Testing Inference Programs for Generative Scene Graphs

by

Austin J. Garrett

Department of Electrical Engineering and Computer Science

Proposal for Thesis Research in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

December 2019

Author .....
Department of Electrical Engineering and Computer Science
December 11, 2019

Certified by .....
Vikash Mansinghka
Research Scientist
Thesis Supervisor

Accepted by .....
Placeholder
Chairman, Department Committee on Graduate Theses

Testing Inference Programs for Generative Scene Graphs

by

Austin J. Garrett

Submitted to the Department of Electrical Engineering and Computer Science
on December 11, 2019, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, I designed and implemented a compiler which performs optimizations that reduce the number of low-level floating point operations necessary for a specific task; this involves the optimization of chains of floating point operations as well as the implementation of a "fixed" point data type that allows some floating point operations to simulated with integer arithmetic. The source language of the compiler is a subset of C, and the destination language is assembly language for a micro-floating point CPU. An instruction-level simulator of the CPU was written to allow testing of the code. A series of test pieces of codes was compiled, both with and without optimization, to determine how effective these optimizations were.

Thesis Supervisor: Vikash Mansinghka
Title: Research Scientist

Acknowledgments

This is the acknowledgements section. You should replace this with your own acknowledgements.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Testing scene graph models

1.1.1 History of Inverse Graphics

1.1.2 Scene Graphs as Compact Scene Representations

1.2 Summary of Thesis

TODO: replace me

Chapter 2

Scene Graphs

2.0.1 Mathematical Description

We model the geometric state of a scene at a single time point as a *scene graph*, which is a tuple (G, Θ, Z), where $G = (V, E)$ is a directed tree that encodes the scene graph *structure* and Θ encodes the scene graph *continuous parameters* and Z encodes the scene graph *discrete parameters*. Although the framework we present can represent articulated objects, in this paper we only consider scenes involving a set of rigid objects O. The scene graph structure includes a vertex $v_o \in V$ for each object $o \in O$ that represents the 6DoF pose of object o, as well as a vertex $r \in V$ that represents the the coordinate frame of the observer. A directed edge $e = (v_i, v_j) \in E$ between objects $i, j \in O$ encodes that the pose of object j is parametrized *relative to* the pose of object i, by parameters θ_e and z_e. The pose of object j relative to object i is denoted $\Delta x(z_e, \theta_e) \in SE(3)$. A directed edge $e = (r, v_j) \in E$ from the root to an object $j \in O$ encodes that the pose of object j is not parametrized relative to that of any object, but is instead a full 6DoF pose $\theta_e \in SE(3)$ where $SE(3)$ is the special Euclidean group consisting of all 6DoF poses. The continuous and discrete parameters of the scene graph consist of the parameters for each edge in the structure ($\Theta := \{\theta_e\}_{e \in E}$ and $Z := \{z_e\}_{e \in E}$). The set of edges E must *span* the set of vertices $V := \{r\} \cup \{v_o\}_{o \in O}$; that is the scene graph structure G is a *directed spanning tree*.

2.0.2 Computing the 6DoF poses of all objects in the scene

Given a scene graph (G, Θ, Z) the pose $x_i \in SE(3)$ of an object i relative to the observer coordinate frame can be computed by walking path in the tree G from the root vertex $r \in V$ to the vertex $v_i \in V$, and successively computing the pose of each object along the path from the pose of its parent. That is, given pose $x_u \in SE(3)$ and edge $(u, v) \in E$, the pose $x_v \in SE(3)$ is computed as $x_v := x_v \cdot \Delta x(z_e, \theta_e)$ where $\Delta x(z_e, \theta_e) \in SE(3)$ is the relative pose between vertex u and vertex v. For an edge from the root vertex to an object vertex ($e = (r, v_i)$), $x_u := \mathbf{1}$ (the identity element of $SE(3)$) and $\Delta x(z_e, \theta_e) = \theta_e \in SE(3)$ (note that there are no discrete parameters in this case, so $z_e := ()$). The functional form of $\Delta x(z_e, \theta_e)$ for an edge e between two objects is discussed below. One requirement is that $\Delta x(z_e, \theta_e)$ is a differentiable function of θ_e; this enables inference algorithm that exploit gradient information.

2.0.3 Modeling face-to-face contact between two objects

An edge $(v_i, v_j) \in E$ from object i to object j indicates that the pose of object j is represented relative to the pose of object i. Various types of relative pose parametrizations for two objects are possible; for simplicity we model objects as polyhedra, and only model face-to-face contact between objects. That is, an edge $e = (v_i, v_j)$ indicates that a face of object i is in flush contact with a face of object j. Since each object has multiple faces, the choice of which pair of faces is in contact is encoded in the discrete parameters z_e for the edge. Concretely, let F_i and F_j denote the faces of object i and object j respectively Then, $z_e \in F_i \times F_j$. The continuous parameters θ_e for an edge e between two objects is an element of $\mathbb{R}^2 \times [0, 2\pi)$ that contains two translational degrees of freedom ($s, t \in \mathbb{R}$, for the relative offset of the two faces) and one rotational degree of freedom ($\phi \in [0, 2\pi)$). For example, a cuboid object $i \in O$, the set of faces is $F_i = \{\text{Top}, \text{Bottom}, \text{Left}, \text{Right}, \text{Front}, \text{Back}\}$. For an edge $e = (i, j)$ where objects $i, j \in O$ where both objects i and j are cuboids, there are 36 possible values for z_e.

Slack variables for face-to-face contact We extend the parametrization of face-to-face contact between objects with three additional degrees of freedom of *slack variables*: (i) one degree of freedom that encodes the perpendicular distance ($d \in [0, \infty)$) between the two contact faces, and (ii) two degrees of freedom for relative orientation of the two faces, encoded the surface normal unit vector $\mathbf{n}$ of the child object's face, relative to the parent object's face, which takes values on the sphere S^2. Therefore, $\theta_e \in \mathbb{R}^2 \times [0, 2\pi) \times [0, \infty) \times S^2$ for an edge $e = (v_i, v_j)$ between two objects i and j. Note that although this edge parametrization uses six degrees of freedom for object-to-object edges like the edge parametrization for edges from the root ($e = (r, v_j)$), the prior distribution on these parameters (described below) encourages object j to be *almost* in face-to-face contact with object i; whereas the prior distribution on the pose of object j for an edge of the form (r, v_j) is very different, and is typically a uniform distribution over positions within the scene bounding volume, and a uniform distribution on orientations.

2.0.4 A Prior Distribution on Scene Graphs

Various prior distributions on scene graphs are possible. In our experiments, we use a generic prior distribution on scene graphs over a collection of objects O that factors into two components: (i) a prior distribution on scene graph structures G, denoted $p(G)$, and (ii) a prior distribution on scene graph parameters (Z, Θ) given structure, denoted $p(Z, \Theta|G)$. While uncertainty about the number of objects is possible to represent in our framework and implementation, for simplicity assume that the set of objects is known a-priori, and that there is one vertex for each object, and one vertex representing the observer coordinate frame, so V is fixed a-priori to $\{r\} \cup \{v_o\}_{o \in O}$. Therefore, $p(G)$ reduces to a prior distribution on edges in the scene graph. For the prior distribution on structure, we use a uniform distribution on directed trees that are rooted at vertex r and span all $|O| + 1$ vertices in the graph. This set of directed trees is isomorphic to the set of undirected spanning trees over $|O| + 1$ vertices. Therefore, the prior probability of a graph $G = (V, E)$ is obtained by using Cayley's formula to count the number of undirected spanning trees on $|O| + 1$

vertices:

$$p(G) := p_{\text{unif}(|O|)}(G) := \begin{cases} (|O|+1)^{1-|O|} & \text{if } G \text{ is a directed spanning tree over vertices } V \text{ rooted at } r \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

The prior distribution on scene graph parameters factors over the edges in the scene graph:

$$p(Z, \Theta|G) := \prod_{e \in E} p_e(z_e, \theta_e) \tag{2.2}$$

where $p_e(z_e, \theta_e)$ is a probability distribution on z_e and θ_e that depends on the two vertices in the edge $e = (u, v)$. For edges $e = (r, v_j)$ where j is an object, $z_e = ()$ and $p_e(z_e, \theta_e)$ is the uniform distribution on elements of $B \times SO(3)$ where B is the bounding volume of the scene and $SO(3)$ is 3D rotation group (the uniform distribution on $SO(3)$ is given by the Haar measure). For edges $e = (v_i, v_j)$ where i and j are objects, recall that $z_e \in F_i \times F_j$ and (using the edge parametrization including slack variables) $\theta_e = (s, t, \phi, d, \alpha_1, \alpha_2) \in \mathbb{R}^2 \times [0, 2\pi) \times [0, \infty) \times [0, 2\pi)^2$. The prior distribution on edge parameters is:

$$p_e(z_e, \theta_e) := \frac{1}{|F_i||F_j|} \cdot p_{\text{norm}(0,\sigma)}(s) \cdot p_{\text{norm}(0,\sigma)}(t) \cdot \frac{1}{2\pi} \cdot p_{\text{exp}(\beta)}(d) \cdot \frac{1}{2\pi} \cdot \frac{1}{2\pi} \tag{2.3}$$

(where p_{norm} and p_{exp} are the normal and exponential distribution density functions, respectively).

2.0.5 Probabilistic Dynamics on Scene Graphs

We build temporal models of scene geometry based on Markov chains of scene graphs $\{(G_t, \Theta_t, Z_t)\}_{t=1}^T$, where t indexes time, with transitions $p(G_t, \Theta_t, Z_t|G_{t-1}, \Theta_{t-1}, Z_{t-1})$. We factor the dynamics model on scene graphs decomposes into a dynamics model on scene graph structure, and a dynamics on parameters:

$$p(G_t, \Theta_t, Z_t|G_{t-1}, \Theta_{t-1}, Z_{t-1}) := p(G_t|G_{t-1}) \cdot p(Z_t, \Theta_t, |Z_{t-1}, \Theta_{t-1}, G_{t-1}, G_t) \tag{2.4}$$

The dynamics on graph structure is based on a mixture of a random walk on structures (to capture incremental changes to the structure of a scene that often occur) and a uniform distribution on structures (to model sudden unexpected changes in structure):

$$p(G_t|G_{t-1}) := 0.9 \cdot p_{\text{walk}}(G_t|G_{t-1}) + 0.1 \cdot p_{\text{unif}(|O|)}(G_t) \tag{2.5}$$

where $p_{\text{walk}}(G'|G)$ is the distribution on graph structures induced by a sampling process in which one vertex in the undirected spanning tree is selected at random, severed from the tree, and grafted back onto the graph at a uniformly chosen vertex, subject to the condition that the resulting graph is a spanning tree. Recall that since the tree contains the root node r, this process can change the edge type (u, v_j) for an object i from (r, v_j) (representing independent 6DoF pose) to (v_i, v_j) (representing flush contact pose with another object v_i. The dynamics on parameters factor according to edges in the graph:

$$p(Z_t, \Theta_t|Z_{t-1}, \Theta_{t-1}, G_{t-1}, G_t) := \prod_{e \in E} p_e(z_e, \theta_e|G_t, G_{t-1}, z_{t-1_e}, \theta_{t-1_e}) \tag{2.6}$$

If an edge e is present in both G_t and G_{t-1}, then this distribution is a mixture of a random walk and a uniform distribution. If an edge e in G_t was not present in G_{t-1}, then this distribution is a uniform distribution.

2.1 Example Applications of Scene Graph Models

TODO: replace me

2.1.1 YCB Objects on a synthetic tabletop

TODO: replace me

2.1.2 Real YCB objects on a physical tabletop

TODO: replace me

```
@gen function model(T::Int)
    latent_gs = []
    for t = 1:T
        # structure and parameter dynamics
        if t == 1
            structure ~ UniformDiForest(N)
            params ~ params_init(structure)
        else
            (prev_structure, prev_params) = decompose(gs[t-1])
            structure ~ StructureTransition(prev_structure)
            params ~ params_dynamics(structure, prev_params)
        end

        # observation
        latent_g = SceneGraph(structure, params)
        obs ~ noise_model(latent_g)
        push!(latent_gs, latent_g)
    end
    return latent_gs
end
```

(a) Top-level scene graph model

```
@gen function init_params(structure::SimpleDiGraph)
    params = []
    for i in vertices(structure)
        if isFloating(structure, i)
            xs = {i} ~ init_floating_pose()
        else
            xs = {i} ~ init_sliding_pose(parent(structure, i))
        end
        push!(params, xs)
    end
    return params
end
```

(b) Parameter initialization

```
@gen function params_dynamics(structure, prev_params, hypers)
    new_params = []
    for i in vertices(structure)
        if isFloating(structure, i)
            new_xs = {i} ~ floating_pose_dynamics(prev_params[i])
        else
            new_xs = {i} ~ sliding_pose_dynamics(prev_params[i], parent(structure, i))
        end
        push!(new_params, new_xs)
    end
    return new_params
end
```

(c) Parameter dynamics

```
RobustNoisyPoseLikelihood = Mixture([GaussianVMF, UniformPose])

@gen function noise_model(g)
    # use Mixture to construct a mixture of uniform and gaussian vmf with prob_outlier of being uniform
    observed_poses = []
    for i in vertices(g)
        latent_pose = get_floating_pose(g, i)  # get 6DoF pose for object i
        observed_pose = {i} ~ RobustNoisyPoseLikelihood([1 - p_outlier, p_outlier],
                                                        [(latent_pose, inlier_pos_stdev, inlier_rot_conc),
                                                        (outlier_bounds,)])
        push!(observed_poses, observed_pose)
    end
    return observed_poses
end
```

(d) Noisy observational model

Figure 2-1: Probabilistic pseudocode for an example dynamic scene graph model

2.1.3 Simulated objects in AI2Thor

TODO: replace me

Chapter 3

Visualizing Scene Graphs

3.1 Desiderata

Before we're able to implement our possible scene graph models, Visualizing the richly structured geometric information contained in a generative scene graph model is vital for proper debugging and analysis of modeling and inference programs. The most basic requirement is visualizing the components of our scene graph representation (G, Θ, Z). It is important to be able to clearly distinguish each part of our representation; as we see later, incorrect behavior in any part of our model or inference can introduce huge variation in the posterior and inferred approximation. Observed neural detections can be visualized as a special-case scene graph with no edges. As such, this covers a point-estimate view of the majority of addresses in the scene graph models we explored.

On top of this, we'd also like to capture some information about distributions over scene graphs. Providing an clearly interpretable view of all possible distributions $p(G, \Theta, Z)$ is a huge task, so we restrict ourselves to sample-based approximations of unimodal distributions over continuous parameters, and relatively small numbers of structures. Within this class of scenes, common characterizing features are the mean and uncertainty, so our visualizations should provide a clear view of one or more of these features.

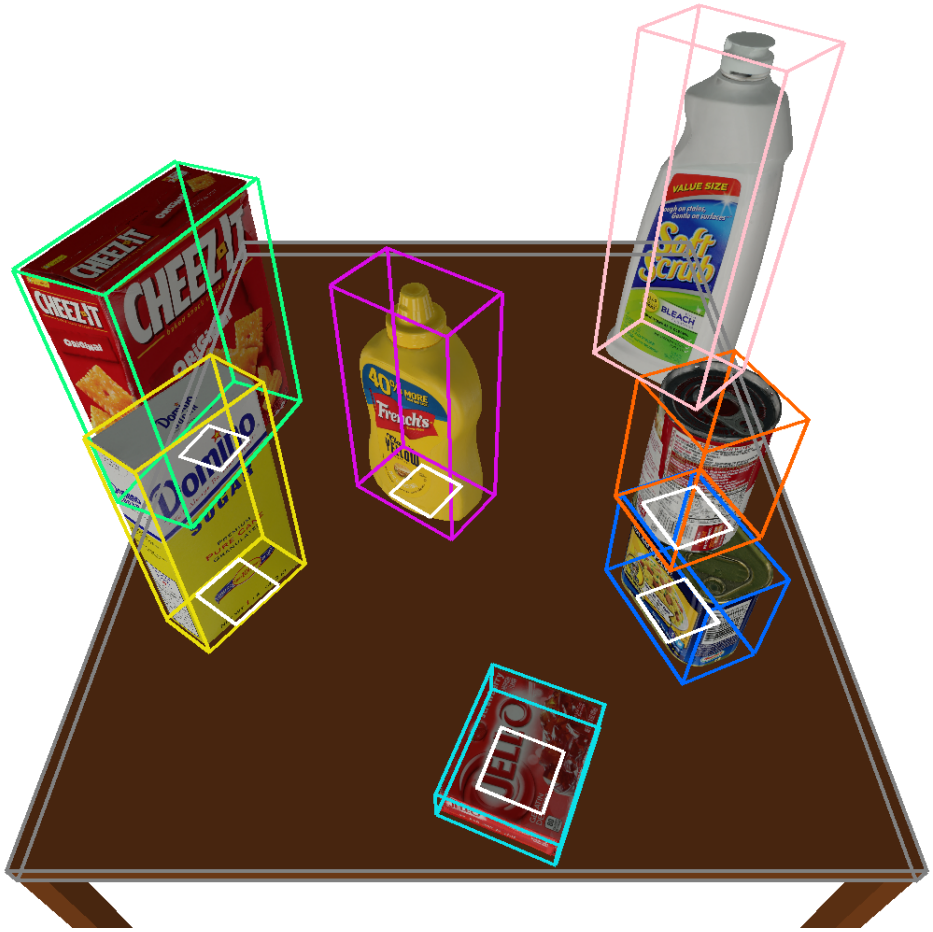


Figure 3-1: Visualization of a synthetic scene with its abstract scene graph overlaid.

3.2 Examples

3.2.1 Visualizing a Single Scene Graph

Our main utility is a function that accepts an image, a scene graph (G, Θ, Z) where $G = (V, E)$, and a camera configuration, and renders that scene graph as a wireframe representation overlaid on top of the image, as viewed from the provided camera specification. Figure **??** demonstrates using this function to see a synthetic rendered scene's underlying scene graph. For each object $o \in O$ in the scene graph, the function renders a colored wireframe bounding box with the dimensions of o, and 6DoF pose given by x_o. Importantly, this is the *absolute* pose of the object, including any slack in relative contacts. Thus, the continuous parameters $\Theta := \{\theta_e\}_{e \in E}$ are not explicitly used in rendering, but can be distinguished in contacting objects $i, j \in O$

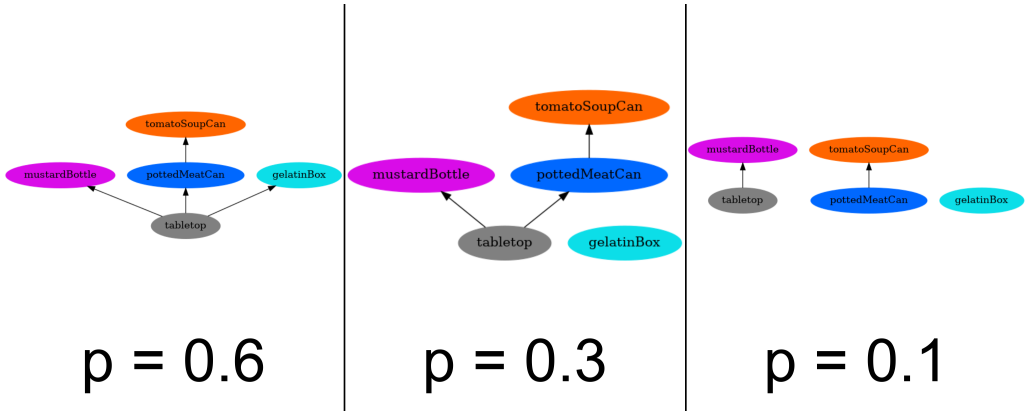


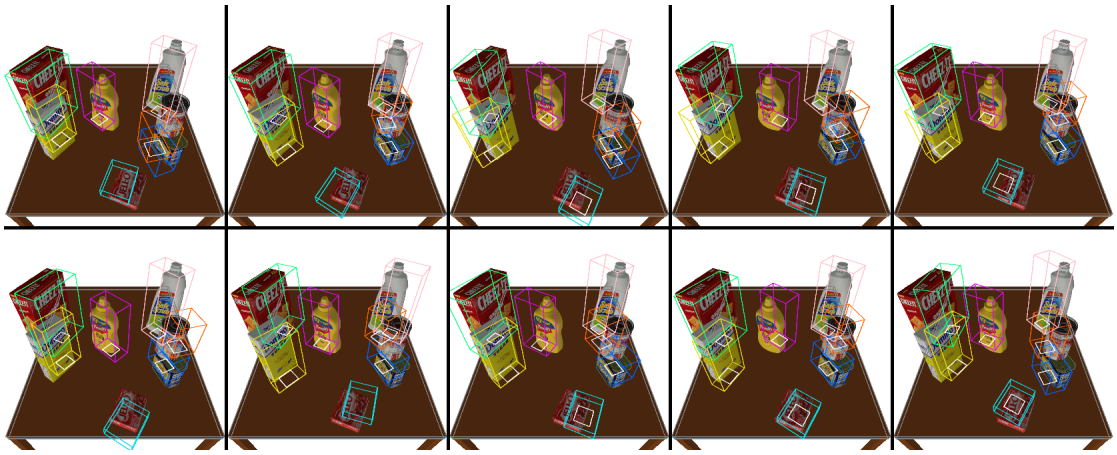
Figure 3-2: Visualization of a distribution over abstract scene graph structure using Graphviz.

by the distance in their corresponding wireframe renderings. Contact edges $e \in E$ are visualized as white squares, located at the center of face F_i. Face F_j is not directly visualized, but can often be inferred from which face is closest to F_i. This does not preclude potential ambiguity for certain pathological slack terms that rotate i close to a multiple of $\pi/2$. However, in practical modeling applications we find such occurrences to be rare, as sensible priors weight heavily against large slack terms.

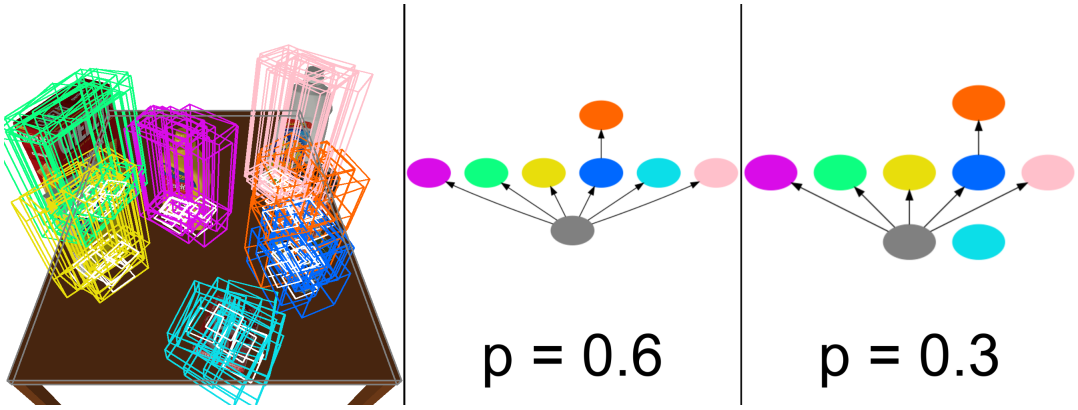
3.2.2 Distributions over Structure Beliefs

As we will quickly see, using the utility shown in Figure 3-1 quickly runs into issues displaying multiple structures over the same image. Since edges are rendered roughly in the same place, it can be hard to tell exactly how many samples have an edge between any two given objects. Even more, it's virtually impossible to tell which vertices and edges are part of the same scene graph. We develop a utility based off the Graphviz [**?**] graph visualization library to more clearly view different discrete structures. It accepts a distribution over scene graphs, and renders a specified number of the most probable present in a scene. In Figure 3-2, we can see how this clearly represents the top competing hypotheses for scene graph structure. This tool can be combined with the wireframe overlay to give rich visualizations of distributions over scene graphs.

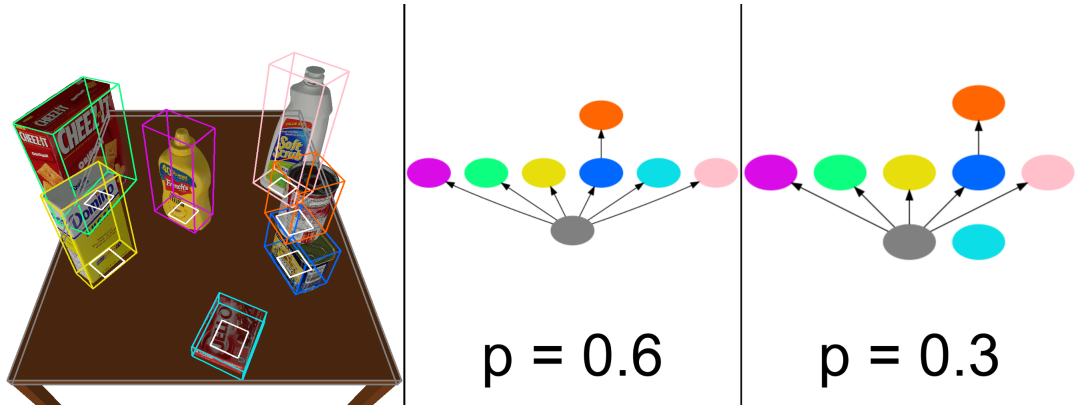
3.2.3 Distributions over Scene Beliefs



(a) Visualization of each sample's state using our scene graph visualization utility.



(b) All 10 samples, rendered over the same image. Only the two most frequent structures are shown.



(c) Aggregated visualization of all the samples. Only the two most frequent structures are shown.

Figure 3-3: Various methods for visualizing multiple beliefs. Node labels are omitted in the rendered structure distribution, and instead the wireframe rendering and GraphViz node for an object o share the same color.

We can now combine our utilities to render rich views of multiple samples. To generate our example visualization distribution, we generate 10 scene graphs $\{\mathcal{G}_i\}_{i\in 1:10}$, with object parameters sampled from a uniform distribution centered at the true scene graph as seen in Figure 3-1. Figure 3-3a shows the simplest form of visualizing the collection of samples, as a collage of separate renderings of each sample. This provides the most information, but at a glance it's difficult to get a sense of the mean and uncertainty of the distribution its representing.

We instead try aggregating the separate samples into a single representative "belief" $\mathcal{G}^* = \text{agg}(\{\mathcal{G}_i\}_{i\in 1:N})$ that is then overlaid on top of the rendered scene. We can then composite this with the abstract structure visualization to improve our ability to see uncertainty in scene structure.

One possible aggregation method is simply taking the disjoint union $\text{agg}(\{\mathcal{G}_i\}_{i\in 1:N}) = \bigsqcup_{i=1}^{N} \mathcal{G}_i$. As we see in Figure 3-3b, this has the effect of overlaying all of the beliefs on top of the same image. This can give a sense of the amount of uncertainty in object positions and rotations, but obscures the underlying object, making it difficult to determine the accuracy of the pose beliefs.

The second method we try $\text{agg}(\{\mathcal{G}_i\}_{i\in 1:N}) = (G^*, \Theta^*, Z^*)$ first takes the most probable structure $G^* = \text{argmax}_G\ \hat{p}(G, \Theta, Z)$ among our samples. The discrete parameters Z^* are then selected arbitrarily from one of the graphs that has this structure. We then let the continuous parameters Θ^* be determined by the average of that scene's objects' absolute poses $x_o^* = \text{avg}(\{x_{(i,o)}\}_{i\in 1:N})$ across all other samples (the average position uses a simple sum, while the average orientation is calculated using the method described by Markley et. al in [**?**]). Figure 3-3c shows an example of this method. In contrast to the Figure 3-3b, this method clearly shows the accuracy of pose beliefs, but loses information about uncertainty in the continuous parameters. These trade-offs mean choosing a visualization is somewhat contextually dependent on if the accuracy or uncertainty is more relevant.

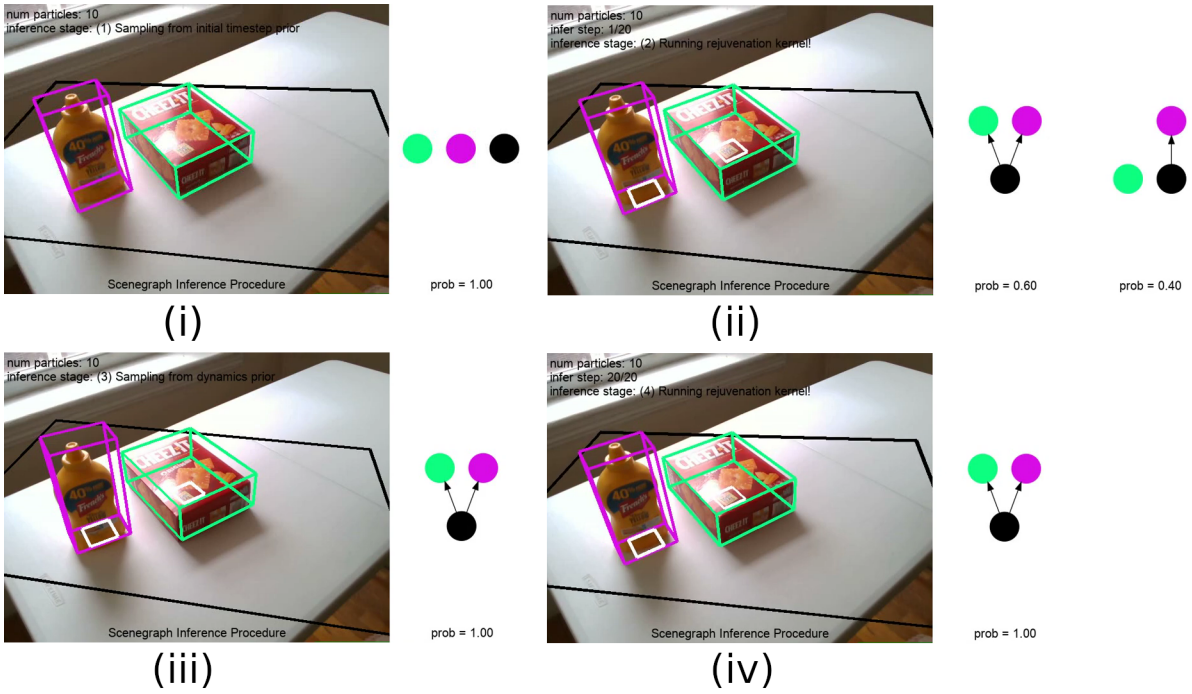


Figure 3-4: Visualization of sampling and rejuvenation moves in a particle filter. A complete video of this scene can be seen at: `https://www.youtube.com/watch?v=0_0TvrGC65Q`. (i) shows the first time step with the overlaid beliefs from the initial time step prior (initialized to the observed neural pose estimates), before structure inference. (ii) shows the beliefs over the first time step, after running 1 step of the rejuvenation kernel. (iii) shows the beliefs over the second time step, after sampling from the dynamics model. (iv) shows the beliefs over the second time step, after running 20 steps of the rejuvenation kernel.

3.2.4 Visualizing Inference in a Particle Filter

Finally, we demonstrate a real-world application by showing the actual usage of these visualization utilities in a particle filter with a corresponding complex inference program. The scene graph model we used leverages the prior from TODO: cite prior scene graph model section and dynamics from TODO: cite scene graph model dynamics. Object poses are initialized to their observed neural detections in the first time step. The inference program leverages a rejuvenation MCMC kernel after sampling from the prior for the first and second time step. This kernel is in turn composed of an interleaved RJMCMC kernel (see TODO: cite RJMCMC section) for discrete structure, and a drift kernel for continuous parameters. We run the

particle filter with 10 particles, and 20 iterations of the rejuvenation kernel. Figure 3-4 leverages the visualization method seen in Figure 3-3c to aggregate the particle filter state into a stable visual estimate of the mean object poses, and to view the distribution over inferred structure. This example shows how we can leverage the methods introduced in this chapter in practice for visualizing and debugging complex inference programs.

Chapter 4

Tests of Inference by Enumeration

4.1 Enumerative Inference

4.2 Slack Parameters

4.3 Analysis of Contact Model Parameters

4.4 Outlier Poses in Blackbox Neural Detectors

4.5 Reversible-Jump MCMC for Structure Inference

4.6 Lessons for Improving Scene Graph Priors

4.6.1 Sensitivity of model quality to hyperparameters

4.6.2 Quantitative analysis via average likelihood and marginal likelihood

4.6.3 Learning prior hyperparameters from data

Chapter 5

Real-world Tests

5.1 Metrics

5.1.1 Marginal Likelihood for Modeling

In our datasets, we use the average log-likelihood as our metric for modeling accuracy. Given our scene graph model m and hyperparameters θ, the log-likelihood is:

$$\mathcal{L} = \sum_{i=1}^{N} \log p(t_i; x, \theta)$$

5.2 YCB-Video

TODO: replace me

5.3 MIT In-House

5.3.1 Data Specification

The YCB-Video data set is a common benchmark for 6 degree of freedom pose estimation, but contains a relatively small number of simple structures. We'd like to generate additional benchmarks and data for tuning hyperparameters for a richer class of scenes.

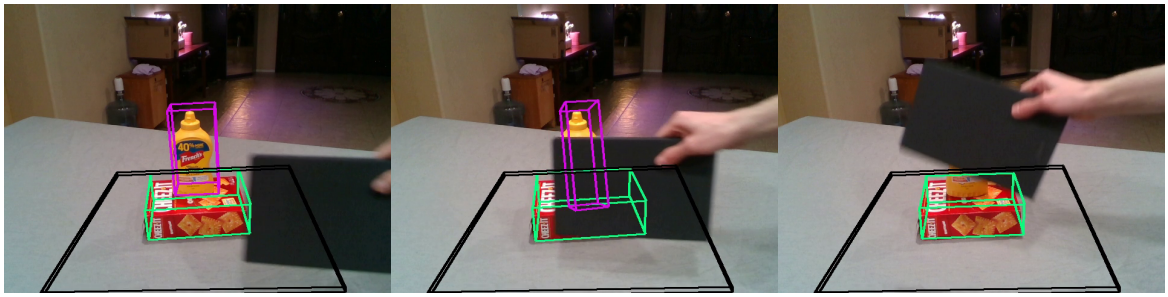


Figure 5-1: A selection of frames from sequence #1, out of 8 captured videos. Each sequence contains a different static structure. We use a dynamic occluder to generate failure modes in the neural network, in which objects flicker out of existence, or have their poses estimated incorrectly with very large error.

Name	**Data**	**Description**
Observed poses	(String, VectorPose)	nVidia Deep Object Pose Estimator neural detections
Ground truth outlier status	(String, VectorBool)	Per-Frame, per-object flag indicating if an object's neurally generated pose estimate is a noisy outlier
Ground truth structure	Directed Forest	Observed static structure for the scene
Ground truth poses	(String, Pose)	Approximate ground truth static pose, generated as average of inlier DOPE detections

Table 5.1: Description of data collected in our real-world experiments with YCB objects.

We collect a set of experiments with static objects to analyze the model in the static case. We introduce noise via occluders that block the camera's line of sight to the objects. The raw video was collected using the Intel RealSense D435 RGB/Depth camera. Only the RGB information is used for the neural detector, but we collect depth data to enable plane detection, and for future tests on models containing likelihood over depth data. Observed poses are generated using the publicly provided nVidia DOPE detector GitHub repository. Ground truth outlier status is manually annotated for each frame in the collected videos. Note that using the vertices, we can also use the scene graph structure to determine which detections are false positives or negatives. Ground truth structure is manually annotated once per scene, and the ground truth poses are estimated as the average over all an object's detected *inlier*

poses. While this can potentially obscure systematic bias across all neural detections in the data set, in nonetheless provides a useful relative analysis of the stability of the neural detector in the presence of varying levels of occlusion. In improving the static version of the model, this could be used as a method to quickly scale the amount of available data for learning model hyperparameters, or even observational model structure.

5.3.2 Contact and Noise Hyperparameters

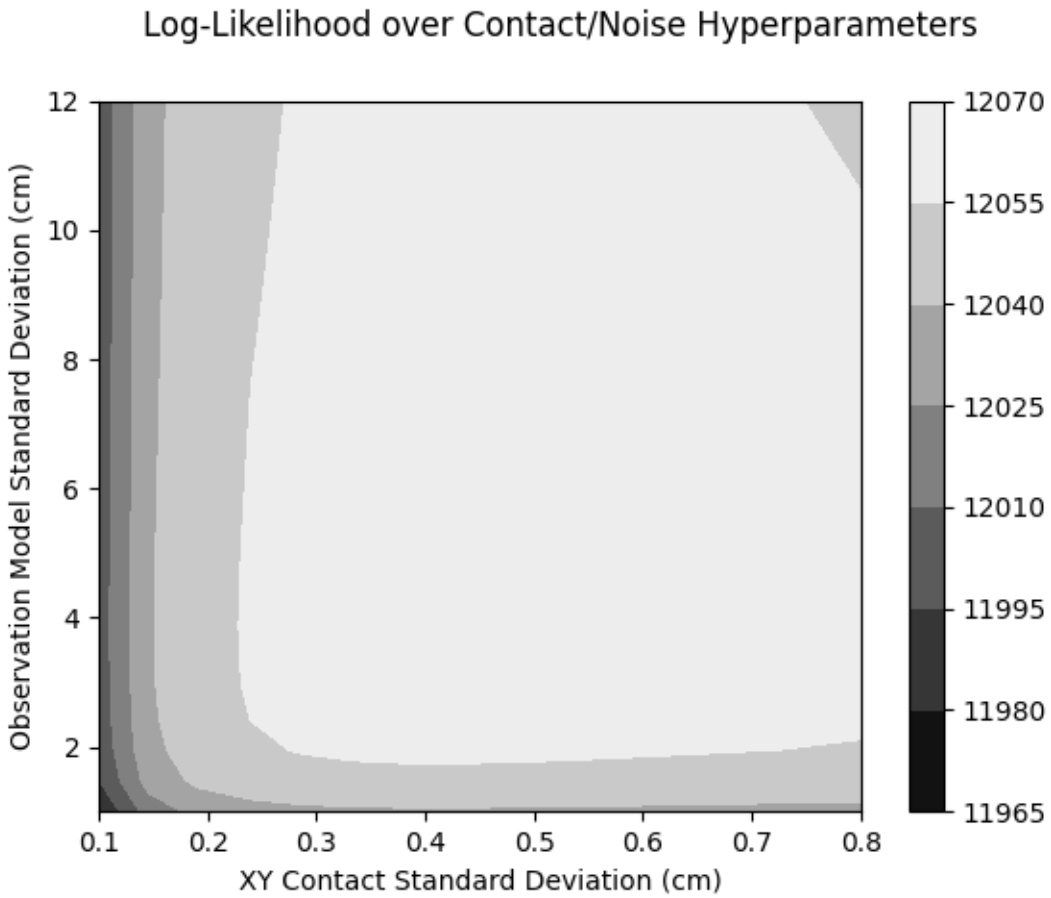


Figure 5-2: Log-likelihood landscape over a grid enumeration of the xy contact model standard deviation, and the observational model standard deviation. The maximum likelihood estimate is located at the star (TODO: Plot the ML estimate for observational model standard deviation versus xy contact standard deviation)

TODO: replace me

5.3.3 Inlier Neural Detection Observational Model

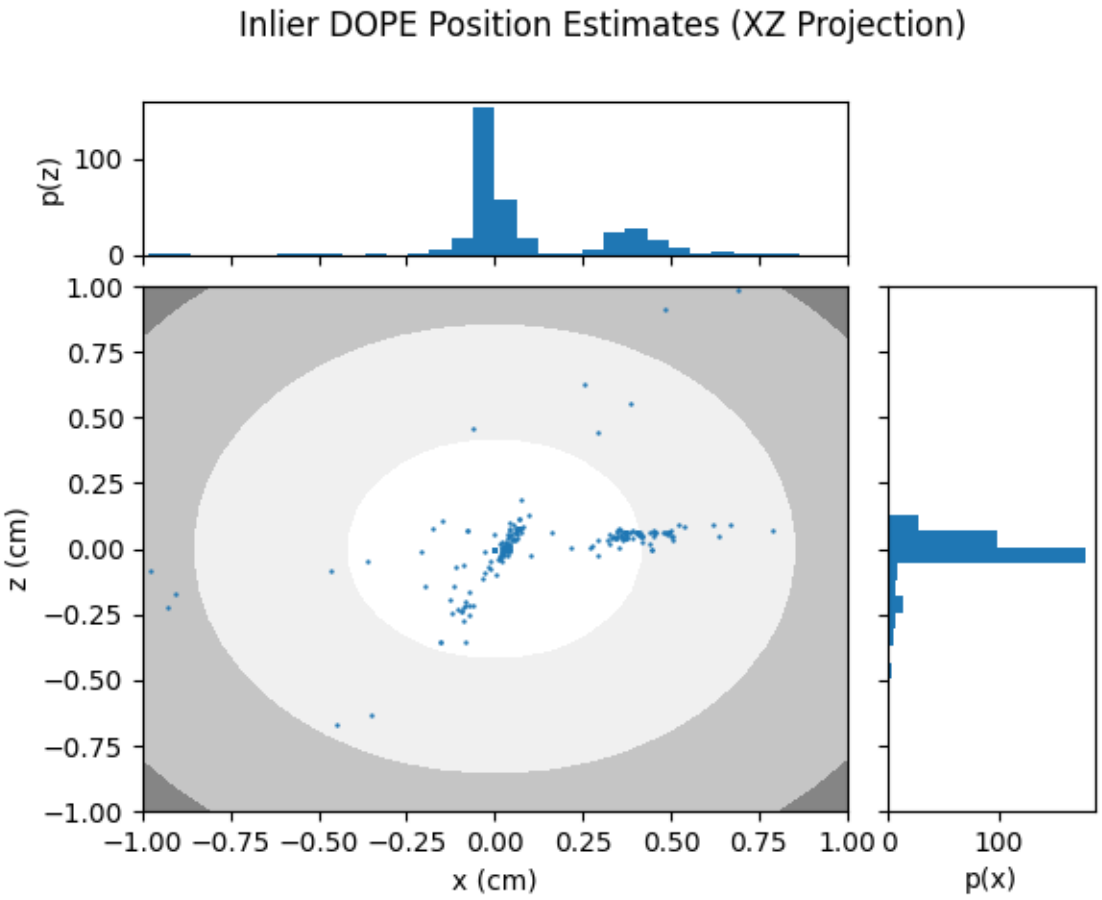


Figure 5-3: Scatter plot of the projection of nVidia DOPE's observed pose estimates to the x and z position axes, along with corresponding marginal histograms for each axis respectively. The plotted contour is the overlaid noisy observation model for inlier detections (in the projection, a 2D multivariate normal distribution), with the standard deviation hyperparameter set to the maximum-likelihood value as determined in the grid search above. Additionally shown are the marginal histograms over the observed poses.

Figure 5-3 provides a comparison of a simple observational model in the positional dimensions, fit to maximum likelihood with respect to inlier pose detections. In small deviations from the ground truth, the neural pose estimates exhibit systematic bias in error that could potentially be more accurately represented with a finer-grained observational model.

Chapter 6

Conclusion

TODO: replace me

Appendix A

Tables

Table A.1: Armadillos

Armadillos	are
our	friends

Appendix B

Figures

Figure B-1: Armadillo slaying lawyer.

Figure B-2: Armadillo eradicating national debt.

Bibliography

[1] Varun Jampani, Sebastian Nowozin, Matthew Loper, and Peter V. Gehler. The informed sampler: A discriminative approach to bayesian inference in generative computer vision models. *CoRR*, abs/1402.0859, 2014.

[2] Tejas D Kulkarni, Pushmeet Kohli, Joshua B Tenenbaum, and Vikash Mansinghka. Picture: A probabilistic programming language for scene perception. In *Proceedings of the ieee conference on computer vision and pattern recognition*, pages 4390–4399, 2015.

[3] Abhijit Kundu, Yin Li, and James M Rehg. 3d-rcnn: Instance-level 3d object reconstruction via render-and-compare. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3559–3568, 2018.

[4] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. Deep object pose estimation for semantic robotic grasping of household objects. *CoRR*, abs/1809.10790, 2018.

[5] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *arXiv preprint arXiv:1711.00199*, 2017.

[6] Ilker Yildirim, Tejas D Kulkarni, Winrich A Freiwald, and Joshua B Tenenbaum. Efficient analysis-by-synthesis in vision: A computational framework, behavioral tests, and comparison with neural representations. In *Proceedings of the thirty-seventh annual conference of the cognitive science society*, 2015.

[7] Ben Zinberg, Marco Cusumano-Towner, and Vikash K. Mansingkha. Structured differentiable models of 3d scenes via generative scene graphs. *Workshop on Perception as Generative Reasoning, NeurIPS 2019, Vancouver, Canada.*, 2019.

@articlemarkley2007averaging, title=Averaging quaternions, author=Markley, F Landis and Cheng, Yang and Crassidis, John L and Oshman, Yaakov, journal=Journal of Guidance, Control, and Dynamics, volume=30, number=4, pages=1193–1197, year=2007

@INPROCEEDINGSEllson03graphvizand, author = John Ellson and Emden R. Gansner and Eleftherios Koutsofios and Stephen C. North and Gordon Woodhull, title = Graphviz and dynagraph – static and dynamic graph drawing tools, booktitle = GRAPH DRAWING SOFTWARE, year = 2003, pages = 127–148, publisher = Springer-Verlag