical models. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2005.

[10] David Eberly. Quaternion algebra and calculus. September 27, 2002.

[11] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2003.

[12] P. Thomas Fletcher, Sarang Joshi, Conglin LU, and Stephen Pizer. Gaussian distributions on Lie groups and their application to statistical shape analysis. In *Proc. Information Processing in Medical Imaging*, pages 450–462, 2003.

[13] David Forsyth and John Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, 2001.

[14] Berthold K. P. Horn. *Robot Vision*. The MIT Press, 1986.

[15] John E. Hummel. Where view-based theories break down: The role of structure in shape perception and object recognition. In E. Dietrich and A. Markman, editors, *Cognitive Dynamics: Conceptual Change in Humans and Machines*, pages 157–185. Hillsdale, NJ: Erlbaum, 2000.

[16] Daniel Huttenlocher and Shimon Ullman. Object recognition using alignment. In *Proc. Int'l Conf. on Computer Vision*, pages 102–111, 1986.

[17] Michael Patrick Johnson. *Exploiting Quaternions to Support Expressive Interactive Character Motion*. PhD thesis, Massachusetts Institute of Technology, 2003.

[18] M. I. Jordan. *An Introduction to Probabilistic Graphical Models*. 2006.

[19] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Prentice Hall, 2000.

[20] J. Koenderink and A. van Doorn. The internal representation of solid shape with respect to vision. *Biological Cybernetics*, 32:211–216, 1979.

[21] John E. Lloyd. Quickhull3d, 2004. A three dimensional implementation of Barber, Dobkin, and Huhdanpaa's Quickhull, in Java.

[22] Tomás Lozano-Pérez and Leslie Pack Kaelbling. Learning to recognize objects, 2004.

[23] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. The MIT Press, 2002.

[24] Brian Mirtich. Fast and accurate computation of polyhedral mass properties. *Journal of Graphics Tools*, 1(2), 1996.

[25] Darnell Moore and Irfan Essa. Recognizing multitasked activities using stochastic context-free grammar. In *Proc. Workshop on Models versus Exemplars in Computer Vision, held in Conjunction with IEEE CVPR*, 2001.

[26] C. Papageorgiou and T. Poggio. A trainable system for object detection. *International Journal of Computer Vision*, 38(1):15–33, 2000.

[27] I. Pollak, J. M. Siskind, M. P. Harper, and C. A. Bouman. Parameter estimation for spatial random trees using the EM algorithm. In *Proc. IEEE Int'l Conf. on Image Processing*, 2003.

[28] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, second edition, 1992.

[29] Azriel Rosenfeld. Progress in picture processing: 1969-71. *ACM Computing Surveys*, 5(2), June 1973.

[30] Fredrick Rothganger, Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. 3D object modeling and recognition using affine-invariant patches and multi-view spatial constraints. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2003.

[31] C.A. Rothwell, A. Zisserman, J.L. Mundy, and D.A. Forsyth. Efficient model library access by projectively invariant indexing functions. In *Proc. IEEE Computer Vision and Pattern Recognition*, 1992.

[32] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, second edition, 2003.

[33] Ken Shoemake. Animating rotation with quaternion calculus. In *ACM SIGGRAPH, Course Notes 10, Computer Animation: 3-D Motion, Specification, and Control*, 1987.

[34] Michael J. Tarr. Visual object recognition: Can a single mechanism suffice? In M.A. Peterson and G. Rhodes, editors, *Perception of Faces, Objects, and Scenes: Analytic and Holistic Processes.* Oxford, UK: Oxford University Press, 2001.

[35] Zhuowen Tu, Xiangrong Chen, Alan L. Yuille, and Song-Chun Zhu. Image parsing: Unifying segmentation, detection, and recognition. In *Proc. IEEE Int'l Conf. on Computer Vision*, 2003.

[36] M. Turk and A. Pentland. Face recognition using eigenfaces. In *Proc. IEEE Computer Vision and Pattern Recognition*, 1991.

[37] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proc. IEEE Computer Vision and Pattern Recognition*, 2001.

[38] Eric W. Weisstein. Stirling number of the second kind. From Mathworld – A Wolfram Web Resource. http://mathworld.wolfram.com/.