3D Scene Graph Inference and Refinement for Vision-as-Inverse-Graphics

Lukasz Romaszko

Doctor of Philosophy

Institute for Adaptive and Neural Computation

School of Informatics

University of Edinburgh

2019

Abstract

The goal of scene understanding is to interpret images, so as to infer the objects present in a scene, their poses and fine-grained details. This thesis focuses on methods that can provide a much more detailed explanation of the scene than standard bounding-boxes or pixel-level segmentation – we infer the underlying 3D scene given only its projection in the form of a single image.

We employ the Vision-as-Inverse-Graphics (VIG) paradigm, which (a) infers the latent variables of a scene such as the objects present and their properties as well as the lighting and the camera, and (b) renders these latent variables to reconstruct the input image. One highly attractive aspect of the VIG approach is that it produces a compact and interpretable representation of the 3D scene in terms of an arbitrary number of objects, called a 'scene graph'. This representation is of a key importance, as it can be useful e.g. if we wish to edit, refine, interpret the scene or interact with it.

First, we investigate how the recognition models can be used to infer the scene graph given only a single RGB image. These models are trained using realistic synthetic images and corresponding ground truth scene graphs, obtained from a rich stochastic scene generator. Once the objects have been detected, each object detection is further processed using neural networks to predict the object and global latent variables. This allows computing of object poses and sizes in 3D scene coordinates, given the camera parameters. This inference of the latent variables in the form of a 3D scene graph acts like the encoder of an autoencoder, with graphics rendering as the decoder.

One of the major challenges is the problem of placing the detected objects in 3D at a reasonable size and distance with respect to the single camera, the parameters of which are unknown. Previous VIG approaches for multiple objects usually only considered a fixed camera, while we allow for variable camera pose. To infer the camera parameters given the votes cast by the detected objects, we introduce a Probabilistic HoughNets framework for combining probabilistic votes, robustified with an outlier model. Each detection provides one noisy low-dimensional manifold in the Hough space, and by intersecting them probabilistically we reduce the uncertainty on the camera parameters.

Given an initialization of a scene graph, its refinement typically involves computationally expensive and inefficient search through the latent space. Since optimization of the 3D scene corresponding to an image is a challenging task even for a few LVs, previous work for multi-object scenes considered only refinement of the geometry, but not the appearance or illumination. To overcome this issue, we develop a framework called ‘Learning Direct Optimization’ (LiDO) for optimization of the latent variables of a multi-object scene. Instead of minimizing an error metric that compares observed image and the render, this optimization is driven by neural networks that make use of the auto-context in the form of a current scene graph and its render to predict the LV update. Our experiments show that the LiDO method converges rapidly as it does not need to perform a search on the error landscape, produces better solutions than error-based competitors, and is able to handle the mismatch between the data and the fitted scene model. We apply LiDO to a realistic synthetic dataset, and show that the method transfers to work well with real images. The advantages of LiDO mean that it could be a critical component in the development of future vision-as-inverse-graphics systems.

Lay Summary

The goal of scene understanding is to interpret images, so as to understand the objects present in a scene and their properties. This thesis focuses on methods that can provide a much more detailed explanation of the scene than standard bounding-boxes or pixel-level segmentation – we predict the underlying 3D scene given only a single image.

We employ the Vision-as-Inverse-Graphics (VIG) approach, which (a) infers the scene parameters such as the objects present and their properties as well as the lighting and the camera, and (b) renders these 3D scene to reconstruct the input image. One highly attractive aspect of the VIG approach is that it produces a compact and interpretable representation of the 3D scene in terms of a number of objects, called a 'scene graph'. This representation is of a key importance, as it can be useful e.g. if we wish to edit, refine, interpret the scene or interact with it.

First, we investigate how the recognition models can be used to understand the scene graph given only a single image. These models are trained using realistic synthetic images and corresponding ground truth scene graphs, obtained from a scene generator. Once the objects have been detected, neural networks are used to predict the object and global parameters of a 3D scene. This allows compression of an image into a scene graph representation, and then rendering it back to reconstruct the image.

One of the major challenges is the problem of placing the detected objects in 3D at a reasonable size and distance with respect to the single camera, the parameters of which are unknown. To infer the camera parameters given the votes cast by the detected objects, we introduce a Probabilistic HoughNets framework for combining probabilistic votes. Each detection provides one noisy vote for the camera parameters, and by intersecting them we increase the accuracy of the prediction.

Given an initialization of a scene graph, its refinement typically involves computationally expensive and inefficient search through the latent space. To overcome this issue, we develop a framework called 'Learning Direct Optimization' (LiDO). This optimization is driven by neural networks that make use of the current scene graph and its render to update the parameters. Our experiments show that the LiDO method converges rapidly and produces better solutions than competitors. We apply LiDO to a realistic synthetic dataset, and show also that the method transfers to work well with real images. The advantages of LiDO mean that it could be a critical component in the development of future vision-as-inverse-graphics systems.

Acknowledgements

I am deeply grateful to my supervisor Chris Williams, with whom I had the privilege to work with for four years. His broad knowledge and attention to very detail have greatly advanced and enriched my research. He has also given me a large amount of patience when discussing the technical aspects and then the details of the write-up. He has also provided me with the opportunities to learn numerous skills relevant to researchers, and to attend three excellent summer schools in Canada, England and Italy, and computer vision conferences.

I would like to thank my thesis examiners, Robert Fisher and Timothy Cootes, for providing rich and valuable feedback.

I am very grateful for the Microsoft Research 2014 PhD Scholarship provided by the joint initiative of the School of Informatics of the University of Edinburgh and Microsoft Research Cambridge. I thank also the Microsoft Research advisor, John Winn for useful comments and discussions. I would also like to thank the staff and the current and past PhD students from the Institute for Adaptive and Neural Computation, and the staff of the Institute of Perception, Action and Behaviour, for their support and feedback.

I am grateful to my parents Krystyna and Albert for their support for my entire life, as well as patience and encouragement during my PhD studies. Finally, to Agnieszka for her love and support, especially during our long long-distance time.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

7th April 2020

(*Lukasz Romaszko*)

To my parents.

Contents

Glossary

Acronyms

CNN – Convolutional Neural Network

GBO – Gradient-Based Optimization

GMM – Gaussian Mixture Model

GT – Ground Truth

HT – Hough Transform

LiDO – Learning Direct Optimization

LV – Latent Variable

MAP – Maximum a Posteriori

MSE – Mean Squared Error

PHN – Probabilistic HoughNet

PHT – Probabilistic Hough Transform

RGB – Red, Green, Blue (colour channels)

Sim – Simplex Optimizer

SSG – Stochastic Scene Generator

VGG – VGG-16 convolutional neural network (Simonyan and Zisserman, 2015)

VIG – Vision-as-Inverse-Graphics

Papers

3DParsing – *Holistic 3D Scene Parsing* paper (Huang et al., 2018)

AIR – *Attend, Infer, Repeat* paper (Esalmi et al., 2016)

IM2CAD – *IM2CAD* paper (Izadinia et al., 2017)

NSD – *Neural Scene De-Rendering* paper (Wu et al., 2017a)

Chapter 1

Introduction

The goal of scene understanding is to interpret images, so as to infer the underlying 3D scene in terms of objects present and their properties as well as the lighting and the camera parameters. However, obtaining an interpretable representation from a 2D image is a complex and an ill-posed task. Solving this task requires embedding certain prior knowledge about the world and image formation, as well as combining evidence from multiple cues to allow for inference of a likely scene configuration and object descriptions. Humans are not only able to easily interpret clean images, but also to understand the scene under unusual types of illumination and heavy occlusions, due to their knowledge about objects and their possible appearances from different views.

Early computer vision research was focussed on images with simple and clean objects. The representational gap (Dickinson, 2009) between the image pixels and 3D shape representation caused difficulties in e.g. generating arbitrary 3D shapes, matching them properly to the observed image, and distinguishing object edges from texture. This led some computer vision researchers into abandoning of 3D reconstruction and solving simpler problems such as providing object detections or pixel-level segmentation in the image frame. To this end, numerous approaches have been proposed, and consequently a wide range of image interpretation tasks have been designed to evaluate their quality, such as object bounding-box detection or semantic instance segmentation. However, these provide only flat 2D information. Similar limitations apply to some approaches that predict depth and surface normals, which output predictions in a per-pixel manner.

The main theme of the thesis is to provide a much more detailed explanation of the scene than bounding-boxes or a pixel-level segmentation – we seek to infer the underlying 3D scene given only its projection in the form of a single RGB image. The representation of the scene is interpretable, by which we mean that it is editable, where one could e.g. rotate an object, or observe the scene from a different viewpoint. In addition to inference of the scene's composition, we seek a description of properties of the objects: their shapes, poses, attributes, parts present and appearances. Much of the literature for the 3D reconstruction task focuses only on representations that are not interpretable and do not provide a full 3D scene explanation, or work only for specific objects with a fixed texture (not object classes) – the relevant literature is discussed in Chapter 2. Another strand of literature considers a richer input than a single image; either using a depth channel, or multiple images of a scene from different viewpoints. In contrast, we consider more challenging problem where only a single RGB image is observed.

The problem of vision is extremely complex: objects of a specific object class can have different shapes and appearances, can be seen from different views, be occluded, may look different depending on the illumination environment and the viewpoint, etc. For instance, objects which occlude others might be difficult to segment when they have similar colours. Therefore, providing of a prior knowledge about the size, shape and appearance is crucial, yet difficult, as all the factors can widely vary. Finding a way to incorporate such prior knowledge into algorithms is a challenging task, yet crucial for successful inference.

1.1 Vision-as-Inverse-Graphics

Our work is carried out in the vision-as-inverse-graphics (VIG) or analysis-by-synthesis paradigm, where the vision task is the inverse of the rendering process. One not only extracts a 3D scene representation from the input image, but *renders* that scene to allow comparison with the input image. This inference of the 3D scene acts like the encoder of an autoencoder, with graphics rendering as the decoder, as shown in Figure 1.1. VIG is a long-standing idea, see e.g. Grenander (1976, 1978); Stevens and Beveridge (2001); Yuille and Kersten (2006), but it can be reinvigorated using the power of deep learning for the analysis stages. Inference in generative models typically involves searching through the space of latent variables. Due to the complexity of scene configurations, we make use of supervised learning techniques to make the search more efficient. These methods rely on large quantity of annotated examples,

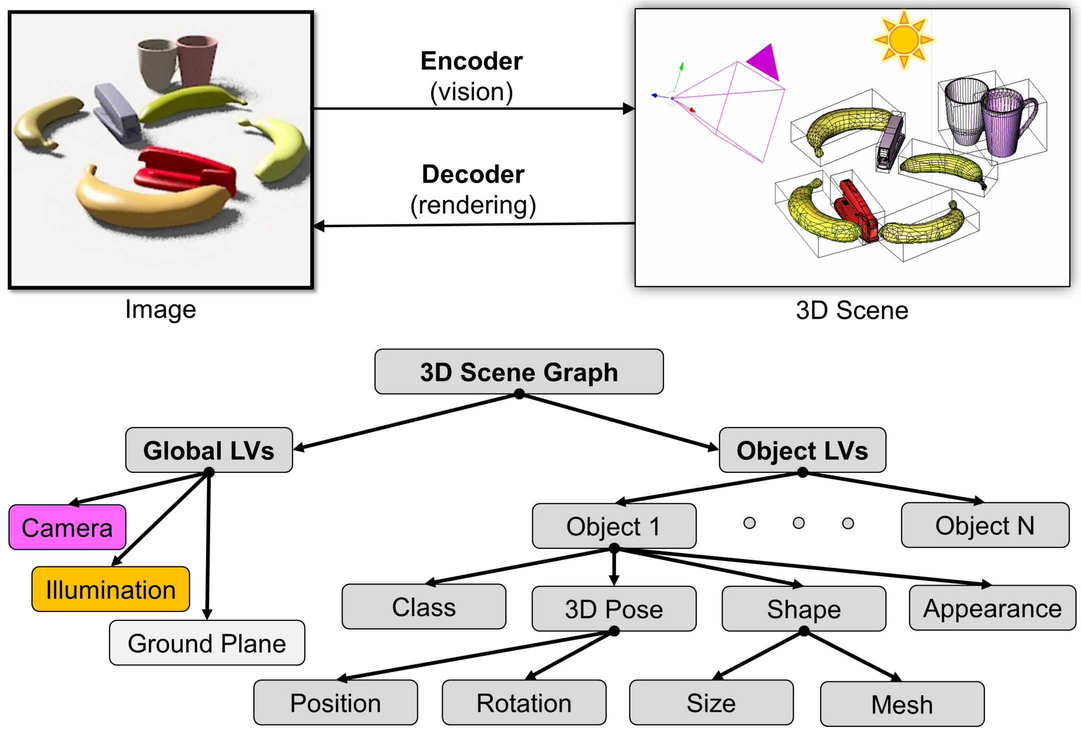


Figure 1.1: Vision-as-Inverse-Graphics paradigm: given an observed image (top left), an interpretable scene graph representation of a 3D scene (top right) is inferred by the encoder, this can be then rendered by the decoder. We specify the decoder in a form of computer graphics renderer and the vision task is then to learn to invert the rendering process. Bottom: outline of the 3D scene graph considered in this thesis.

which we can generate using a synthetic scene generator. One of the aspects of the VIG approach is the possibility to make use of modern 3D computer graphics engines at inference time, as used in recent VIG literature that considered a single object (Kulkarni et al., 2015a; Shotton et al., 2011; Yildirim et al., 2015; Jampani et al., 2015).

One highly attractive aspect of the VIG approach is that it produces a compact and interpretable representation of the 3D scene in terms of an arbitrary number of objects (as shown in Figure 1.1, right). This representation is called a 'scene graph' (see e.g. Angel 2003, sec. 9.8) as used in computer graphics for describing a scene in terms of objects as well as lighting and cameras. We specify the "3D scene graph" as a collection of the scene latent variables representing a 3D scene, to differentiate from scene graphs considering 2D vector graphics. This representation is of a key importance, as it is useful e.g. if we wish to edit, refine, interpret the scene or interact with it.

The problem of 3D scene reconstruction from a single image that we investigate is

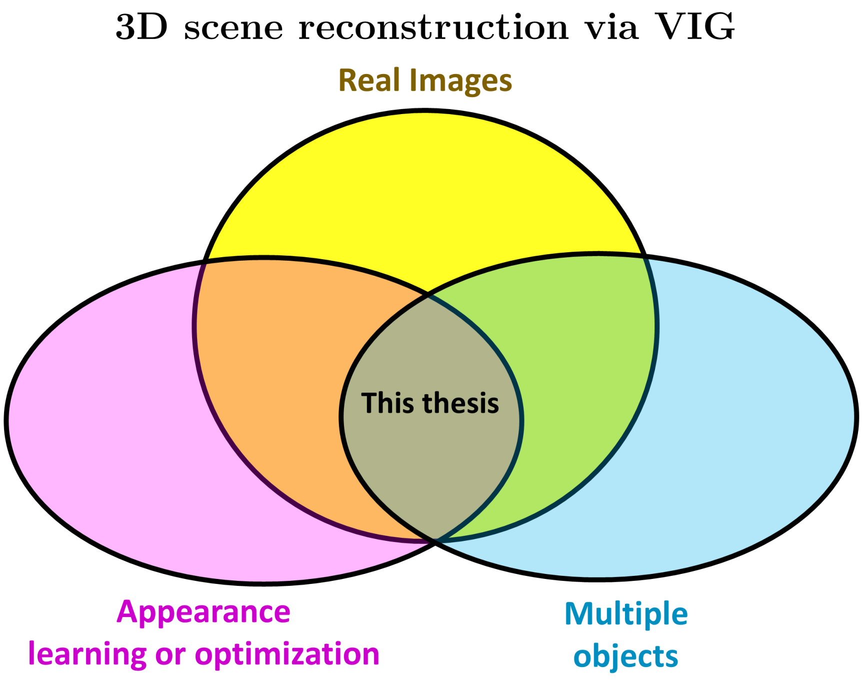

3D scene reconstruction via VIG
Real Images
This thesis
Appearance learning or optimization
Multiple objects

Figure 1.2: The three main properties of 3D reconstruction via VIG considered in this thesis. We handle real images of multi-object scenes and obtain a rich 3D scene representation that includes inference of the appearance and illumination.

very challenging. We build methods that can produce a richer representation than the existing VIG approaches for scenes containing multiple objects. Related work that predicts a 3D scene representation for multi-object scenes has usually considered simplified scenarios, either working only with objects of fixed identity (thus with a known appearance and size), or by focusing only on geometry matching, but not searching over appearance and illumination. Often only toy artificial datasets are used, so as the models work only for the images used during training, but are not applicable to real images. This thesis develops methods to provide a rich framework for scene understanding via VIG without the above limitations, to enable handling of real images, and serve as a base for further development of VIG approaches. Figure 1.2 outlines the three main properties of the problem we solve: the ability to handle real images at test time (including dealing with variable camera pose and object appearance), scenes containing multiple objects, and inference of the scene appearance (including object colours and illumination), so the output is a similar RGB image to the input. The next section provides the details of the claims and of the contributions.

1.2 Thesis Outline

The further chapters of the thesis are organised as follows:

Chapter 2 provides ***Background*** on the history of computer vision, a general overview of the computer vision literature for object explanation, and comparison to the related work for image interpretation, including the approaches that incorporate the VIG paradigm.

Chapter 3 entitled ***Rich 3D Explanation of a Multi-Object Scene via Inverse Graphics*** considers inference of a 3D scene graph from a single RGB image. The models are trained using realistic synthetic images and corresponding ground truth scene graphs, obtained from a rich stochastic scene generator, to predict the object and global latent variables. In the framework we incorporate auxiliary modules for object contact point and projection scale prediction, to allow the computation of object poses and sizes in 3D scene coordinates, given the camera parameters. The contributions of this chapter are:

- We show that we can successfully reconstruct a 3D scene via VIG from a single RGB image (for a given set of object classes).
- We demonstrate the quality of the reconstructions obtained quantitatively on synthetic data, and qualitatively on real scenes.

Chapter 4 entitled ***Probabilistic HoughNets*** considers a new method for combining of the probabilistic votes. The method is applied to the camera estimation problem, where the votes are cast by the detected objects. Each detection provides one noisy low-dimensional manifold in the Hough space, and by intersecting them probabilistically we reduce the uncertainty on the camera parameters. Previous VIG approaches for multiple objects usually only considered a fixed camera, while we allow for variable camera pose and place the detected objects in 3D at a reasonable size and distance with respect to the single camera. The contributions of this chapter are:

- We introduce PHNs, a new method for combining of the probabilistic votes.
- We show that the models that predict directly given the whole image, rather than combining information from the detected objects perform poorly.
- We show that combination using PHNs outperforms standard combination methods using the same CNN architecture.

- We show that the PHNs are robust in case of misdetections.

Chapter 5 entitled ***Learning Direct Optimization for Scene Understanding*** thoroughly investigates the problem of iterative scene refinement. Since optimization of the 3D scene corresponding to an image is a challenging task even for a few latent variables, previous work for multi-object scenes considered only refinement of the geometry, but not the appearance or illumination. The standard way to proceed is to measure the error between the observed image and the predicted render, and use an optimizer to minimize the error. However, it is unknown which error measure would be most effective for simultaneously addressing issues such as misaligned objects, occlusions, textures, etc. To overcome this issue, we develop a framework called 'Learning Direct Optimization' (LiDO) for optimization of the latent variables of a multi-object scene. Instead of minimizing an error metric that compares the observed image and the render, this optimization is driven by neural networks that make use of the auto-context in the form of a current scene graph and its render to predict the latent variable update. The experiments have been performed both on synthetic and real datasets. This allows for a smarter refinement, for example that is robust to occlusions, and can ignore the noise such as a mismatch in texture while refining the object pose.

The contributions of this chapter are:

- We introduce LiDO, a new method for optimization of the latent variables of a multi-object scene.
- We introduce the first method that refines the geometry, appearance and illumination for multi-object scene at the same time.
- We verify that LiDO is much faster than the standard optimizers.
- We verify that LiDO performs significantly better than the standard optimizers in terms of various error measures.

Chapter 6 provides ***Conclusion and Future Work***. We summarize the thesis results and contributions, and outline several interesting directions for the future developments of the VIG paradigm.

Appendix A provides a chapter entitled ***Modelling Shape Using a Skeleton-Based 3D Deformable Mesh***. A deformable generative shape model is developed that can model a wide range of object shape variability, as well as generalizing to generate new shapes of a given object class. This is done by specifying an object template mesh and a skeleton (consisting of a number of “bones”), and implementing the deformation to be automatically-differentiable so as to fit the template to a collection of 3D mesh instances using a gradient-based optimizer. The skeleton is controlled by joint locations, and we specify methods for automatic computation of bone rotation parameters. This allows for robust local gradient-based optimization where the parametrization of the child bone joint location are independent of the parent. Importantly, such a skeleton-based representation is interpretable, and could be used in VIG approaches, where one may want to explain an object by fitting its render to the observed image, and sample new object shapes for the purpose of generation of a large amount of synthetic data. We also demonstrate the performance of our generative shape model on the object completion and constrained sampling tasks.

1.3 Statement of the Contributions

The work in each chapter was primarily supervised and guided throughout by Chris Williams. Below we outline the work done by additional contributors.

Chapter 3 (VIG Framework) All the implementation of the detector and initialization networks was done by myself. The initial code for importing CAD models and generating and rendering scenes with multiple objects was provided by Pol Moreno. These have been thoroughly extended and improved by me for the purposes of this project.

All the experiments related to the detector and the initialization networks were conducted by myself. Pol Moreno conducted the gradient-based optimization experiments. These additional experiments were done given the scene graph initializations obtained from the VIG framework developed in this thesis.

Pushmeet Kohli was a co-investigator of this project and provided the initial guidance.

Chapter 4 (PHNs) The PHNs implementation and experiments were conducted entirely by myself.

Chapter 5 (LiDO) The LiDO implementation and experiments were conducted entirely by myself.

John Winn played a key part in the formulation of LiDO, provided guidance via video calls, and commented on the drafts of the paper.

Publications The work in Chapter 3 and Chapter 4 is published as: Romaszko, L., Williams, C. K. I., Moreno, P., and Kohli, P. (2017). Vision-as-Inverse-Graphics: Obtaining a Rich 3D Explanation of a Scene from a Single Image. In *ICCV 2017 Geometry Meets Deep Learning Workshop*, pages 851–859, 2017.

The work in Chapter 5 has been accepted for publication as: Romaszko, L., Williams, C. K. I., and Winn, J. (2020). Learning Direct Optimization for Scene Understanding. *Pattern Recognition*, 2020.

Chapter 2

Background

This chapter provides background on object recognition and scene understanding relevant to this thesis. Section 2.1 provides analysis of different approaches for scene representation (2D vs 3D), and Section 2.2 provides the discussion of the early as well as recent related work for object detection. In Section 2.3 we discuss generative models for image reconstruction. Finally, in Section 2.4 we compare the work carried out in this thesis to the related work on image interpretation, focusing on the approaches that incorporate the VIG paradigm. We overview the related literature by starting with autoencoders, then discussing methods that reconstruct a single object in 3D, and finally the methods that reconstruct multiple objects in 3D.

2.1 2D vs. 3D Representation

The key challenge of computer vision is how to find a representation of images which are 2D projections of 3D scenes. The early computer vision research focused on modelling the 3D representation of the objects, see e.g. the work of Marr and Nishihara (1978). One of the obstacles is that datasets consisted only of 2D images and there exists a significant gap between 2D and 3D representations (see e.g. Grimson and Lozano-Perez, 1984). Since the datasets consisted only of images, a 3D representation had to be model-based. The developed methods include approximating the shape with simpler shapes such as generalized cylinders (Brooks et al., 1979) or using a set of 3D CAD models (Tan et al., 1998). These were not able, however, to deal with objects of different appearances and textures, so images were often pre-processed to detect edges or other simpler representations, which in turn led to a significant loss of information.

For these reasons, currently the majority of computer vision approaches aim at directly extracting properties of objects from their view present in an image, without any underlying 3D shape model. These approaches explicitly explain the image pixels, i.e. they use a projection of a scene (such as usual real images) and try to learn the shape and appearance from it. Such models learn appearances of an object given a lot of examples of the given object class or their parts, seen from different views. For these approaches there were multiple tasks created, for example through the Pascal VOC Challenges (Everingham et al., 2010) which annually took place from 2005 until 2012. These tasks include classification (a binary classification for each class whether there is an object of this class present in the scene), object detection (predicting object bounding boxes for each object of a given class), object segmentation (assignment of coherent regions of pixels to respective objects or to the background). All these tasks consider only making predictions in the 2D image frame, as such annotations can be easily produced by image annotators. Such predictions do not provide any 3D explanation of the scene.

In contrast, humans understand the underlying scene, that is they immediately think of a 3D representation of the objects in the space. To bridge the gap between 2D and 3D representations, a depth map (or a Z-order) might be inferred to recover 3D scene configuration to some extent. For example, Hoiem et al. (2007) develop methods for labelling pixels with their Z-order, nevertheless such an enhancement does not solve the object detection problem. Recent work on this theme includes pixel depth prediction (Li et al., 2018a); exploiting 3D geometry by placing object detections into perspective at predicted scale and depth (Hoiem et al., 2008); and representing an indoor scene by furniture items described by 3D bounding-boxes (Choi et al., 2015).

A key difficulty is the problem of putting objects in 3D if the camera view is unknown, therefore most of the methods which explain scenes with multiple objects make use of a depth channel (Zou et al., 2019; Song and Xiao, 2014) or explain scenes using simple bounding boxes, for example R-CNN network (Girshick, 2015), sometimes predicting the camera view (Hoiem et al., 2008; Kar et al., 2015). Other works assume usage of objects of known shape and texture (e.g. Kendall et al., 2015). There are also several works that predict pose and overlaid 3D deformable model for each object in the image, but they are applied per object basis (Su et al., 2015; Tulsiani et al., 2016), so the view is predicted independently, with a separate camera for each object.

To perform accurate inference, it is helpful to incorporate the context of image regions. For instance, Torralba (2003) shows that given a street scene, humans recognize vertical blurry shapes as pedestrians and the same shapes rotated by 90 degrees as cars, even though alone they do not resemble any concrete object. In addition to the shape and pose, the scale and the location of an object suggests the most likely interpretation. Moreover, the context of the nearby objects is of high importance, thus incorporation of such information is crucial for successful image recognition. For example, a high accuracy of image labelling is obtained by the multiscale Conditional Random Fields (mCRF) approach (He et al., 2004). To improve the classifier predictions, the authors use contextual features and directly model patterns of label variables. These patterns are based on the commonly neighbouring classes or regions where a particular class usually tends to be present.

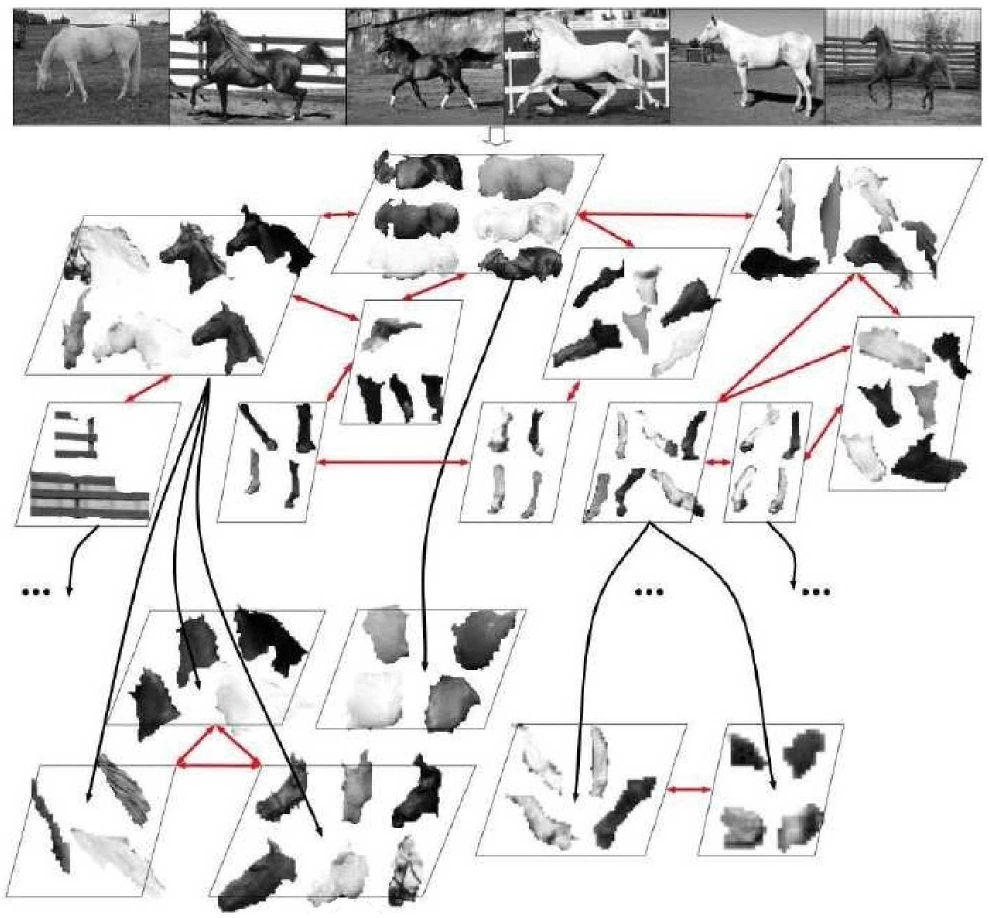


Figure 2.1: Learned hierarchical representation of a horse. From Ahuja and Todorovic (2008).

In addition to the contextual influence upon understanding of a particular image region, another key problem of vision is how to deal with intra-class shape and appearance variability. One approach is to decompose objects into smaller pieces and learn scale-invariant appearances of the parts (e.g. Fergus et al. 2003). Objects can look different not only due to shape and appearance variability of their parts, but also due to pose variation or having different structure. Figure 2.1 illustrates the system developed by Ahuja and Todorovic (2008) which employs a tree-based segmentation hierarchy to represent fragments of objects, which can handle a shape variability of non-rigid objects such as animals.

For a 3D model-based representation of a shape the simplest solution would be to use a set of 3D models of a given class. However, it is crucial to interpolate between shapes using a deformable model. A common approach to model a shape variability is to use Principal Component Analysis (PCA) to lower the dimension of the shape representation, e.g. Cootes et al. (1995) consider 2D shapes, and Blanz and Vetter (2003) develop a 3D morphable face model. An analogous generative approach can be employed to model objects as well as their parts using Probabilistic PCA (PPCA, Tipping and Bishop, 1999). Alternatively, a deformable mesh could be specified by a skeleton, which represent shapes in a natural way. This approach is common in the context of hand pose detection and body pose estimation, where an explicit skeletal model is defined and then the latent variables of the skeleton are inferred (Sharp et al., 2015). Other approaches establish correspondences between a set of 3D shapes and model the shape by a parametrized surface (Davies et al., 2002). Recently, several approaches learn a 3D shape model of an object class directly from manual annotations of real images, such as object segmentation masks and keypoints (Tulsiani et al., 2016; Kanazawa et al., 2018).

2.2 Discriminative Models for Object Detection

The first step in scene explanation is to detect the objects present in the scene. Discriminative recognition models map an image to features suitable for a given task, with the aim of reducing the search space by using a more informative feature representation than raw pixels. Usually a set of local features is detected within the whole image that together should contribute to a given shape. There were a variety of feature extraction methods developed, a common early one is the Scale-Invariant Feature

Transform (SIFT, Lowe, 1999). SIFT features make use of interest points which are extracted from the given images, often used to match to object templates. Later, a frequently employed image descriptors were Histograms of Oriented Gradients (HOG, Dalal and Triggs, 2005) which represent the types of gradients located in smaller cells of the image, highly invariant to the colour and the appearance. The features are usually obtained at multiple scales to provide scale-invariance. Vector-representations of the features, also called bag-of-features, were often used as an input to common classification models, such as Support Vector Machines (SVM).

Performing object detection is difficult since objects can look different and might be seen from many viewpoints. Therefore, a given class may be represented as a set of templates. For dealing with the template matching problem, one well known method is the Generalized Hough Transform (GHT) introduced by Ballard (1981) which performs voting for template pose latent variables by accumulation of evidence. The evidence can be obtained from the detected edges which are then used to predict the possible pose of the template shape. For example Leibe et al. (2004) describe the Implicit Shape Model where interest points are treated as hypotheses, employing GHT for template matching and a Minimal Description Length (MDL) criterion to limit inclusion of additional hypotheses if the potential improvement is too low. Although the GHT technique was successfully applied in this work, the trained model was applied only to the side views of cars.

To advance object detection, a lot of approaches focus on modelling parts of objects instead of rigid templates, as objects can exhibit high intra-class variability. This includes a variability in appearance of object parts, different kinds of parts that may be present, or self-occlusion. In 2010, the best results for the object detection in the PASCAL VOC Challenge were obtained by methods such as Deformable Parts Model (DPM, Felzenszwalb et al., 2010). This approach uses a star-structured, part-based model, which stores for each part templates of their shape and appearance. A detection whether each part is present is performed across the whole image using a sliding window approach. The confidence of the detections at the given locations are stored in the form of a grid, called a heat-map. The heat-maps for each object part are obtained at different resolutions and then combined. DPM relies on several aforementioned techniques, such as HOG, PCA and SVM.

Nowadays the best performing methods are deep learning-based ones. These methods use neural networks (NNs) to extract features from a large amount of examples, and can be trained for arbitrary tasks and using a whole range of objectives. Approaches using neural networks have yielded the state-of-the-art performance for several computer vision tasks, learning to detect features that are important for a given objective. Convolutional Neural Networks (CNNs) are particularly suitable for vision tasks as they explicitly consider the spatial relationships in the input. CNNs date back to the late 1980's (LeCun et al. (1990), presented in 1989), but only since 2012 have they begun to outperform other computer vision methods (Krizhevsky et al., 2012; Szegedy et al., 2015). One such very popular model is the VGG network (Simonyan and Zisserman, 2015). VGG is a CNN classifier of 1000 object classes, which has been used in numerous works which have fine-tuned it for specific purposes.

CNNs are composed of several layers, where each layer learns to recognize features from the previous one. Shared weights, representing different patterns, improve the convergence by significantly reducing the number of parameters. CNNs recognize small patterns at each layer, detecting higher order, more complex patterns in higher layers. They are trained using back-propagation and may use a variety of loss measures specified on their outputs, parameters or additional intermediate representations. They generally require large datasets for training, therefore a training is usually performed on GPUs which speed up the execution several times, allowing to train larger networks and to obtain a higher accuracy. Although usual NNs provide little interpretation of the learned features, CNNs features (filters) can be visualized (Zeiler and Fergus, 2014).

Currently deep learning methods are being applied to even more complex tasks, often being trained for several goals with different objectives simultaneously. These include object detection in the form of bounding-boxes (e.g. Uijlings et al. 2013, Redmon et al. 2016), object instance segmentation (Hariharan et al., 2016) and semantic pixel labelling (e.g. Badrinarayanan et al. 2017) in the image frame. Other application domains include object tracking (Nam and Han, 2016), 3D shape segmentation (Kalogerakis et al., 2017), and 3D shape reconstruction using voxel representation for a single object from a single image (Wu et al., 2017b; Sun et al., 2018)

When a CNN detector is applied to an image as a sliding window, a consequent issue becomes how to perform the actual detection from heat-maps, as they are blurred. Non-Maximum Suppression (NMS) is a straightforward technique commonly applied to tackle this problem (see e.g. Neubeck and Van Gool, 2006). The input to this algorithm is a set of the proposed bounding boxes. The goal is to filter out the detections which capture only a fragment of a given object if there already exists another bounding box which better captured the whole object. It was shown (Hosang et al., 2016) that an NMS alternative, where the neural-network re-scores the detections, may also be learned. Currently state-of-the-art approaches for object detection, such as YOLO networks (You Only Look Once, Redmon et al. 2016, Redmon and Farhadi 2017) make use of a single network that divides the image into regions (Regions of Interest), each region predicts bounding-boxes anchored at it and the detection probability.

2.3 Generative Models for Image Reconstruction

Our VIG approach employs a renderer of a 3D scene, but in general it is not necessary to use a renderer to specify a generative model of images. Numerous works employ generative models that output an image without the 3D rendering step. Note these methods can be trained also on images rendered using 3D graphics, but instead of generating the underlying interpretable 3D scene, they predict only the image in a per-pixel manner, nowadays usually using a neural network, e.g. PixelCNN (Van den Oord et al., 2016). Although recently realistic high resolution image samples have been obtained by employing numerous types of Generative Adversarial Networks (GANs) such as BigGAN (Brock et al. 2019), these can only sample new images but not explain them. Such approaches can be seen as autoencoders, which encode an image (usually demonstrated on face and gray-scale chair data) into an uninterpetable low dimensional vector or array-like representation. Although there have been attempts to obtain GANs with disentangled representation (see e.g. InfoGAN, Chen et al. 2016), these were not very successful and produced images that are of relatively poor quality, see Figure 2.2 column B.

Much focus has hence been on Variational Auto Encoders (VAEs, Kingma and Welling 2014), employing extensions aiming to provide some notion of interpretability. These variations of VAEs allow to learn some explainable factors, such as object azimuth or width (only to some extent, and when varying a given factor, often other unrelated object properties are being changed). These methods are for instance: Deep Convolutional Inverse Graphics Network (DC-IGN, Kulkarni et al. 2015b); beta-VAE (Higgins et al., 2017); and FactorVAE (Kim and Mnih, 2018). Figure 2.2 shows two random examples of the generated images by each of these methods. For instance, DC-IGN method performs training in batches where only a single latent variable is being changed to allow learning a specific “disentangled” representation of an image. However,

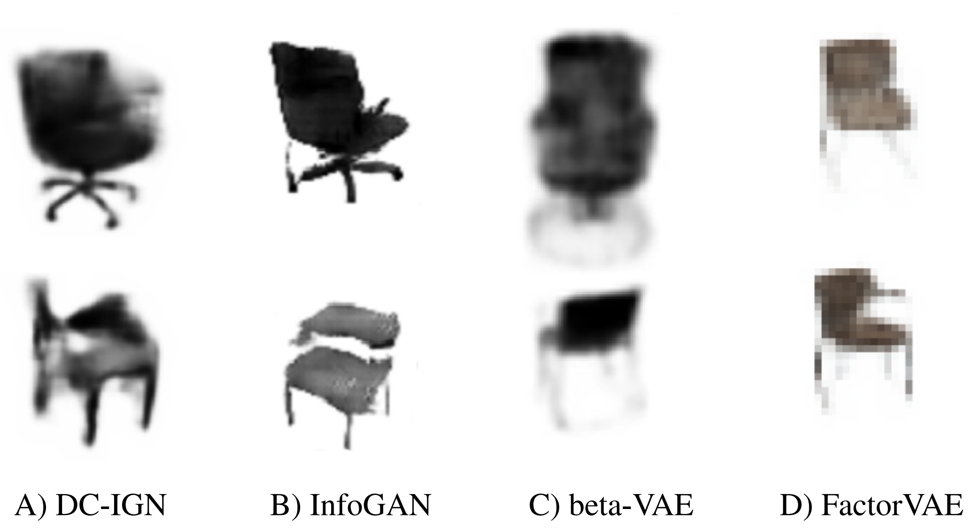


Figure 2.2: Each column shows two random example outputs for the chair dataset, from: A) DC-IGN (Kulkarni et al., 2015b), B) InfoGAN (Chen et al., 2016), C) beta-VAE (Higgins et al., 2017), D) FactorVAE (Kim and Mnih, 2018).

this method does not make use of a renderer to generate novel images. In consequence, their model can generate only a blurry object, the shape of which is approximated by a CNN un-pooling layers (see Figure 2.2, column A). To sum up, pixel-based methods can learn some fuzzy patterns of factors present in the aligned training datasets, but unfortunately all of them produce low resolution images and lack of an interpretable representation where one could edit the scene to e.g. shift an object.

There are some very recent works that consider the case of multiple objects: one example is a Generative Query Network (Eslami et al., 2018) that predicts an image from a queried viewpoint given a set of observations from known cameras. This method was trained on toy scenes with 3D primitives[1]. Again, such a representation does not allow for editing, such as shifting an object. The above approaches aim to model the distribution of pixel intensities of simple and low-resolution images of an aligned object, but do not generalize to actual images. There is also recent work that uses GANs to model appearance of an interpretable object-based representation of multiple cars, and predicts 3D models for these (Yao et al., 2018). However, the cars are only overlaid on the image, as the method does not model object locations or a camera pose in 3D space. In the next section we provide background on the approaches that reconstruct an underlying 3D representation.

[1]Note that for this method, to obtain an autoencoder, we would need to query for the same viewpoint as in the observed image.

2.4 Vision as Inverse Graphics

The fact that the latent variables of the generative model of real images are unknown has led much computer vision research to focus on discriminative recognition models as outlined above. These approaches belong to the class of bottom-up models, as they map an image to features suitable for a given task. An alternative, top-down approach to the recognition problem is to consider a 3D scene representation and the process of the projection of the scene onto an image. When a scene generator is defined in a form of a standard graphics renderer, the vision task requires inversion of the rendering process. Contrary to the popular bottom-up approaches, the top-down ones do not make use of features obtained from the given input image, but define an underlying generative process in order to explain the data. Figure 2.3 illustrates the generic bottom-up (recognition model) and top-down VIG approach (generative model).

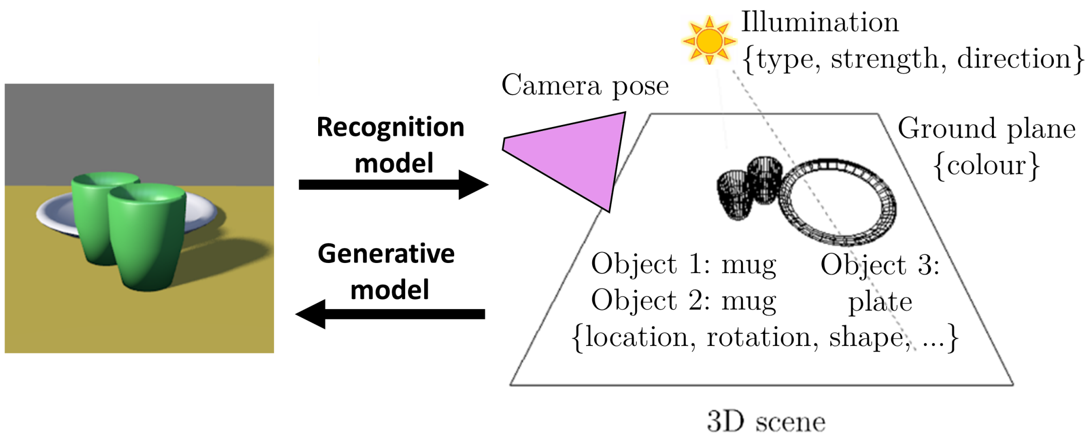


Figure 2.3: VIG paradigm: for the observed image (left) one aims to infer the 3D scene representation (right), bottom-up approaches make use of features to predict the representation via a recognition model, the top-down VIG approach defines a generative model in order to explain the input via its latent variables.

An important strand of VIG research uses RGB images plus the depth channel. For instance Sharp et al. (2015) apply a VIG approach successfully to human pose recognition from RGB+Depth images. Other very related work that also takes as input a depth channel is the work by Zou et al. (2019), applied to multi-object scene parsing. The authors describe a technique for 3D scene parsing closely related to the problem of 3D scene understanding that we investigate. They employ a recognition approach to recreate an indoor scene by composing furniture pieces and cuboid shapes to create a 3D representation (Figure 2.4). They search over CAD models and their poses whose shape best fits the depth point cloud. Their method uses 1,500 real images and their annotated scenes. The 3D annotations are hand-crafted approximations, making the method less scalable and less accurate. In contrast, our VIG approach does not require manual annotations as we can make use of a much larger amount of synthetic scenes sampled by a synthetic scene generator.

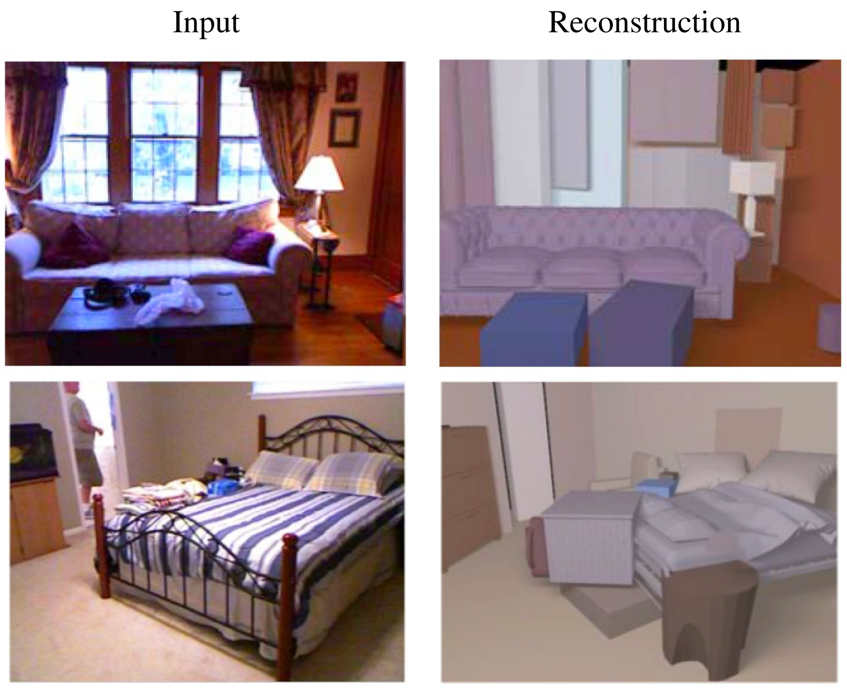


Figure 2.4: Example of an indoor scene and its reconstruction for observed RGB+Depth images (Zou et al., 2019), the top row shows a very good prediction, and the bottom row a failure case with superfluous object detections.

The lack of the depth channel makes the 3D reconstruction problem much more challenging, as then the image depth has to be reconstructed as well. While humans can evaluate the depth due to observing the scene with both eyes, at least to some extent, they are also capable of understanding scenes just from single RGB images. This is exactly the problem covered in this thesis.

The final three sections below provide background on VIG approaches. Section 2.4.1 describes methods that reconstruct (sometimes via optimization) a single underlying 3D object, Section 2.4.2 presents the most related methods that reconstruct a 3D scene containing multiple objects. Section 2.4.3 provides the summary of the discussed VIG approaches.

2.4.1 Single-Object Reconstruction in 3D

In this section we discuss VIG methods that output a single 3D object (usually as a mesh), not image pixels. These single-object works assume an object of a given class to be centred in the image. Single-object works are interesting on their own, and single-object fine-tuning methods could be used to improve explanation of each individual object in a multi-object scene. Once all the objects are instantiated in the scene, one can take image patches with the objects located at a fixed distance and view-point in the 3D scene, and then refine these. In general, VIG inference can be broken down into an initialization phase, and a subsequent refinement phase. The work of Williams et al. (1997a) is an early example of using neural networks for the initialization phase. More recent works make use of a deep recognition models for initialization, and later refine predictions using MCMC or gradient-based methods. Several works for this task are fitting a parametrized geometry to an underlying 3D object (Kulkarni et al., 2015a; Yildirim et al., 2015; Jampani et al., 2015). For instance, in the work by Kulkarni et al. (2015a) the authors chose to use the space of the features extracted by a CNN to perform the comparison of the observed and rendered image. This method could possibly be resistant to details like noise or texture, but it is not easy to interpret the activations of a hidden layer. There is very recent work on 6DOF pose refinement by Manhardt et al. (2018), which we call "6DPR", and the DeepIM method (Li et al., 2018b), also applied to the problem of object 6DOF pose estimation. These methods make use of a render of a considered object during the refinement, but this was demonstrated only for an object with a known identity, and known texture that was also applied in the renderer.

There are a few single-object works that handle a more interesting case, which in addition to the geometry and the pose, predict or optimize also the *appearance* (sometimes including the illumination), as outlined in Figure 2.5. Here and in subsequent figures with exemplary predictions, the observed input is given on the left, and the inferred output on the right. For instance Moreno et al. (2016) consider the VIG problem for an occluded teapot object on synthetic data. A different, non-mesh based approach is developed by Tulsiani et al. (2017), who make voxel-based predictions by learning by multi-view supervision via differentiable ray consistency of arbitrary shapes, where they define gradients of the voxelized representation. This is applied to model each object class separately, however the method is not able to e.g. reconstruct the chair wheels (see Figure 2.5B, bottom).

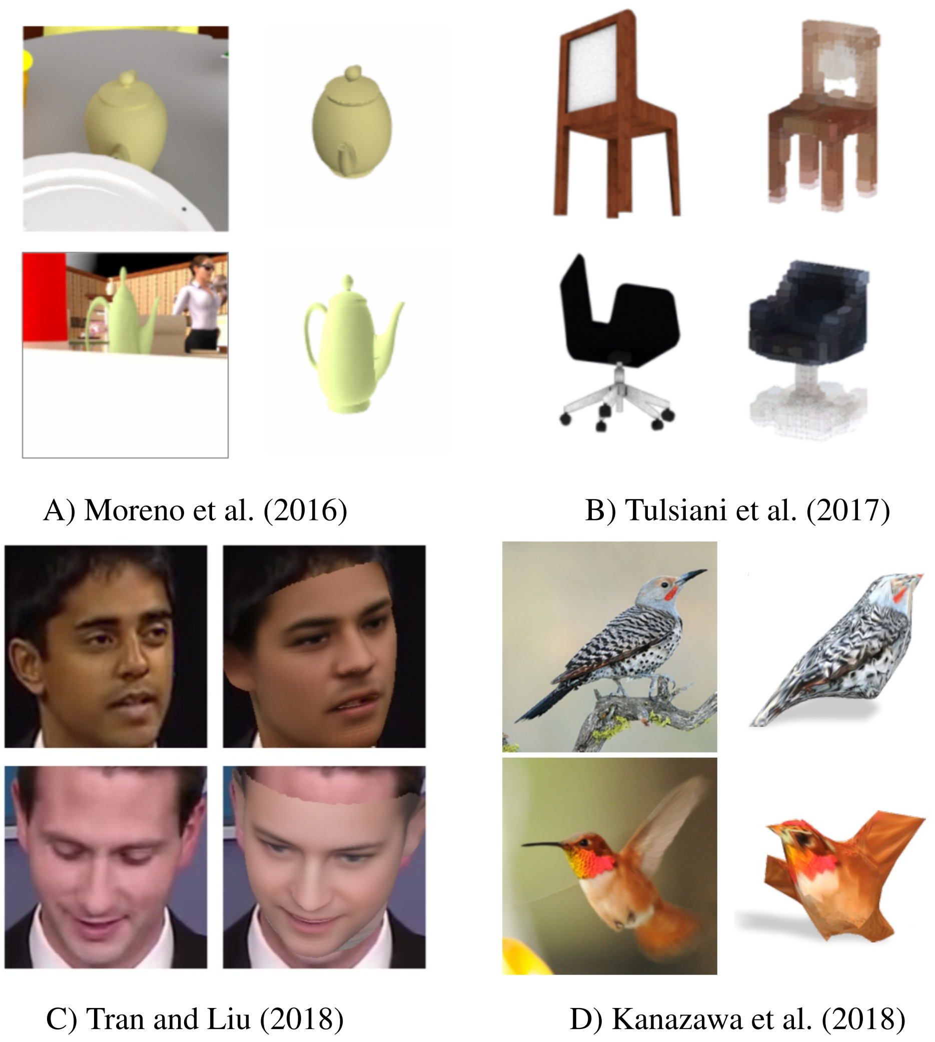


Figure 2.5: Examples of methods for single object reconstruction that jointly fit both shape and appearance, for synthetic images (top panels) and real images (bottom panels). The cases considered by the related works are as follows: A) synthetic data with occlusions and known shape model (Moreno et al., 2016), B) synthetic data and unknown shape model (Tulsiani et al., 2017), C) testing on real images, learning the shape model on synthetic data (Tran and Liu, 2018), D) training on real images, and learning the shape model (Kanazawa et al., 2018).

For real images, the majority of the works use face data, since there are several datasets of faces that are centred in the image (e.g. Liu et al. 2015 (*CelebA*), Huang et al. 2008), and there are publicly available 3D morphable models of face shape and appearance (Blanz and Vetter, 2003). Recent works not only fit a parametrized model, but also learn the shape and appearance model. For instance Tran and Liu (2018) and Genova et al. (2018) train autoencoders for this task. A novel approach by Kanazawa et al.

(2018) jointly models the shape, pose and texture for a single object and is trained only on real images. This method is evaluated on the bird class dataset, see Figure 2.5D for examples. The object is represented as a deformed sphere, plus a texture field mapped onto it. The texture field maps a mesh surface to the corresponding observed pixels, and the whole reconstruction is trained to maintain the projection consistency, including pixel colours, object mask and the keypoints. This work is interesting as it uses only real images for training, using masks and keypoint annotations to help learn the shape model, without supervision on synthetic data.

2.4.2 Multi-Object Reconstruction in 3D

In this section we discuss related work that considers multi-object scenes. This case is much more challenging, as one needs to infer object locations and poses, and deal with the ambiguities arising from unknown object sizes and the camera pose. Hence, the related work on multi-object scenes often handles only simplified scenarios. It is crucial to represent the presence of each whole object as a separate entity to deal with objects in a natural way, which is done by all the discussed related work below.

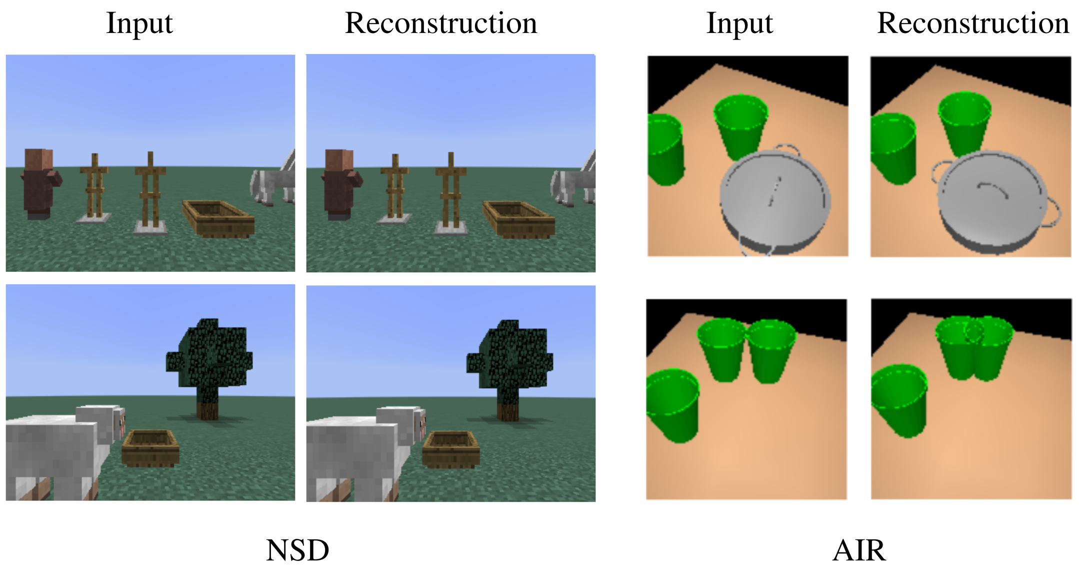


Figure 2.6: VIG frameworks that consider synthetic objects of fixed appearance. Left: example of images from the NSD paper (Wu et al., 2017a) and their reconstructions by the same renderer, right: example of images from the AIR paper (Esalmi et al., 2016) and their reconstructions by the same renderer.

Our framework is closely related to Neural Scene De-rendering (NSD, Wu et al. 2017a), who also define the vision problem as 3D scene graph inference. However, their approach was demonstrated to work only on synthetic images: the authors consider scenes comprised of either 2D sprite type objects, or 3D objects from the Minecraft game with 12 object types with fixed shapes and appearances. Figure 2.6 (left) shows example images and predictions from the NSD paper. However, note that their background scene (green grass and blue sky) is fixed, as are the camera parameters and the lighting. Also their predictions are made only for the cartoon scenes, not real images.

There is also an earlier work that considers multiple objects and a renderer within a VAE framework, the Attend, Infer, Repeat (AIR) network (Esalmi et al., 2016). One issue with the AIR network is that it uses a LSTM-based recurrent network to *sequentially* select an unordered *set* of object detections. Figure 2.6 (right) shows example images and predictions from the AIR paper. Although the visualized images are of a higher resolution, the network takes as input only 32$\times$32 images, which cannot represent even relatively simple real images. Most of the AIR work is on 2D images, but the authors do provide a demonstration of the AIR network on a simple “tabletop” scene, where different object types and the background have fixed and unique colours. This means the network does not need to learn to detect certain objects, but the problem is simplified to the detection of a given colour. Note since the camera is always at a fixed distance and elevation relative the ground plane, it is trivial to place the objects on the ground plane.

One of the key features in our VIG approach is an ability to transfer from synthetic to real images. The NSD and AIR works use objects of constant identity and appearance for both training and testing. Our work considers understanding of *real images*, thus we render high-quality anti-aliased realistic scenes with variable illumination, objects with different textures and cast shadows.

Our application of VIG to real images is very related to two works developed in the context of explaining real indoor images: IM2CAD by Izadinia et al. (2017) and the most recent work by Huang et al. (2018), Holistic 3D Scene Parsing (which we call “3DParsing”). These methods detect the objects, infer and optimize the room layout and object shapes and locations, and are evaluated in terms of the predicted 3D geometry, but not in terms of the reconstructed image.

The IM2CAD and 3DParsing methods recover the geometry for the purpose of room layout estimation, 3D room free space prediction and 3D object localization. IM2CAD

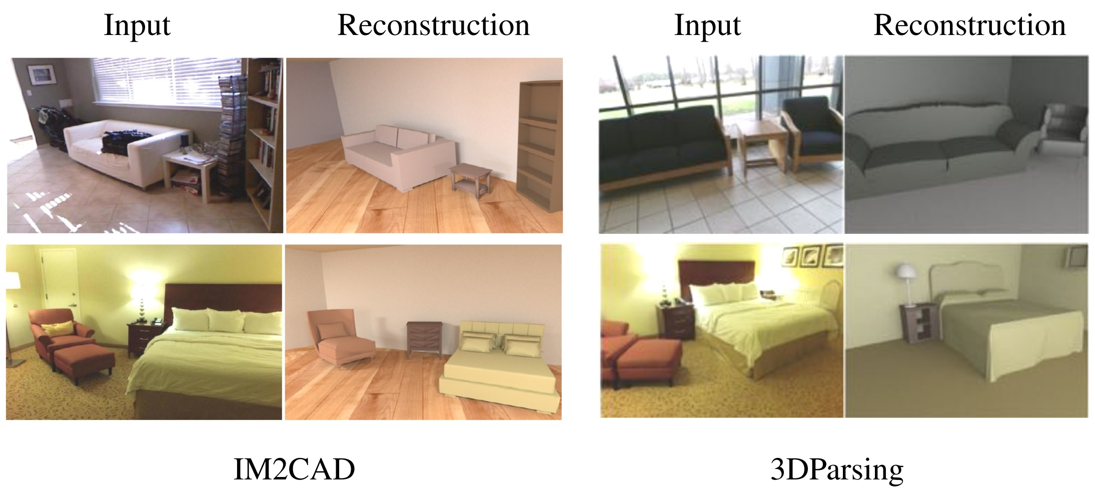


Figure 2.7: Related works that consider real images. Left: example recovered geometry outputs for IM2CAD; right: for 3DParsing. These methods reconstruct the 3D geometry of indoor scenes, but do not match the colours of images or model the illumination. For visualizations, objects are given a colour in a post-processing step.

uses a simple and not very effective refinement of the initialization by optimization of the poses of the predicted and observed shapes based on the VGG activations. The 3DParsing method is much more advanced overall. It adds extra regularization terms for geometry and functionality, and matches objects to the predicted surface normals, depth and semantic maps. The optimization involving all these terms is done by sampling, and 3DParsing obtains better results for all the metrics than IM2CAD.

The main difference between our work and the IM2CAD and 3DParsing methods is that these methods do not optimize the appearance or illumination parameters (for visualizations, objects are given a colour in a post-processing step). These works are designed for a good performance on the geometry metrics, but do not match images in the original RGB space. Obviously, for any method that outputs a 3D shape, whether single object or multi-object, one can set a colour in a post-processing step to match the pixels covered by the mask of the predicted object. For example, the colour applied can be the mean colour of these pixels. Note however that this does not include the further effects of illumination, the colour and illumination are not included in the optimization procedures, does not match the brightness, and is not robust when the objects are not well-aligned. For instance in Figure 2.7, in the top-right example, the predicted objects are shifted compared to the observed ones, and obtain grey colours as a result of covering a number of objects. In our framework we consider the colours and illumination as latent variables, and learn to predict and optimize these.

2.4.3 Summary of the Methods for 3D Reconstruction via VIG

Figure 2.8 presents an annotated version of Figure 1.2, highlighting the related work wrt. *real images*, *appearance learning or optimization*, and *multi-object scenes*. Our work deals with all the three aspects simultaneously, which is why our methods can serve as a base for development of VIG approaches for real images, and can be further extended.

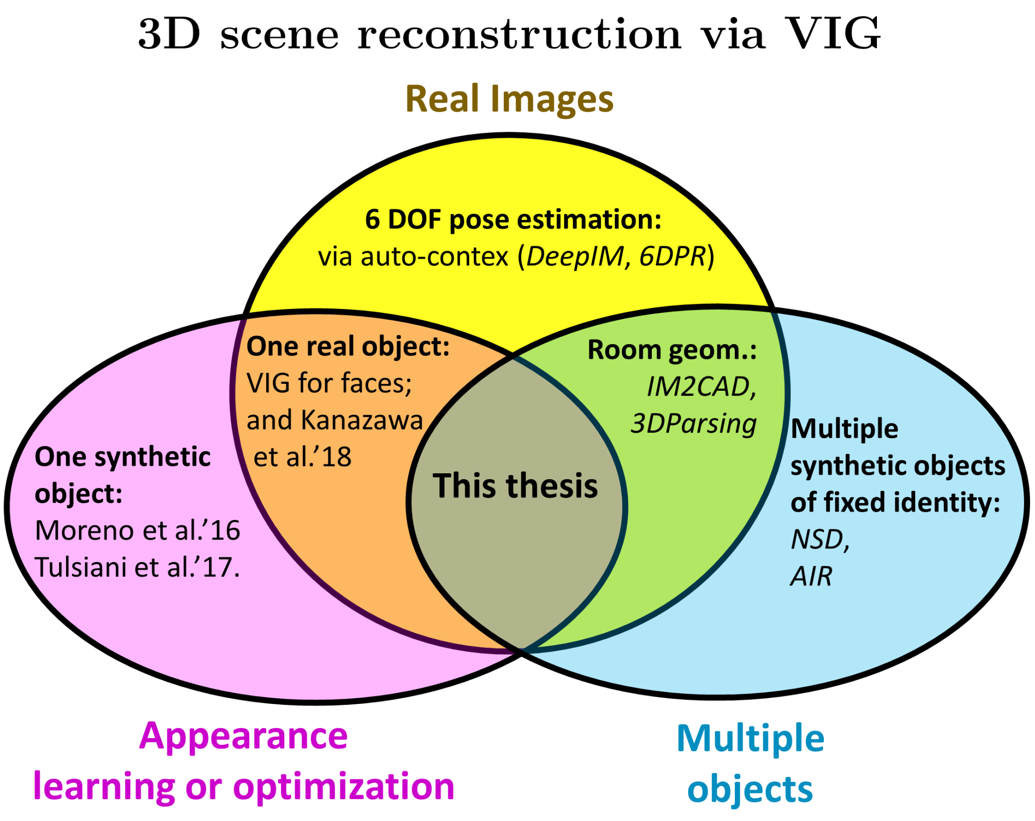

3D scene reconstruction via VIG
Real Images
6 DOF pose estimation: via auto-contex (DeepIM, 6DPR)
One real object: VIG for faces; and Kanazawa et al.'18
Room geom.: IM2CAD, 3DParsing
One synthetic object: Moreno et al.'16 Tulsiani et al.'17.
This thesis
Multiple synthetic objects of fixed identity: NSD, AIR
Appearance learning or optimization
Multiple objects

Figure 2.8: Classification of the discussed papers for 3D reconstruction into three groups of problems that they deal with. The works the closest to ours are the ones at the intersections of any two groups, either full VIG reconstruction for a single object (orange background) or multi-object room geometry reconstruction methods (green background). The remaining works handle simplified scenarios and only some aspects that we incorporate.

Chapter 3

Rich 3D Explanation of a Multi-Object Scene via Inverse Graphics

Our goal is the classic computer vision task of *scene understanding*, by which we mean obtaining a scene graph representation that includes descriptions of the objects in the scene (shape, appearance and pose) and their spatial layout, as well as global factors like the camera parameters and lighting. In this chapter we investigate how the recognition models can be used to infer the scene graph given only a single RGB image. This is to be contrasted with methods that simply predict 2D image-based bounding boxes or pixel labelling. Our models are trained using realistic synthetic images and corresponding ground truth scene graphs, obtained from a rich stochastic scene generator. We demonstrate the quality of the reconstructions obtained quantitatively on synthetic data, and qualitatively on real scenes.

3.1 Introduction

Our work is summarised in Figure 3.1. Object detectors (stage A) are run over the input image, producing a set of detections. We then predict the scene latent variables, consisting of the camera parameters (stage B), global parameters (stage D) and object descriptions (stage C), and back-project objects into the scene given the predicted camera (stage E). This acts like the encoder of an autoencoder, with graphics rendering as the decoder. Importantly the scene representation is *interpretable* and is of variable dimension to match the detected number of objects plus the global variables.

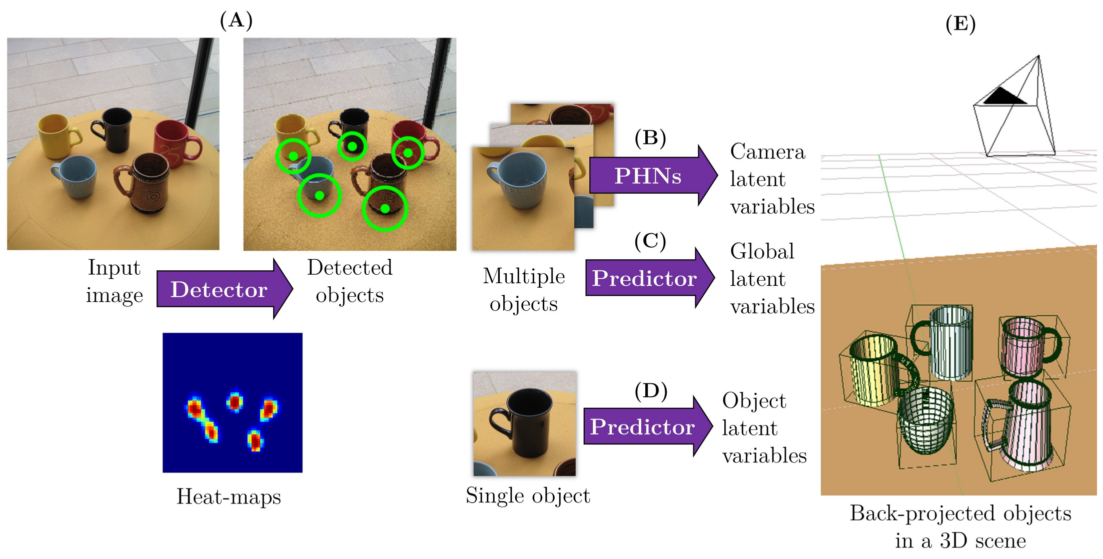


Figure 3.1: Overview: (A) Objects are detected in the image (green dots: contact points), which jointly predict the camera parameters (B) using PHNs. Then (C) other global parameters (e.g. lighting) and (D) object latent variables are predicted. These allow back-projection of the objects into the scene (E), and iterative refinement.

To solve the problem of inferring the camera parameters we introduce a novel Probabilistic HoughNets (PHNs) architecture, which carries out a principled integration of information from multiple object detections. The scene latent variables can then be rendered by a graphics engine to produce the predicted image, and by optimizing them the match to the input image can be refined iteratively.

We develop accurate recognition models trained on latent variables of realistic synthetic images in a way that they transfer to work with real images. We demonstrate the quality of the reconstructions obtained quantitatively on synthetic data, and qualitatively on real scenes.

In Section 3.2 we describe the scene latent variables and the Stochastic Scene Generator (SSG). We use our Stochastic Scene Generator to generate the datasets for the experiments. We then present our approach in Section 3.3, including the image formation mechanism from a 3D scene to a 2D image, as our goal is to invert this process. We describe all the framework stages as outlined above, to produce the final 3D scene representation. Section 3.4 presents the details of the experiments, Section 3.5 provides the results, and Section 3.6 provides the discussion.

In Chapter 4 we introduce Probabilistic HoughNets architecture and its use for principled integration of information both for uni-modal and multi-modal problems. We use PHNs for estimating the camera parameters within the framework as per label B in Figure 3.1.

3.2 Stochastic Scene Generator

3.2.1 Overview

The main task of our research is recognition and inference on multiple objects. We consider scenes with various items located on a ground plane. This allows us to focus on the core of the problem, i.e. the explanation of multiple objects present in the scene.

We develop a stochastic scene generator, which samples 3D scenes in a form of the scene graph and renders them to produce images. The fact that all the latent variables are specified means that it is known what objects are present in the scene and what their poses are. Images with labels serve as the dataset for training of the recognition model. As we are interested in creating a generative model, we can employ the VIG paradigm, because then all the latent variables are directly available. Importantly, this is also the case for pixel labelling of any kind (such as segmentation or depth masks).

The scene latent variables (scene graph) are denoted by $\mathbf{z} = \{\mathbf{z}^{global}, \mathbf{z}^{object}\}$. The two subsets of scene latent variables are as follows:

- global variables $\mathbf{z}^{global} = \{\mathbf{z}^{cam}, \mathbf{z}^{ill}, \mathbf{z}^{col}\}$: the camera latent variables, $\mathbf{z}^{cam}$, the illumination $\mathbf{z}^{ill}$ and the ground plane colour $\mathbf{z}^{col}$.
- the set of latent variables of each of O objects, $\mathbf{z}^{object} = \{\mathbf{z}_o^{object}\}$, $o = 1 \dots O$.

3.2.2 Scene Graph: Global Latent Variables

3.2.2.1 Camera

The camera model has both intrinsic and extrinsic parameters. The extrinsics are the translation and rotation of the camera; we assume that the objects lie on the (x, z) plane, and that the camera is at height $y = h$ above the origin. This is valid as we wish to estimate object poses *relative* to the camera. The camera rotation is as shown in Figure 3.2, with the camera looking at the ground plane at angle of elevation α.

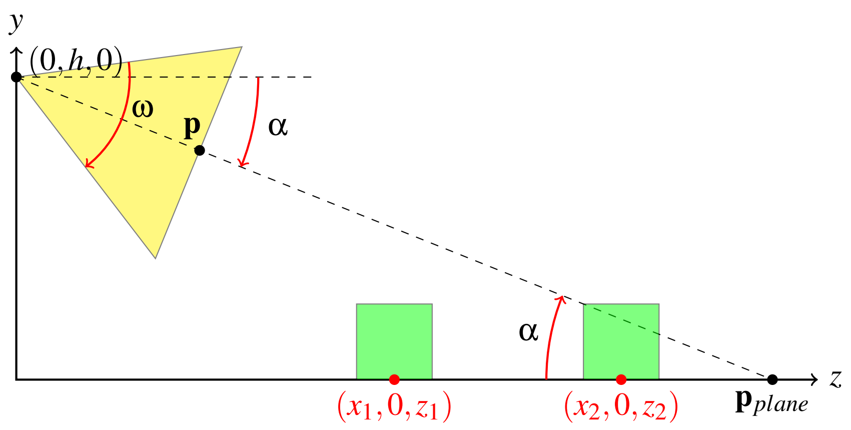


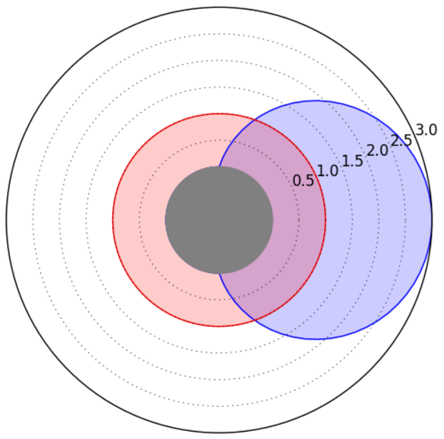
Figure 3.2: Scene side-view diagram depicting camera latent variables: (α, h, ω), two example objects, and camera principal point $\mathbf{p}$ projected onto the plane $\mathbf{p}_{plane}$

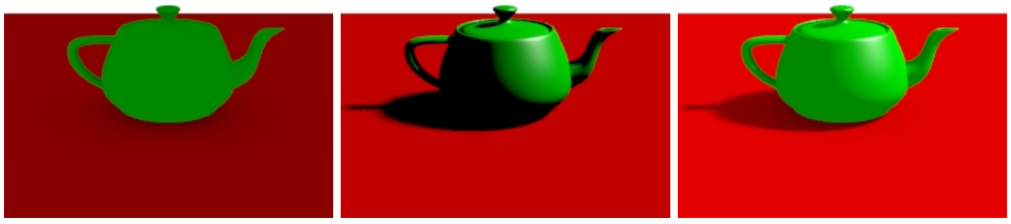
We assume a standard pin-hole camera model with no in-plane rotation. The camera sensor is a square of a standard size with a side a_0 of 35 mm; this is valid for any rectangular input as in such a case some areas at the border are inactive (truncated). The intrinsic parameter to be determined is the focal length f, or equivalently the angle of view (AoV) ω, which are related by $f = a_0/(2\tan\omega/2)$. This is also equivalent to inferring the zoom given a constant focal length. Thus $\mathbf{z}^{cam} = (\alpha, h, \omega)$.

3.2.2.2 Illumination

Another set of latent variables is responsible for the illumination parametrization. The illumination is represented as a uniform light around the whole scene plus a single directional light source, which we call Uniform-Directional (UD) representation. The directional light rays are parallel to each other, as for a distant light source. Figure 3.3 shows scenes with different UD light set-ups. The UD representation is able to cover a wide range of real illuminations, such as light coming from a lamp located on a tabletop or any mixture of uniform illumination coming from an overcast sky plus the directional one from the Sun. The UD representation $\mathbf{z}^{ill}$ consists of four latent variables: the strength of the uniform light, the strength of the directional light, and the azimuth within range $[0°, 360°]$ and elevation within range $[0°, 90°]$ of the rotation of the directional light.

One may note that a general scheme for representation of a function on a sphere, such as illumination environment, are Spherical Harmonics (SH, Ramamoorthi, 2006). SH are a series of orthogonal functions that are used to represent a function defined on





a) b) c)

Figure 3.3: UD light. **Top**: a polar plot of the amount of light received from the uniform component (red) and directional component (blue). The lights are around a sphere and are at the maximal strengths used in the experiments. **Bottom**: example scene consisting of a red ground-plane and a green teapot, with the mean strength of colours and the mean strength of illumination: a) only Uniform light, b) only Directional light from the side and at 45 degrees elevation angle, c) both lights together.

the surface of a sphere, in our case the illumination strength around the scene. Given enough coefficients, they can approximate any function arbitrarily well. However, although SH are suitable for illumination representation once it is known, it has several drawbacks when it comes to the inference.

First, the representation is very complex, as each but first degree of freedom of SH effectively controls multiple light sources. Although when observing the whole function there is only one solution, several different combinations of SH coefficients may lead to the same illumination when observing a scene from a single viewpoint, hence the problem becomes ill-posed. Further difficulties arise as observed shading/shadows in real images are evidence for the presence of positive and additive illumination, while SH can combine multiple negative light components, thus allow light subtraction. Therefore, CNN recognition models are unable to accurately predict the coefficient given the image (cf. Moreno et al., 2016). On the other hand, the UD representation is disentangled as the direction and strength of the shading and the shadow in the image approximately determine the direction and proportion of directional light. Moreover, the UD representation admits a straightforward sampling scheme, while for SH it is not obvious which combinations of SH coefficients may likely appear in real images, as these could not contain negative light sources.

Finally, UD illumination allows for straightforward manipulation of the configuration in order to refine the prediction, e.g. the direction of light is parametrized using the azimuthal and elevation angles. This is not the case for SH, since a change in rotation leads to a different configuration of SH coefficients responsible for different sphere areas. The coefficients configuration has to change in such a way that the ones that caused light in the area before the rotation would make the illumination strength lower, while causing the light strength to be higher at the light area at the final configuration.

3.2.2.3 Ground Plane

The remaining global latent variable is related to the colour of the ground plane. Thus $\mathbf{z}^{col}$ is an RGB representation of the ground plane colour.

3.2.3 Scene Graph: Object Latent Variables

Each rigid object $\mathbf{z}_o^{object}$ in a 3D scene is described by its class c_o, and its position $(x_o, 0, z_o)$; $y_o = 0$ as the object is lying on the scene plane. For reference, Figure 3.2 presents the orientation of the axes and two objects in a 3D scene. The other variables are scaling factor s_o (where the scaling factor $s_o = 1$ is the mean scale of the object class) and azimuth angle ϕ_o (rotation around the vertical y-axis). The remaining latent variables are class-specific ones which are responsible for appearance and shape descriptions, $\mathbf{a}_o$ and $\mathbf{h}_o$. Summing up, the object is represented as $\mathbf{z}_o^{object} = (c_o, x_o, z_o, s_o, \phi_o, \mathbf{a}_o, \mathbf{h}_o)$.

A key assumption behind our inference approach is that all objects are placed on the surface of a plane e.g. a mug on a tabletop. Hence, the key question is how we place a 3D object at the desired position $(x_o, 0, z_o)$ on the ground plane. To this end we specify the central contact point of an object, and then learn to detect it. This enforces that an object located lower in the image frame is located closer to the camera in the 3D scene. A contact point is defined as the point on the plane below the centre of the main part of a given object class. From the top view it is the point around which the object's azimuthal rotation is defined, which allows the object position to be azimuthal rotation invariant. For example, for a mug class it will be the point on the plane below the centre of the mug body.

3.2.4 Sampling and Rendering Procedure

Even though the recognition models are trained on synthetic images, we keep in sight the main goal, which is to facilitate understanding of real images. Therefore, the scene generator is able to generate plausible scenes and render very realistic, anti-aliased images with noisy textures and cast shadows, similar in their structure and appearance to the real ones.

For each image we first sample the global parameters and a number of objects which lie on the ground plane. These are rendered using Blender at 256×256 resolution. To sample a scene we first select a target number of objects. We then sample the camera parameters and the plane colour. Objects are added sequentially to the scene, and a new object is accepted if at least a half of it is present in the image, it does not intersect other objects, and is not occluded by more than 50%. If is is not possible to place the target number of objects in the scene (e.g. when a camera is pointing downwards from a low height) we reject the scene.

We impose a prior on the scale variable, $p(s_o)$, reflecting our class-specific knowledge about a realistic scale of objects of a given class, and sample the scale according to it. Without knowing object prior scales, an inferred scene configuration would be in unknown units, since the whole scene could be scaled without a change of the image.

For each object we sample its size, shape, colour and rotation, and also a random texture. The textures are applied to introduce noise which is present in real images (such as labels or text), otherwise the trained networks could deal only with untextured objects. A collection of textures is converted to grey-scale and then the object colour is multiplied by a random texture. The associated ground-truth colour is the mean colour of the resulting pattern. This way we obtain various noisy patterns in our images. This dataset obtained from the Stochastic Scene Generator serves for training, validation and testing purposes.

3.3 Approach

We train our models to predict the scene latent variables, i.e. we do not aim to simply minimize the reconstruction error in terms of pixel colours of the observed and predicted image. Since similar images may have different interpretations, we do not want to predict a scene where objects look similar to the ones in the observed image but are for instance of different classes (e.g. a cup vs. a mug). The same is true for other latent variables, such as the object azimuth: an identical mug to the ground-truth one rotated by 180 degrees will still match almost all the pixels, with minor errors where the handles differ. However, the predicted azimuth latent variable will be incorrect, and a wrong initialization would make the fine-tuning infeasible. It was shown by Williams et al. (1997b) that a recognition model can help to efficiently find a solution for inverse graphics problem, therefore our first goal is to perform accurate inference of scene latent variables without an additional optimization procedure at the test time.

We explain all objects (their pose, size, shape etc. in 3D units) using a single camera and common illumination. For a single image with a single object there exist multiple likely solutions to the scene configuration problem. An object can be a large instance of its class, or a smaller one with the camera located either closer, or with the camera located farther but using a larger zoom (focal length). The more objects are located in the scene, the posterior over the camera and other global scene latent variables becomes less ambiguous. Analogously, each object brings in additional evidence for a particular illumination. We allow the objects and their sizes to explain the camera, contrary to the approaches that make use of vanishing lines or Manhattan-world assumptions (Bazin et al., 2012; Lee and Yoon, 2015), as they are not generally applicable (e.g. boats on the sea). We also do not use the notion of the horizon which may be outside of the image frame.

Recall that $\mathbf{z}^{object} = \{\mathbf{z}_o^{object}\}$, $o = 1 \dots O$, and $\mathbf{z}_o^{object} = (c_o, x_o, z_o, s_o, \phi_o, \mathbf{a}_o, \mathbf{h}_o)$. The set of object latent variables $\mathbf{z}^{object}$ is the representation of all the objects in the 3D scene. However, as we process the input in a form of a 2D image, there are latent variables defined within the image frame and depending on the view-point: the position in the image frame x_o^{view}, y_o^{view} and projection scale of the object s_o^{view}. The position in the image frame is related to the pixel position (row/column), where $(0,0)$ position is the centre of the image, and image is of a unit size and is defined for $[-0.5, 0.5] \times [-0.5, 0.5]$.

Since the distance of the object is unknown, we can directly predict only the scale of the projection s_o^{view} (Szeliski, 2010, page 57), see also Equation 4.12. As s_o^{view} is the scale of the projection, for a fixed viewpoint and object size it is related to the number of pixels that an object covers, so it can be easily predicted from the image window (image window). Given the camera, we can later back-project these variables into the 3D scene to obtain x_o, z_o, s_o.

Typically, the size of an object is captured using bounding boxes, and prediction of bounding boxes is the ultimate goal of many standard computer vision tasks. However, bounding box dimensions depend on the view, which makes using them for the object scale inference much more complicated. The reason is that we would then need to somehow fit the object to the bounding-box, but the silhouette depends on the viewpoint, shape and pose of the object. Therefore, we instead predict the projection scale s_o^{view}, which is directly related to s_o through the mean size of the given object class. For instance the bounding-box of a mug varies when we rotate the mug or change the elevation of the view-point. By using the notion of the projection scale, we directly predict the representation we need to incorporate. Here we directly map from the image window to the object instantiation, and avoid using additional constructs such as the object bounding-box.

To sum up, the object detection is represented by a class c_o, the contact point position (x_o^{view}, y_o^{view}), plus the projection scale s_o^{view}. Thus the view-dependent latent variables and the object class are predicted by the object detector, while the remaining ones, specific to each class $(\phi_o, \mathbf{a}_o, \mathbf{h}_o)$ are directly predicted by the object predictor conditioned on the object detection.

3.3.1 Detector

We propose our own original object detector. The main reasons for not using any already existing one are: (i) typically the detections are blurry as detectors activate at any part of the object of a given class, while we aim to detect object contact points and (ii) classifiers do not predict any of the variables of interest to us. Nonetheless, we do use all the convolutional layers from the VGG-16 network to substantially improve the accuracy on real data. We train our detector to predict specific outputs by training only the fully connected layers on top of all 13 convolutional VGG-16 layers. The parameters of VGG network are kept fixed, this significantly decreases the number of trainable parameters and prevents overfitting. Such an approach has been shown to perform with a great success in Kar et al. (2015) and Bansal et al. (2016).

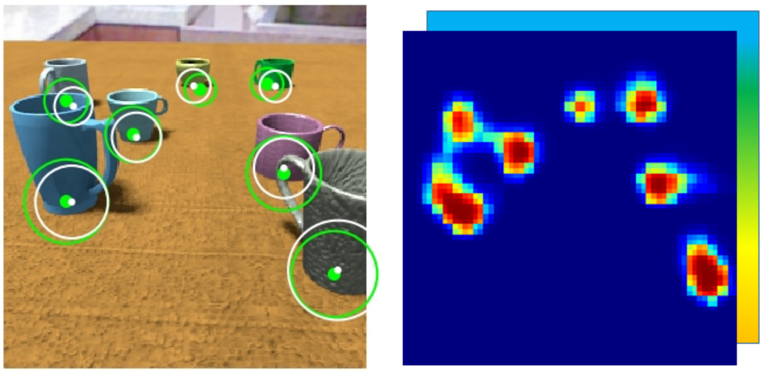
Thus the detector is used to produce object detections in the image frame. The set of possible classes includes a class representing the background is included. It was crucial to prevent the detector from activating for other object classes or noisy backgrounds that are present in real images, therefore a half of the training dataset consists of random negative windows from real images. The background class is predicted in a case when there is no object, or the object is actually present but its contact point is greater than some distance from the centre. In this way we force the detector to have the highest confidence when the object is centred, as we need to know the object contact point to perform back-projection.

Note that the x^{view}, y^{view} variables are not predicted, but are defined by the location of the detector application. The detector is trained in a supervised manner on positive and negative image windows, and there are two neural networks in total, i.e. class c_o (one-hot-encoded) for all the image windows, and projection scale s_o^{view} (single value) for the positive image windows.

A CNN sliding window detector produces predictions in the form of two heat-maps with entries representing the detection class and projection scale. Heat-maps are a suitable representation as they can capture a variable number of objects. A potential drawback of the sliding window approach is its slow execution. However, the standard practice is to overcome this inefficiency by performing the whole computation of the sliding window algorithm at once. This is possible for our CNNs, as the feature maps in all convolutional layers can be computed at once, thus the execution time of a sliding window evaluation is similar to the application of a CNN to a single input image.

We use the above approach to produce candidate detections in the form of two heat-maps (see Figure 3.4b). The output probability map of the detector is thresholded, and finally, the detections are sparsified using a method based on non-maximum suppression (NMS, Neubeck and Van Gool, 2006). Then, the value of the projection scale at the pixel location of the detected contact point is used as the predicted projection scale for the given detection.

We generate bounding-boxes as squares of the predicted size, and extend the NMS to use the notion of object contact point, i.e. we disregard the detections if there is a higher scoring contact point within the bounding-box area. This allows us to detect more objects located further away or in cluttered scenes. Note the positions of detected contact points and predicted sizes are very accurate. Objects located far away, truncated, or partly-occluded are usually detected properly.



(a) detections (b) heatmaps (class, scale)

Figure 3.4: The detected objects (a) and example heat-maps (b, detection confidence and projection scale). The green circles in (a) represent the position of the detected contact points and mug diameters, the diameters are proportional to the predicted projection scale. The white circles are the ground truth. The detector peaks are at the contact points of the mugs.

Once the object detector network is trained, we obtain a derived dataset of the true positive detected image windows, which incorporates errors made during the detection step. For each detection the closest object within a fixed radius is assigned, along with the corresponding object latent variables that are known for each of the assigned object. We use 24 pixel radius for the derived dataset of the true positive detections, note this is much larger distance than the average error of the detector, which is 5.4 pixels. The average size of an object's bounding box in the Mugs dataset is 42×45 pixels, so this is about half the spatial size of the object. We then use this dataset for training the object and global predictor models.

3.3.2 Predictor Networks and Scene Graph

The global and object predictor networks predict the scene latent variables. The networks are applied individually to each image windows with the detected object to predict the global latent variables and the remaining object latent variables, $(\phi_o, \mathbf{a}_o, \mathbf{h}_o)$. These networks also take as input the position and projection scale of the object

$(x_o^{view}, y_o^{view}, s_o^{view})$. We then aggregate the estimates so as they vote together for the most likely scene configuration, as outlined below. The details of the networks are given in Section 3.4.2.

Illumination is predicted by making lighting predictions for each detected object, and then combining these by the taking median in each dimension (latent variable) across the detections. This works much better than providing whole images as input; we believe that providing the detections allows cues from shadows and shading to be used more effectively. The ground plane colour is predicted in an analogous manner.

The instantiation of object poses in the 3D scene is very sensitive to any changes in the camera parameters, and larger errors could prevent the gradient-based refinement (see Section 3.3.4) to converge properly. Therefore we developed the Probabilistic HoughNets method (see Chapter 4) to infer the camera LVs with a high accuracy, even though it was more computationally expensive. The probabilistic camera predictions made by each object are combined using PHNs, then the detected objects are back-projected into a 3D scene given the predicted camera to obtain the 3D positions of objects. The next section provides the details of the back-projection process.

The outputs of the above stages are assembled into a scene graph, in the form as outlined in Figure 1.1. As the scenes we study contain only certain types of latent variables, the output of our analysis can be expressed in terms of a domain specific scene graph language. Using the object instantiations, the plane appearance and the illumination latent variables we can then render the scene as observed by the predicted camera.

3.3.3 Image Formation and Back-Projection

A 2D image is a projection of a 3D scene generated by Stochastic Scene Generator (SSG), which leads to the task where the inferred position of the object in the image frame has to be mapped to the position in the 3D scene frame. Therefore, we need to perform a back-projection of the detected objects from the image frame onto the plane. This is possible due to the assumption that objects are placed on the plane, otherwise objects could be hanging in the air and their depth would be ambiguous. The problem of mapping objects from an image to 3D was studied e.g. by Hoiem et al. (2008), however under a simplifying assumption that the camera view was horizontal. We relax this restriction and treat the camera elevation angle as an unknown latent variable. We assume a unit-aspect, zero-skew pin-hole perspective camera with no in-plane rotation. For a given camera the mapping from the image frame to the 3D scene plane is known, but the camera parameters have to be inferred.

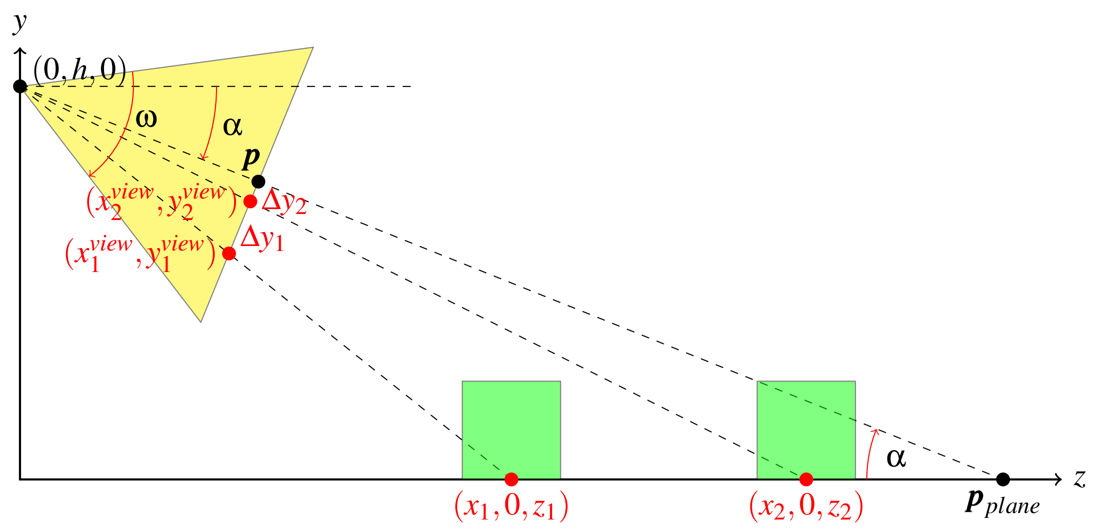


Figure 3.5: Two objects and projections of their contact points in the image frame: (x_1^{view}, y_1^{view}), (x_2^{view}, y_2^{view}). Δy_1 and Δy_2 are the y-axis deviations of the observed object contact points from the centre of the image.

The perspective projection formula gives us the relationship between the world (scene) and the image coordinates through the camera extrinsic and intrinsic parameters. The centre of the image (the principal point position) is located at $(p_x, p_y) = (0,0)$. This way, we can relate the detections of the object at the point (x_o^{view}, y_o^{view}) in the image frame to the camera variables and world position $(x_o, 0, z_o)$ of the object ($y_o = 0$ as objects are on the plane). Figure 3.5 shows how the scene is projected onto the image. The projection (using homogeneous coordinates) is given by the following formula (Szeliski, 2010, page 51):

$$\begin{bmatrix} x_o^{view} \\ y_o^{view} \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{R}(-\alpha) & \mathbf{R}(-\alpha)\mathbf{t}(-h) \end{bmatrix} \begin{bmatrix} x_o \\ 0 \\ z_o \\ 1 \end{bmatrix}, \tag{3.1}$$

where $\mathbf{K}$ represents the calibration matrix depending on the camera intrinsic parameters. $\mathbf{R}(\cdot)$ is a 3×3 rotation matrix of the 3D scene for the camera elevation angle α, $\mathbf{t}(\cdot)$ is a 3×1 translation vector along the y-axis, as follows:

$$\mathbf{R}(-\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(-\alpha) & \sin(-\alpha) \\ 0 & -\sin(-\alpha) & \cos(-\alpha) \end{bmatrix}; \qquad \mathbf{t}(-h) = \begin{bmatrix} 0 \\ -h \\ 0 \end{bmatrix}. \tag{3.2}$$

The affine transformations are negative as in the actual projection formula we transform the 3D scene instead of the camera pose. **K** has the following form (zero entries omitted for clarity) :

$$\mathbf{K} = \begin{bmatrix} m_1 & & \\ & m_2 & \\ & & 1 \end{bmatrix} \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix} = \begin{bmatrix} mf & & \\ & mf & \\ & & 1 \end{bmatrix}, \tag{3.3}$$

where we assume pixel magnification factors denoted $m_1 = m_2 = m$ are equal in each direction, and the principal point position $(p_x, p_y) = \mathbf{0}$ as it lies in the centre of an image. The first matrix in (3.3) is simply a scaling transformation. Note that the input images still come in arbitrary scales. The second matrix scales the image by f. Thus, to sum up, the matrix **K** simply scales the image by a value proportional to f.

Once we infer all the camera latent variables, we can calculate the contact point position on the ground plane $(x_0, 0, z_0)$ given the detected contact point in the image frame (x_o^{view}, x_o^{view}) and back-project the object.

3.3.4 Iterative Refinement

As the predictions are in the form of a scene graph, we can refine the fit iteratively using the generative model (graphics renderer) to further enhance the predicted scene.

The likelihood of the scene graph is based on the similarity over pixel intensities of the observed and the generated image. Observed image $\mathbf{I}^O$ and the rendered image $\mathbf{I}^R$ given the scene graph $\boldsymbol{\theta}$ have P pixels and are represented as a vector of a length $3P$, with P values for each RGB colour channel.

We compute the match between the actual and rendered images using a robustified Gaussian likelihood model as in Moreno et al. (2016), i.e. we assume that pixel intensities come from a mixture of a Gaussian and a uniform distribution, where α is the mixing proportion:

$$p(\mathbf{I}_i^O|\boldsymbol{\theta}) = \alpha\mathcal{N}(\mathbf{I}_i^O; \mathbf{I}_i^R(\boldsymbol{\theta}), \sigma^2) + (1-\alpha)\,\mathcal{U}(\mathbf{I}_i^O). \tag{3.4}$$

The mean of the Gaussian component is equal to the predicted pixel intensity of a given colour channel. The uniform component is introduced to deal with outliers, i.e. noisy pixels that were not explained properly, for instance due to occlusion, a texture, or inability of the generative model to represent each object shape or appearance detail exactly.

To allow to perform the iterative optimization we use an enhanced differentiable renderer[1] based on OpenDR: Differentiable Renderer (Loper and Black, 2014). The basic version is extended to simplify rendering multiple textured objects and to use modern OpenGL functionality (e.g. shaders). The speed of differentiation is of a great importance for us, as we have several latent variables per object, approximately 50 on average for the whole scene. The main advantage of the differentiable renderer is a faster gradient computation than when using finite differences.

In the original OpenDR implementation the derivatives of the pixel intensities with respect to vertex translation are approximated using the Sobel Filters. To more accurately capture the gradients at the object edges, as the gradients are the largest for the pixels where multiple objects interact, we perform anti-aliasing by calculating 8 samples per pixel. The derivatives of the likelihood computed by OpenDR are fed to a nonlinear conjugate gradient optimizer[2].

3.4 Experimental Set-Up

This section provides the experimental details for the presented approach. In the first part (Section 3.4.1) we present the set-up of the Stochastic Scene Generator which was used for generating the datasets. The second part (Section 3.4.2) describes the details of the CNN architectures.

The questions to be assessed are as follows:

- Can we can successfully reconstruct a 3D scene via VIG from a single RGB image (for a given set of object classes)?

[1]https://github.com/polmorenoc/inversegraphics
[2]http://learning.eng.cam.ac.uk/carl/code/minimize/

- What is the quality of the reconstructions on synthetic data?
- Do the models transfer to work well on real images?

3.4.1 Mugs Dataset: Experimental Set-up

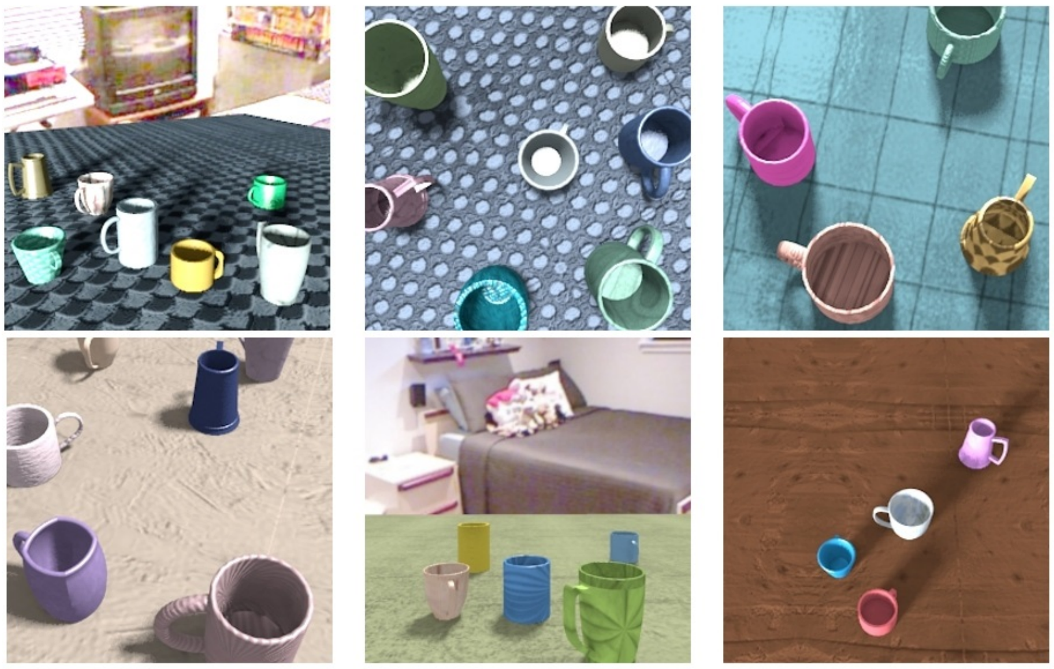


Figure 3.6: Example realistic synthetic images from the Mugs dataset.

We use our Stochastic Scene Generator to generate the training dataset of 7k images. They contain a total of 35k objects, with up to 7 objects per scene. The Background images are taken from the NYU Depth V2 dataset (Silberman et al., 2012). The collection of 200 textures with a wide range of pattern kinds was obtained from the Mayangs website [3]. Figure 3.6 presents a few examples of the generated scenes. The SSG is implemented in Python and makes use of the Blender API [4] to handle the scenes [5].

[3]www.mayangs.com

[4]www.blender.org

[5]The initial script for importing CAD models, generating scenes with multiple objects, and rendering synthetic images was provided by Pol Moreno. This has been thoroughly extended and improved: I collected and included randomized textures and backgrounds, made improvements in the specification of the scene and camera placement, fixed a major memory leak and made the data generation reproducible even if any rendering call fails. I also made various fixes in object collision code, code for creating Blender materials and illumination, and improved efficiency of the scene sampling. Later on, for Chapter 5, I collected, prepared and aligned new CAD models, for which several manual fixes such as reversing face normals or decreasing amount of vertices had to be performed. I also implemented handling multiple object classes within SSG, and specified an improved method for sampling the colours.

We sample the camera AoV and then height and elevation uniformly in the appropriate ranges: $\omega \in [20^\circ, 60^\circ]$, $\alpha \in [0^\circ, 90^\circ]$, $h \in [0, 150]$ cm. Illumination is represented as Uniform-Directional illumination, with the strength of the uniform light in the range $[0,1]$ and the strength of the directional light in the range $[0,3]$; the azimuth $\in [0^\circ, 360^\circ]$ and elevation $\in [0^\circ, 90^\circ]$ of the rotation of the directional light.

As per the dataset name, the objects are of the mug class, and we use 15 diverse mug shapes from ShapeNet[6]. The size prior is specified in terms of a mug diameter and is uniform within $[8.0, 10.4]$cm. The colours of the objects and the ground plane are sampled uniformly in the RGB space. Thus the object shape $\mathbf{h}_o$ is represented as 1-of-K CAD shapes, the appearance $\mathbf{a}_o$ is object colour represented as an RGB triple.

3.4.2 CNNs Experimental set-up

All the CNNs, as presented in Table 3.1, are based on the VGG-16 network of Simonyan and Zisserman (2015) except for those that predict colour, which are standard 3-layer CNNs. The reason for using simple CNNs for colour prediction is that VGG layers extract the shapes, but do not preserve the colours. The networks' hyper-parameters were optimized on a validation dataset. The VGG-based networks use all 13 convolutional layers of VGG for 128×128 input, but without the last two max-pooling layers in order to be more spatially accurate, resulting in an output of size $512 \times 16 \times 16$. On top of these, we train three convolutional layers with 50 filters, each of a filter size of 6×6. We have found this configuration to work best amongst different CNN architectures. This leads to an output of the third convolutional layer being of size $50 \times 1 \times 1$. We then use this representation of the length of 50 values as an input to fully connected layers. The output of each network is a specific set of latent variables, as outlined in Table 3.1. Since the predictor networks take the image windows as input, we also concatenate the input to the first fully connected layer, for each network, with $(x_o^{view}, y_o^{view}, s_o^{view})$ of an image window. The values in the dataset of the latent variables and those of the image window are standardized. We only train all the layers on top of the VGG ones, the VGG layers are kept fixed; this decreases significantly the number of trainable weights to approximately 1 million for the detector and 0.4 million for the rest of the VGG-based predictor networks.

We use *softmax* output for classification and *sigmoid* for regression as all our latent

[6]https://www.shapenet.org/

	Detector		Predictor		
Resolution (outgoing)	Class	Scale	Shape	Azimuth	Lighting
$128 \times 128 \times 3$	RGB Image				
$16 \times 16 \times 512$	VGG-16 (all 13 convolutional layers)				
$11 \times 11 \times 50/20$	C-50-6	C-50-6	C-50-6	C-20-6	C-20-6
$6 \times 6 \times 50/20$	C-50-6	C-50-6	C-50-6	C-20-6	C-20-6
$1 \times 1 \times 50/20$	C-50-6	C-50-6	C-50-6	C-20-6	C-20-6
	F-200	F-200	F-50	F-50	F-100
	Softmax-2	Sigm-1	Softmax-15	Sigm-2	Sigm-5

Learning rate				
Class	Scale	Shape	Azimuth	Lighting
0.001	0.0002	0.001	0.0001	0.0001

Table 3.1: The configurations of the five main CNNs and learning rates. Each CNN is 18 layers deep. Layer types are: C – Convolutional, F – Fully connected, Sigm – Sigmoid and Softmax. Convolutional layers are described as follows: C - number of output units - filter size, for the remaining layers the name is followed by the number of units (e.g. F-200 is a fully connected layer with 200 units/neurons).

variables are bounded. For rotations we predict the sine and cosine of the angle, hence two network outputs are used for each azimuthal rotation latent variable. Networks are trained by SGD with Adam optimization algorithm Kingma and Ba (2015). We found the *tanh* activation in all the layers on top of VGG to be superior to other activations for all the main recognition models – but note that all the VGG layers use rectified linear (*ReLU*) activations, hence the majority of layers in each CNN uses *ReLU*.

Table 3.1 shows the network configurations and learning rates used for training. We use all 13 convolutional layers of VGG-16 as the core but on 128×128 pixel input. The fully connected layers of the detector networks are implemented as filter 1×1 convolutional layers, so they can be efficiently applied in a sliding window manner. We use dropout in the convolutional layers on top of VGG ones: in the detector networks with $p = 0.5$ and in predictor networks with $p = 0.2$. The colour networks are 3-hidden-layers CNNs (128^2, C-27-6, F-40, F-40, Sigm-3), however here we use *leaky* *rectify* activations in the whole network and stride 6 in the first layer.

We performed a set of preliminary experiments that led to these hyper-parameters. The reason for using a higher dropout value in the detector than in the predictor is that the detector uses a larger dataset. The VGG output resolution ($N \times N$ of 512 channels) can be controlled by using less or more max-pooling layers, and we found 16×16 to work better for latent variable prediction than the original 7×7 used for the ImageNet classification task. The most important was the choice of the network architecture on top of the VGG, given the constraints of 16×16 input resolution and 1×1 output resolution. We have tested different configurations of the number of layers and the filter size, and eventually used 3 convolutional layers with filter size 6×6. We do not use any padding or stride in our convolutional layers.

3.5 Results

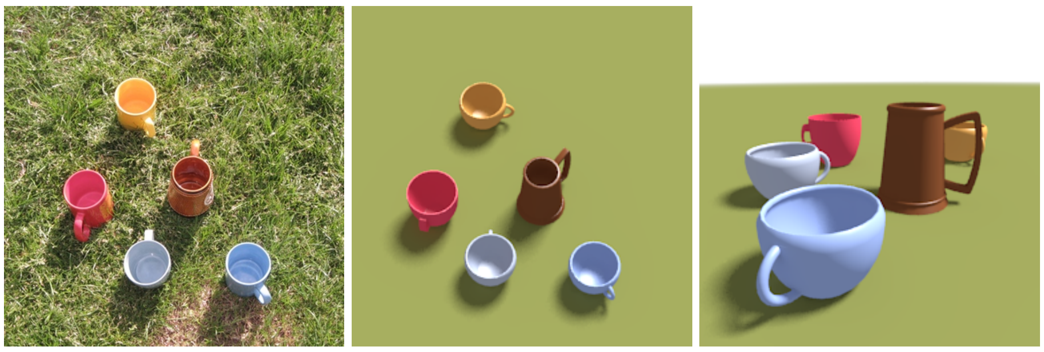


Figure 3.7: Input real image, reconstructed 3D scene and a different view of it. Due to the interpretable representation, one could easily edit the scene, e.g. change object positions or their colours.

We perform a quantitative evaluation of all the components on a synthetic test set of two hundred images containing approximately 1000 objects for which we know all the latent variables. Note that we are interested in the correct underlying scene interpretation, we evaluate accuracy in object detection and each object and global latent variable separately, so as each aspect of the scene reconstruction is assessed.

First, we evaluate the predictions of the global and object-specific latent variables. Finally, we evaluate the prediction qualitatively for real images, showing that the proposed framework is able to transfer to real images. Figure 3.7 shows an example of an inferred scene representation.

3.5.1 Detector

Figure 3.8 shows the precision-recall curve when varying the detection threshold from 0 to 1. Note that the precision-recall curve is near to the perfect classifier. The detector has 98% precision at 93% recall – we chose the threshold such as the false positives rate is much lower than the false negative rate. Thus superfluous objects are hence very rare (only 2%) and usually are due to multiple detections of different object parts, not an accidental detection in a noisy background. The inclusion of random real images in the training dataset significantly decreased false positive detections in the background by around one order of magnitude.

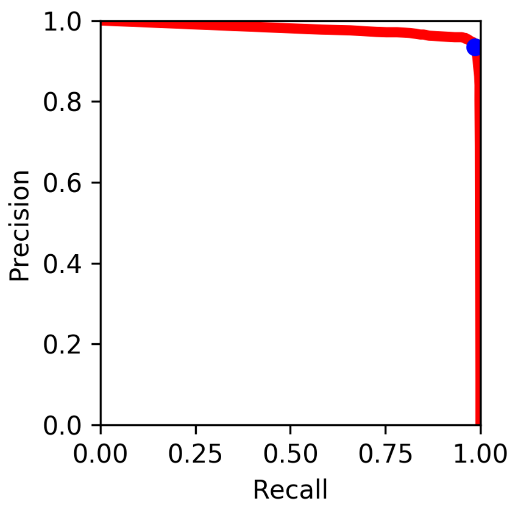


Figure 3.8: Detector: precision-recall plot. The blue dot indicates the performance at the threshold of 0.75, leading to 98% precision at 93% recall.

We treat the detection as positive if the predicted contact point is within a 24 pixel radius around the ground truth contact point. The average size of an object's bounding box in the Mugs dataset is 42×45 pixels, so this is about half the spatial size of the object. Figure 3.9 shows the histogram of the distances between the predicted and ground truth object contact points. The average distance is 5.4 pixels, this is very accurate compared to the average bounding box size of 42×45 pixels, and the image size of 256×256 pixels.

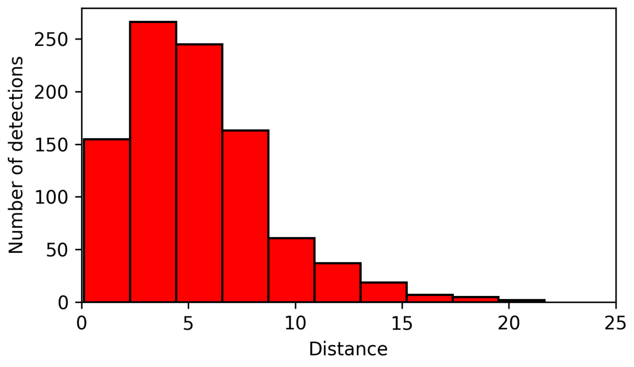


Figure 3.9: Histogram of the distances between the predicted contact point and the ground truth contact point, in pixels (the image size is 256×256 pixels).

3.5.2 Evaluation of Global LVs

The error metric of the ground plane colour is the mean square error (MSE) of colour (a,b) components in the Lab space[7]. The Lab space expresses a colour using three values: L is the lightness, (a,b) are green-red and blue-yellow colour components. By disregarding the L component, we evaluate the hue but not brightness, since multiplicative interaction between illumination and colour introduces a problem when evaluating the colours directly, as the actual illumination is unknown. The baseline is the mean intensity of the RGB channels in the training set[8].

The whole predicted uniform-directional lighting is projected onto a full sphere S, this produces intensity $i_{p,s}$ at any point s on the sphere. The illumination is evaluated at a fixed number of points uniformly distributed on a sphere. The points were generated using a set of 15 equidistant latitude lines, with the points at each latitude being at the same distance as the distance between each latitude line, using the algorithm from Deserno (2004). In our case this resulted in $N = 313$ points. Note one could use any method for choosing equidistant points, as long as the number of the generated vertices is high.

The strength of the illumination is evaluated at N points, and the errors between $i_{gt,s}$

[7]`https://en.wikipedia.org/wiki/Lab_color_space`

[8]The colour prediction results are better than in Romaszko et al. (2017). Earlier the colour GT values were noisy as they did not include the darkening effect of the textures, thus also the colour predictors were less accurate.

are computed by MSE, so as to approximate the LHS of

$$\frac{1}{I}\int_S (i_{p,s} - i_{gt,s})^2 ds \approx \frac{1}{I}\frac{1}{N}\sum_{n=1}^{N}(i_{p,n} - i_{gt,n})^2. \tag{3.5}$$

To allow comparison with other illumination intensity ranges, the illumination is scaled by a normalizing constant I so that the maximal potential error is unity.

For illumination we use a constant baseline that minimizes the error containing both uniform illumination and directional illumination from the top, at optimal strengths.

A thorough analysis and quantitative evaluation of the camera predictions are the main problem studied in Chapter 4. Note, however, that qualitative reconstructions in Figure 3.12 and Figure 3.13 show the quality of the predicted camera.

3.5.3 Evaluation of Object LVs

For azimuthal rotation we measure the absolute angular difference between the prediction and ground truth, but with wrap-around, so the maximum error is 180°. The baseline is a fixed rotation angle chosen to minimize the error. We evaluate the object colour in the same way as for the ground plane colour. For object shape prediction we make a 1-of-K classification ($K = 15$).

3.5.4 Scene Understanding – Quantitative Results

Global LVs	Baseline	CNN
Illumination [MSE]	0.084	0.025
Ground plane colour [MSE]	2.414	0.174

Table 3.2: Global latent variables: median errors. Colour errors $\times 10^2$.

Object LVs	Baseline	CNN
Azimuthal rotation [degrees]	91°	22°
Colour [MSE]	2.172	0.211

Table 3.3: Object latent variables: median errors. Colour errors $\times 10^2$.

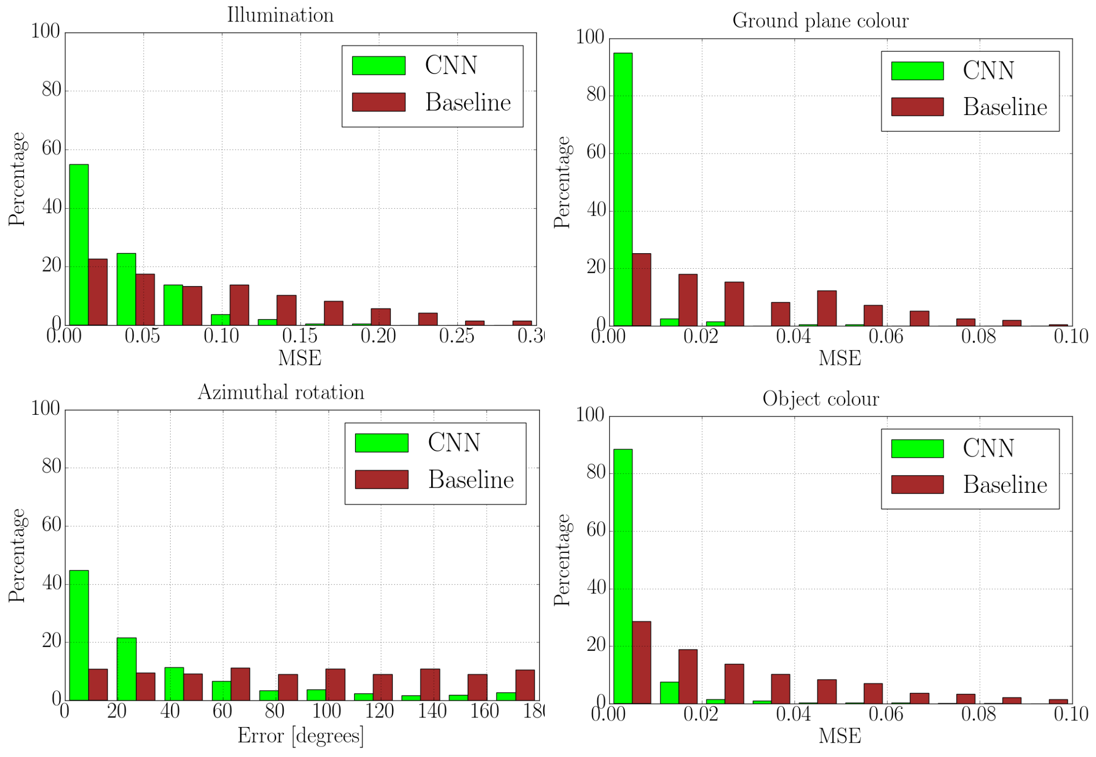
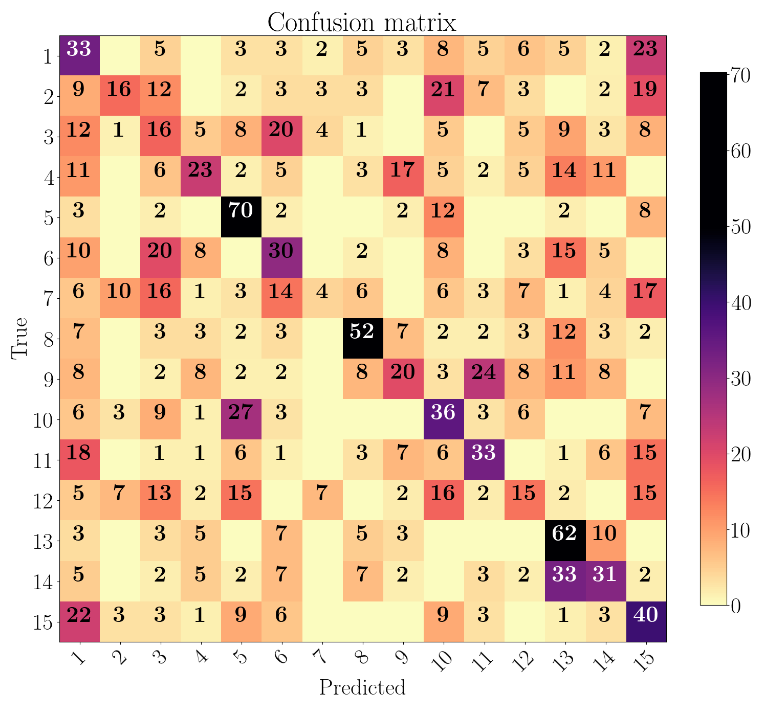


Figure 3.10: Histograms of the errors in the latent variables.

The results for the global latent variables and object latent variables are given in Table 3.2 and Table 3.3. We can note that for all the LVs our CNNs make predictions that are several times better than the simple baselines. For illumination the median predictions are more than 3 times better, for object azimuth more than 4 times better, while for the ground plane and object colours these are at least 10 times better.

The accurate predictions are further confirmed by the histograms of the prediction errors. Figure 3.10 shows the histograms of the errors in the latent variables for both the baseline (red) and our CNNs (green). We can note that for all the LVs for the CNN predictors the majority of the errors fall into the leftmost bin, with higher errors being much less frequent than the Baseline errors.

For the object shape classification the accuracy is 31.6%, compared to 6.7% for a random choice. This is a good result as often it is difficult to distinguish particular shapes, e.g. when a mug is viewed from the top, or from far away. Figure 3.11 shows the confusion matrix of the shape prediction. The best predictions are obtained for shapes that are not similar to other examples, e.g. mug number 13. On the other hand, examples 1 and 15 are similar, for these the majority of the prediction is that either these are classified correctly (73 times in total), or the prediction of these shapes is



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	33		5		3	3	2	5	3	8	5	6	5	2	23
2	9	16	12		2	3	3	3		21	7	3		2	19
3	12	1	16	5	8	20	4	1		5		5	9	3	8
4	11		6	23	2	5		3	17	5	2	5	14	11	
5	3		2		70	2			2	12			2		8
6	10		20	8		30		2		8		3	15	5	
7	6	10	16	1	3	14	4	6		6	3	7	1	4	17
8	7		3	3	2	3		52	7	2	2	3	12	3	2
9	8		2	8	2	2		8	20	3	24	8	11	8	
10	6	3	9	1	27	3				36	3	6			7
11	18		1	1	6	1		3	7	6	33		1	6	15
12	5	7	13	2	15		7		2	16	2	15	2		15
13	3		3	5		7		5	3				62	10	
14	5		2	5	2	7		7	2		3	2	33	31	2
15	22	3	3	1	9	6				9	3		1	3	40

Figure 3.11: Shape prediction – confusion matrix for 15 CAD shapes.

swapped, which happened 45 times (22+23).

3.5.5 Scene Understanding – Qualitative Results

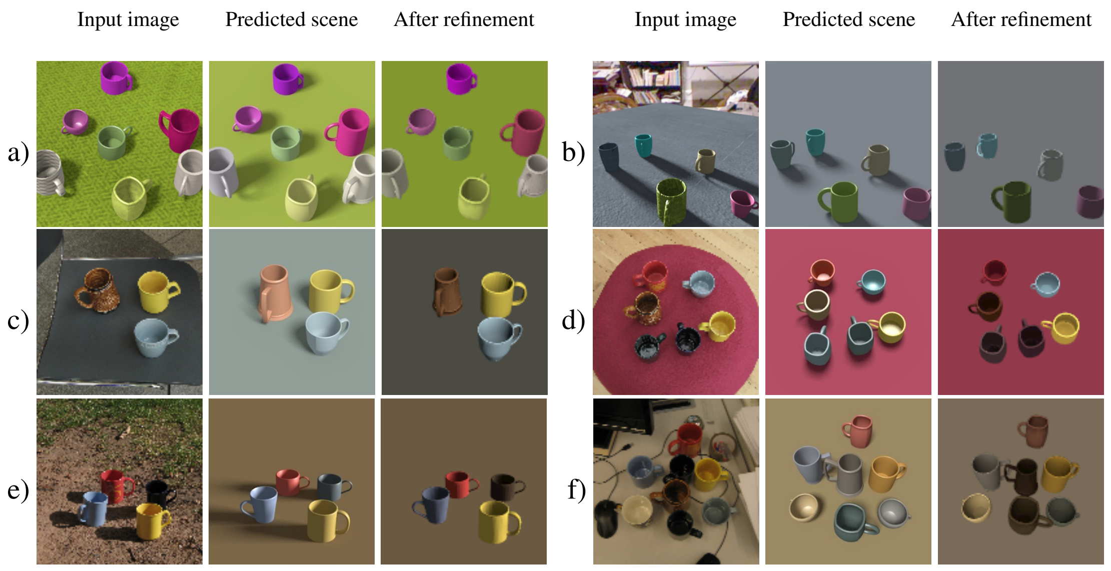


Figure 3.12: Results on synthetic (top row) and real scenes (middle, bottom). For each example the input image, predicted 3D scene, and result after iterative refinement are shown (left to right).

In Figure 3.12 we show results on both synthetic (top row) and real scenes (middle and bottom rows). We note that our methods work well on real images, despite not having been trained on them. The mugs are generally predicted well in location, azimuth and colour, and the camera parameters and lighting are in good agreement with the input image. The iterative refinement (rightmost panels of each example) mainly improves the colours of the objects. Note iterative refinement uses the OpenGL renderer, which cannot produce shadows. In (e) the directional light source is predicted almost properly. In the cluttered scene (f) all mugs are detected properly except one.

3.5.6 Scene Understanding – Additional Results

Figure 3.13 provides more examples of prediction for real images. The scenes are generally reconstructed well, and the predicted view-points are accurate. Again, the iterative refinement improves the colours of the objects and of the ground plane.

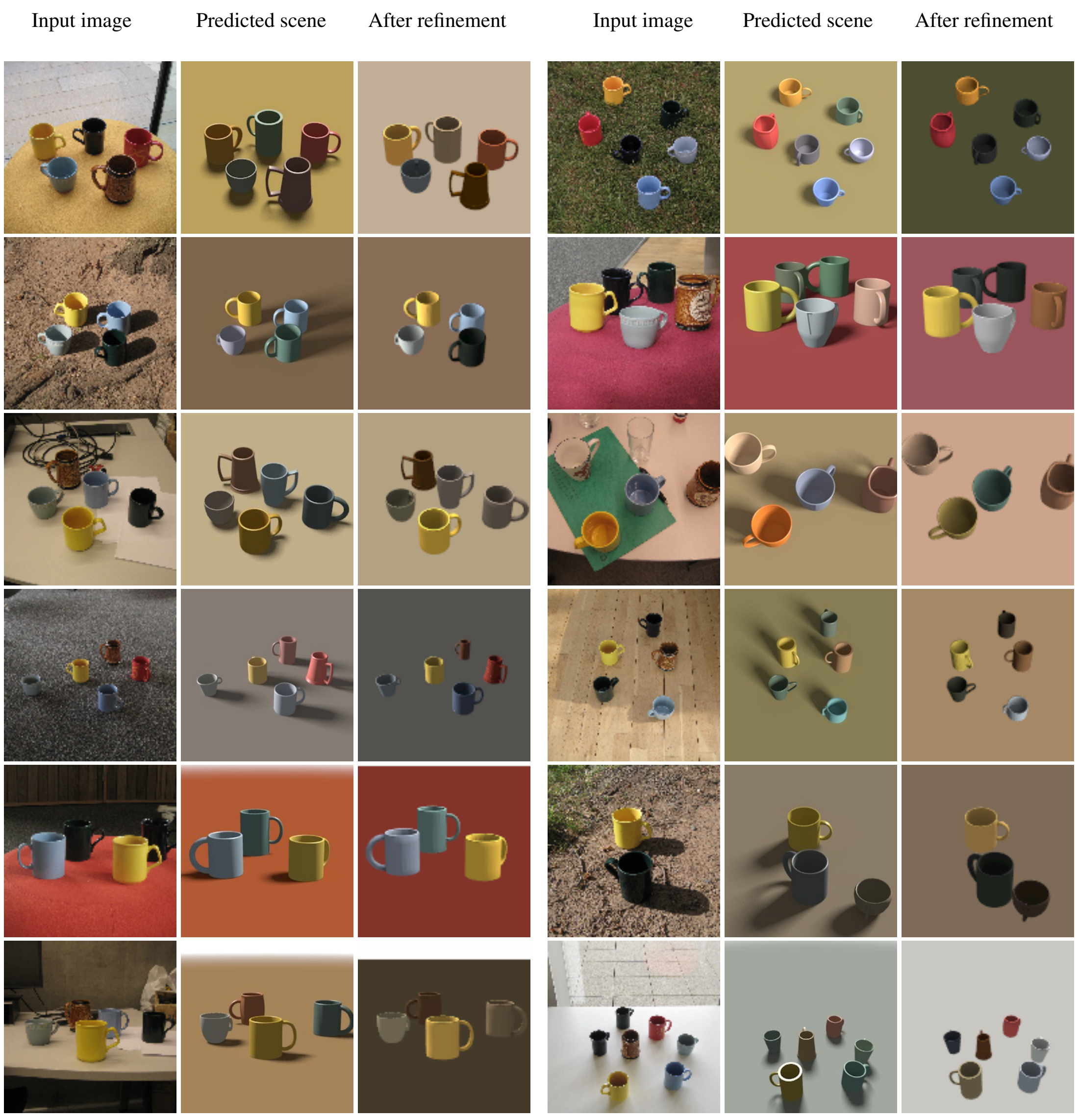


Figure 3.13: Results on real scenes. In each the order is the input image, predicted 3D scene, and the result after iterative refinement. Four examples in the bottom two rows have errors in detection, yet even in these cases the remaining objects are reconstructed correctly, as well as the correct view-point.

3.6 Discussion

We have shown how to successfully put all of the framework components together to create an interpretable scene-graph representation of a 3D scene from a single image. Importantly, the framework has been shown to work on both real and synthetic images. The models are trained using realistic synthetic images and corresponding ground truth scene graphs, obtained from a rich stochastic scene generator, to predict the object and global latent variables. In the framework we incorporate auxiliary modules for object contact point and projection scale prediction, to allow the computation of object poses and sizes in 3D scene coordinates, given the camera parameters.

The framework acts like an autoencoder, where the latent representation of the scene is interpretable and is of variable dimension to match the detected number of objects. The scenes for both synthetic and real images are generally reconstructed well, and the predicted view-points are accurate. We have shown that this interpretable representation can be used for editing the scene to further refine the initial predictions.

As outlined in the Background chapter, the competitor VIG methods usually handle only the geometry, but not the appearance of the objects. At the time this work was done (2016 – 2017) the related works (AIR, NSD) predicted only object class and poses, but not the appearance and size of objects, camera and illumination. The most related works did not exist (IM2CAD, 3DParsing). Although these works predict or refine the shape and the size of the objects, they still focus only on the object geometry and the camera.

There are a wide variety of possible extensions to explore, including the use of more object classes, and richer models of shape and appearance for each object class. Therefore, we perform experiments for a more complex dataset in Chapter 5.

Chapter 4

Probabilistic HoughNets

In this chapter we introduce *Probabilistic HoughNets* (PHNs), a new method for combining of the probabilistic votes. The method is applied to the camera estimation problem, where the votes are cast by the detected objects. One of the major challenges is the problem of placing the detected objects in 3D at a reasonable size and distance with respect to the single camera, the parameters of which are unknown. Previous VIG approaches for multiple objects usually only considered a fixed camera, while we allow for variable camera pose. In PHNs, each detection provides one noisy low-dimensional manifold in the Hough space, and by intersecting them probabilistically we reduce the uncertainty on the camera parameters. Section 4.1 discusses the background on the Hough transform, and Section 4.2 introduces the PHNs framework. Section 4.3 gives the analytical solution for the camera parameters given an observed object that should be expected in the noise-free set-up. Section 4.4 presents the details of the experiments. Section 4.5 provides the results, and the discussion is given in Section 4.6.

4.1 Introduction

We introduce Probabilistic HoughNets in order to combine information from a number of *voting elements*[1]. Our examples below are on the estimation of the camera parameters $\mathbf{z}$ based on detections of multiple objects $\{\mathbf{x}_i\}$ in a scene. Each voting element i has a local descriptor $\mathbf{x}_i$, and provides evidence via a mixture of Gaussians in the *Hough space* for the instantiation parameters $\mathbf{z}$. With Hough transforms, the predictions of a

[1]We use the terminology from Barinova et al. (2012).

voting element can lie (in the noise-free case) on a low-dimensional manifold in Hough space. When noise is present the manifold is "fuzzed out" in the remaining dimensions. In our case for the problem of the camera estimation, a 1D manifold arises from the trade-off between the distance of an object from the camera and the camera's focal length (or zoom) in creating an object image of a given size. Each detection provides one such manifold, and by intersecting them (probabilistically) we reduce the uncertainty on the camera parameters. Standard approaches for camera pose estimation use either known objects (e.g. checkerboards) or exploit structure like the vanishing points of lines in the scene, but these are not available in our scenes.

The Hough transform (HT) is a classic computer vision algorithm dating back to 1962 (Hough, 1962). It was originally proposed for detecting lines, but was then generalized by Ballard (1981) to arbitrary templates. A classic Hough example would be the evidence provided by a point $\mathbf{x}_i$ in 2D about the parameters $\mathbf{z}$ of a straight line passing through that point, and the combination of this information across different points to provide evidence about straight lines in the data.

Let the set of voting elements $\{\mathbf{x}_i\}$ be denoted by X. Stephens (1990) pointed out how the HT can be made probabilistic by writing

$$p(\mathbf{z}|X) = \frac{p(\mathbf{z})p(X|\mathbf{z})}{p(X)} = \frac{p(\mathbf{z})}{p(X)} \prod_{i=1}^{n} p(\mathbf{x}_i|\mathbf{z}), \tag{4.1}$$

assuming that the $\mathbf{x}_i$'s are conditionally independent given $\mathbf{z}$. By taking logs of this equation Stephens shows how terms involving $\log p(\mathbf{x}_i|\mathbf{z})$ can be added up, mirroring the standard Hough space accumulator. If $\mathbf{x}$ is high dimensional (e.g. an image patch) and $\mathbf{z}$ is low dimensional it makes more sense to model $p(\mathbf{z}|\mathbf{x}_i)$ rather than $p(\mathbf{x}_i|\mathbf{z})$. Applying Bayes' theorem again to eq. 4.1 we obtain

$$p(\mathbf{z}|X) \propto \frac{\prod_{i=1}^{n} p(\mathbf{z}|\mathbf{x}_i)}{p(\mathbf{z})^{n-1}}, \tag{4.2}$$

ignoring terms involving $p(X)$ or $p(\mathbf{x}_i)$ which are fixed given the image evidence. This argument in eq. 4.2 was given in Allan and Williams (2009, §3.5) and Barinova et al. (2012). Gall et al. (2011) used random forests to also predict (in our notation) $p(\mathbf{z}|\mathbf{x}_i)$, and were able to obtain good results for problems of object detection, tracking and action detection. Their method makes predictions in Hough space for each $\mathbf{x}_i$ as a set of Gaussians at $\mathbf{x}_i$-dependent locations, and then uses a probabilistically incorrect method of summing the predictive densities (rather than their logs); as explained in Allan and Williams (2009, §3.5) this can be seen as an approximation due to robustification.

4.2 Probabilistic HoughNets

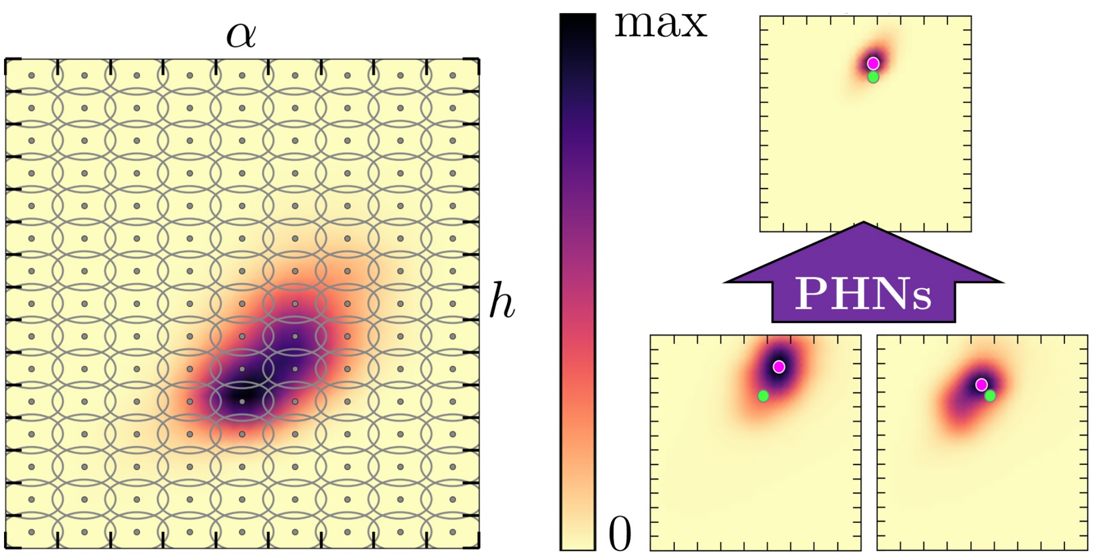


Figure 4.1: **Left:** The density plot predicted by a CNN given one voting element; the dots are Gaussian means, ellipses show standard deviations; and the colour shows the overall density. **Right:** combining multiple PHNs; green dot is the GT, red dot is the MAP.

Figure 4.1 (left) illustrates a prediction in the Hough space given one voting element, represented as a mixture of Gaussians; these predictions are then combined to give a final predictive distribution for $\mathbf{z}$. To obtain a point estimate we can then find its mode, Maximum a Posteriori (MAP).

The Probabilistic HoughNet represents $p(\mathbf{z}|\mathbf{x}_i)$ using a mixture of Gaussians with the means $\{\boldsymbol{\mu}_j\}$ arranged in a grid in Hough space:

$$p(\mathbf{z}|\mathbf{x}_i) = \sum_j c_j(\mathbf{x}_i)\mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}), \tag{4.3}$$

where j is an index over the grid, and the $c_j(\mathbf{x}_i)$'s are $\mathbf{x}$-dependent mixing coefficients. These are estimated by using a softmax layer at the output of a deep convolutional neural network, as in a mixture of experts (Jacobs et al., 1991), but where only the mixing proportions but not the $\boldsymbol{\mu}_j$'s or $\boldsymbol{\Sigma}$ depend on $\mathbf{x}$. We train the neural network by maximizing the log-likelihood of the ground-truth instantiation parameters given a voting element.

If the Hough space dimension d is high and there are N components per dimension in the grid, then the softmax layer will have N^d outputs parametrized by a large number of weights, which could lead to overfitting. In this case one can make use of the chain rule, e.g. splitting $\mathbf{z}$ into $\mathbf{z}_1 \cup \mathbf{z}_2$ and then writing $p(\mathbf{z}|\mathbf{x}_i) = p(\mathbf{z}_1|\mathbf{x}_i)p(\mathbf{z}_2|\mathbf{z}_1, \mathbf{x}_i)$. In this fashion the exponential scaling of the softmax outputs with d can be mitigated.

We use a grid-based representation of the $\{\mu_j\}$s in all dimensions in the experiments to show how to deal with complex multimodal distributions, and to confirm the efficiency of our method even when predicted distributions have a large number of components, e.g. when a PHN would predict a 3-dimensional manifold in a higher dimensional space. Instead of using a grid in Hough space as in eq. 4.2 it would be possible to use a more general mixture of experts framework where the $\boldsymbol{\mu}_j$'s and respective covariance matrices depend on the input; this would likely require fewer experts but would make the PHN networks much more complex and difficult to train.

An important aspect of the Hough transform is the ability to deal with outliers; this can be handled by robustification, replacing $p(\mathbf{z}|\mathbf{x}_i)$ in eq. 4.2 with $\gamma p(\mathbf{z}|\mathbf{x}_i) + (1-\gamma)p_0(\mathbf{z})$ for $\gamma \in [0,1]$, where $p_0(\mathbf{z})$ is a broad prior over $\mathbf{z}$-space, i.e. :

$$p(\mathbf{z}|X) \propto \frac{\prod_{i=1}^{n}(\gamma p(\mathbf{z}|\mathbf{x}_i) + (1-\gamma)p_0(\mathbf{z}))}{p(\mathbf{z})^{n-1}}. \tag{4.4}$$

The outputs from eq. 4.3 are combined as per eq. 4.4, and inference is carried out by seeking the mode $\mathbf{z}_{MAP}$ of $\log p(\mathbf{z}|X)$ by using BFGS hill-climbing search. The details are given in the next section.

4.2.1 Finding the MAP of the Joint Posterior:

The density $p(\mathbf{z}|X)$ in eq. 4.2 is a product of several densities that are each a Gaussian Mixture Model (GMM). Although a product of two GMMs is still a GMM, the resulting product cannot be computed directly (even for only a few observations) due to the exponential scaling of the number of components. In the PHNs framework the whole computation given the terms obtained from single PHNs is exact. We maintain the functions $p(\mathbf{z}|\mathbf{x}_i)$ and $p_0(\mathbf{z})$, and do not create the mixture with $O(N^{dn})$ components explicitly. For each observation i we obtain $p(\mathbf{z}|\mathbf{x}_i)$ from a PHN and we store associated GMM coefficients. To search for $\mathbf{z}_{MAP}$, we can then evaluate the function at any point and obtain the gradient using Automatic-Differentiation (AD). The derivative is with respect to only d variables, so it is quick to compute by AD. The gradient may be also used for efficient sampling from $p(\mathbf{z}|X)$, such as Hamiltonian Monte Carlo (Neal, 2010).

To search for $\mathbf{z}_{MAP}$, we perform the BFGS hill-climbing in the log-space, i.e. we maximize $\log p(\mathbf{z}|X)$. We first find the maximal value of the target function at the locations of all the Gaussians. We then start the hill-climbing multiple times, from all the Gaussian centres at which the target function value is larger than a certain fraction of the maximal value found initially.

4.2.2 Probabilistic Chain Rule within PHNs

In a case of a multidimensional grid-based GMM, it may be undesirable to make predictions directly in the d-dimensional Hough space, as this has $O(N^d)$ mixture components, where N is the average number of components per dimension. Assuming the number of hidden units in the layer before the softmax one is on average U, the number of weights in the *softmax* layer is $O(UN^d)$, which would likely lead to over-fitting.

Fortunately PHNs allow us to decompose a single PHN that models $p(\mathbf{z}|\mathbf{x}_i)$ into a number of simpler networks via the probabilistic chain rule. We decompose our network which in a direct 3D case has $O(UN^3)$ weights into two networks with a GMM of dimension 2 and 1 (e.g. 2D: height h and angle α, 1D: angle of view ω), so $O(UN^2 + UN) = O(UN^2)$ weights. The resulting networks are conditional, where one needs to iterate through the grid of the variables that the network is conditioned on to create the full prediction, the details of this decomposition-composition procedure are given in the next two sections. In general we can decompose any grid-based PHN to a single PHN for each dimension, leading to a total of $O(dUN)$ weights in the last layers. Thus using more but smaller networks we can obtain a GMM which if predicted directly would lead to a network that would have too many weights compared to the dataset size we have.

4.2.2.1 PHN Decomposition

We use the following decomposition of a PHN modelling $p(\mathbf{z}|\mathbf{x}_i)$, where $\mathbf{z} = \mathbf{z}_1 \cup \mathbf{z}_2$:

$$p(\mathbf{z}|\mathbf{x}_i) = p(\mathbf{z}_1, \mathbf{z}_2|\mathbf{x}_i) = p(\mathbf{z}_1|\mathbf{z}_2, \mathbf{x}_i)p(\mathbf{z}_2|\mathbf{x}_i). \tag{4.5}$$

We note that we need a decomposition-composition procedure to produce the whole PHN network, as we do not simply multiply the values, but create the whole density that represents the product in the whole space. We decompose the full PHN $H = p(\mathbf{z}|\mathbf{x}_i)$ into two PHNs:

$$H^1(\mathbf{z}_1|\mathbf{z}_2, \mathbf{x}_i) = p(\mathbf{z}_1|\mathbf{z}_2, \mathbf{x}_i), \tag{4.6}$$

$$H^2(\mathbf{z}_2|\mathbf{x}_i) = p(\mathbf{z}_2|\mathbf{x}_i). \tag{4.7}$$

H^1 predicts the probabilistic vote conditioned on the observation and a Hough space variable (or a set of variables) $\mathbf{z}_2$ given which there is a simpler solution.

4.2.2.2 PHN Composition

Suppose $\dim(H^1) = d_1$ and $\dim(H^2) = d_2$, with $\dim(H) = d = d_1 + d_2$. We want to compose the GMM obtained from H^1 whose components are indexed by $(i_{\mathbf{z}_1})$ and H^2 whose components are indexed by $(i_{\mathbf{z}_2})$ into a d-dimensional GMM represented by H and indexed by $(i_{\mathbf{z}_1}, i_{\mathbf{z}_2})$. We can obtain $H = H^1 \otimes_{\mathbf{PHN}} H^2$ representing the density $p(\mathbf{z}|\mathbf{x}_i)$ from these two models as follows: we iterate through the values in the $\mathbf{z}_2$ dimension at the positions of the components as defined by the grid, denoted by $\mathbf{z}_2(i_{\mathbf{z}_2})$. Then we evaluate H^1 by conditioning on each of these values, obtaining $N_{\mathbf{z}_2}$ GMM slices in the $\mathbf{z}_1$-space. We also predict GMM in the $\mathbf{z}_2$-space using H^2. The GMM mixing coefficients are given by:

$$H(i_{\mathbf{z}_1}, i_{\mathbf{z}_2}|\mathbf{x}_i) = H^1(i_{\mathbf{z}_1}|\mathbf{x}_i, \mathbf{z}_2(i_{\mathbf{z}_2}))H^2(i_{\mathbf{z}_2}|\mathbf{x}_i), \tag{4.8}$$

which directly represents a probability distribution in the desired grid, where each component mean is at the location of respective indices and the covariance matrix is the same as before the decomposition.

4.3 Exact Solution for the Camera Latent Variables

We use PHNs as per eq. 4.2 to find the most likely camera configuration. This section analyses the solution for camera parameters given the view-dependent (projected) variables of an observed object. This allows us to understand the expected shape of the votes in the Hough space.

4.3.1 Motivation

Although the view-point at which objects are observed provide clues for the camera parameters, one may notice that the mapping from observed object to camera parameters is not straightforward, as the view of an object is affected by its position in the image frame. For example different sides of an object can be visible as shown in Figure 4.2 (right) when using a standard pin-hole camera model with perspective, even though the global rotation of each object in the scene frame is the same (left). For a given focal length, we know how the view of a given object is affected due to its position in the image frame. The shorter the focal length, the larger the deviation of the viewpoint of the observed object. The view elevation angle at which an object is visible provide clues about the camera elevation (camera tilt), while the size of the object provide clues about the camera height. In the next section we derive a solution for the camera parameters given the observed object.

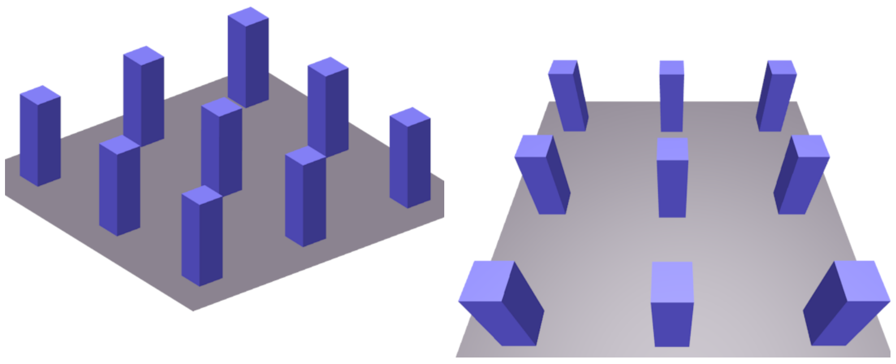


Figure 4.2: Left: 3D scene (orthogonal view), right: differing appearances of identical objects (blue pillars) at the same global rotation when using a pin-hole camera with perspective.

4.3.2 Angle of View vs Focal Length

Figure 4.3 presents the scene projection diagram. The camera parametrization is $\mathbf{z}^{cam} = (\alpha, h, \omega)$: camera elevation, camera height, and angle of view, as described in Section 3.2.2.1. In the figure one can see that the focal length f and the angle of view (AoV) ω are related as follows: $f = a_0/(2\tan(\omega/2))$, where a_0 is the known sensor size.

The main reason for choosing the AoV parametrization rather than the focal length is that it is easier to interpret, as the AoV is very related to the property of interest: the zoom. The AoV does not depend on the sensor size, and AoV is easier to model as in general the range is bounded by $(0, 180°)$. The focal length is not bounded as its range is $(0, +\infty)$, the viewpoint depends on the sensor size, the value is not easy to interpret even if we know the sensor size. Therefore, we initially use the focal length as this allows to derive the analytical solution, and then we express it using the AoV. Since AoV is a monotonic function of the focal length, the 1D shape of the manifold still holds.

4.3.3 Solution

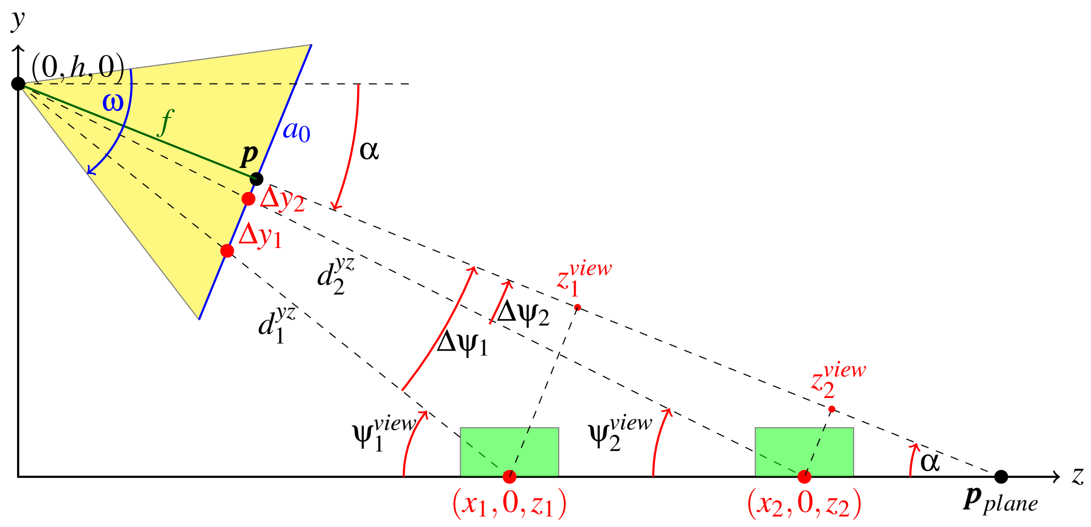


Figure 4.3: Different observed view elevations (ψ_1^{view}, ψ_2^{view}) for two objects. The focal length is denoted by f and the principal point by $\boldsymbol{p}$. The sensor size is denoted by a_0.

Figure 4.3 depicts camera parameters and several observed angles. The y-axis points upwards, the objects are located on the x-z plane, where the x coordinate controls left/right object positioning, and z coordinate controls the depth. The diagram is a projection of the scene along the x-axis onto the y-z plane.

Importantly, we can derive the solution in the projected configuration and disregard values of the x-coordinates. This is due to several reasons outlined below. As the camera elevation is defined as the rotation around the x-axis, hence all the elevation angles of our interest lie also in the y-z plane. Due to the pin-hole camera properties and no camera roll (in-plane) rotation, the projected object size does not change when shifting an object along the x-axis. This also does not affect any of the depicted viewpoint variables, as these are defined as the rotation around the camera's x-axis, which is the same as the global x-axis.

To derive the dependencies between object view and camera variables, we will first define all additional variables used in Figure 4.3.

- The object contact point (o denotes the o-th object) in the 3D scene is given by $(x_o, 0, z_o)$.
- The view-dependent (projected) object variables are: deviation vector along y-axis from the principal point $\boldsymbol{p}$ to the contact point in the image frame Δy, view elevation (ψ_o^{view}) and projection scale s_o^{view} for each object o. As these can be measured in the image space given a known shape and a constant size of an object, these can be predicted, e.g. by a neural network.
- Deviation of the viewpoint elevation angle from the elevation angle at which an object located in the centre would be observed is denoted by $\Delta\psi_o$. This is defined as the angle around the camera origin, between the principal point and the object contact point. see Figure 4.3. This angle can be calculated using Δy.
- The distance projected onto the y-z plane of the object o to the camera origin at $(0, h, 0)$ is given by d_o^{yz}.
- The depth of the object z_o^{view} is the distance from the camera at $(0, h, 0)$ to the projection of contact point $(x_0, 0, z_o)$ on the line (camera origin – principal point $\mathbf{p} - \mathbf{p}_{plane}$) and is denoted by $z_o^{view} = d_o^{yz} \cos \Delta\psi_o$.

Since we assume a pin-hole camera, the actual object's size is inversely proportional to the depth of the object z^{view}. However, as the focal length f is unknown, we can directly predict only the scale of the projection $s^{view} = \frac{f}{z^{view}}$ (Szeliski, 2010, page 57). If the object size is not constant, this introduces the noise around the exact solution for the mean object size. Below we present the relations between the camera and the view variables. The relations between the angle variables can be easily seen in Figure 4.3.

First, we can note $\Delta\psi_o$ can be calculated from the f–Δy_o triangle (o denotes the o-th object):

$$\Delta\psi_o = \arctan \frac{\Delta y_o}{f}. \tag{4.9}$$

The camera elevation angle is the view elevation of an object excluding the additional influence of the viewpoint due to the elevation deviation. However, note $\Delta\psi_o$ rotation is anticlockwise, while the other angles, ψ_o^{view} and α, are clockwise. Therefore, we multiply $\Delta\psi_o$ by -1 so as the usual sum of absolute angles holds as can be observed in the diagram, thus:

$$\alpha = \psi_o^{view} - (-1)\Delta\psi_o = \psi_o^{view} + \Delta\psi_o. \tag{4.10}$$

From the triangle origin–camera–contact point $(x_o, 0, z_o)$ we have:

$$h = d_o^{yz} \sin\psi_o^{view}. \tag{4.11}$$

We know the formulas for object's depth:

$$s_o^{view} \overset{\text{def}}{=} \frac{f}{z_o^{view}} \Longrightarrow z_o^{view} = \frac{f}{s_o^{view}}. \tag{4.12}$$

But from the diagram in Figure 4.3 we also have that

$$z_o^{view} = d_o^{yz} \cos\Delta\psi_o \tag{4.13}$$

thus the distance and the camera height are given by

$$d_o^{yz} = \frac{z_o^{view}}{\cos\Delta\psi_o} = \frac{f}{s_o^{view}\cos\Delta\psi_o}. \tag{4.14}$$

Substituting eq. 4.14 and eq. 4.9 into eq. 4.11 we obtain

$$\begin{aligned} h = d_o^{yz}\sin\psi_o^{view} &= \frac{f}{s_o^{view}\cos(\arctan\frac{\Delta y_o}{f})}\sin\psi_o^{view} = \\ &= \frac{f\sqrt{1+(\frac{\Delta y_o}{f})^2}}{s_o^{view}}\sin\psi_o^{view} = \frac{\sqrt{f^2+\Delta y_o{}^2}}{s_o^{view}}\sin\psi_o^{view}. \end{aligned} \tag{4.15}$$

Assuming we know the object view variables (no-noise setup), the camera elevation angle α and the camera height h for a single object, given the unknow focal length f and other known variables, are given by:

$$\begin{cases} \alpha = \psi_o^{view} + \arctan\dfrac{\Delta y_o}{f}, & (4.16) \\ h = \sin\psi_o^{view}\dfrac{\sqrt{f^2+\Delta y_o{}^2}}{s_o^{view}}. & (4.17) \end{cases}$$

Since f is a function of ω, i.e. $f = a_0/(2\tan(\omega/2))$, where a_0 is the known sensor size, we obtain (where we can vary ω):

$$\begin{cases} \alpha = \psi_o^{view} + \arctan\dfrac{2\Delta y_o \tan\frac{\omega}{2}}{a_0}, & (4.18) \\ h = \dfrac{\sin\psi_o^{view}}{s_o^{view}}\sqrt{\left(\dfrac{a_0}{2\tan\frac{\omega}{2}}\right)^2 + \Delta y_o{}^2}. & (4.19) \end{cases}$$

The equations above give us the solutions for the camera parameters that we aimed to find. There are three variables and two equations, so possible solutions for an object of the constant scale in the 3D scene are represented by a curve, as the focal length f (eq. 4.16, 4.17) or AoV ω (eq. 4.18, 4.19) is unknown. Hence, we need to aggregate the solutions obtained from all objects to obtain a single prediction, and also to increase the predictive accuracy.

Above we have assumed that objects are of constant size, thus the solution is a curve. However, as we impose a prior on the scaling factor, we accept some variability in object size. The predicted camera configuration will produce a likely scene, still maintaining realistic object scales.

Figure 4.6 presents a few votes in the Hough space. From the top view ($\alpha \times f$ coordinates), each vote has an asymptote at α which value is equal to the estimated elevation angle ψ_o (see eq. 4.16).

4.3.4 Practical Example

Below we compare two examples of the voting for prediction of camera latent variables in the Hough space. Figure 4.5 presents a single scene configuration together with two different cameras that produce the renders shown in Figure 4.4.

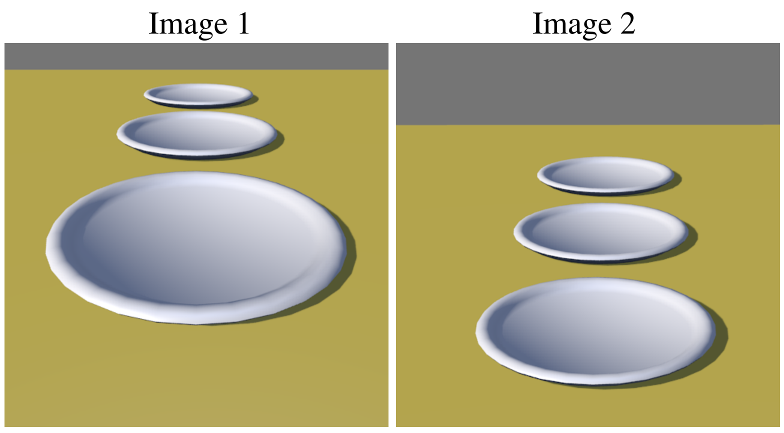


Figure 4.4: Renders of the scene using Camera 1 and Camera 2. The plates are all of the same size.

The observed variables used for voting are: size s^{view}, view elevation ψ^{view}, y-deviation Δy. The deviation from the image centre Δy is given as a fraction of the image height. Note s^{view} is proportional to the observed width (diameter) of a plate, and the value of

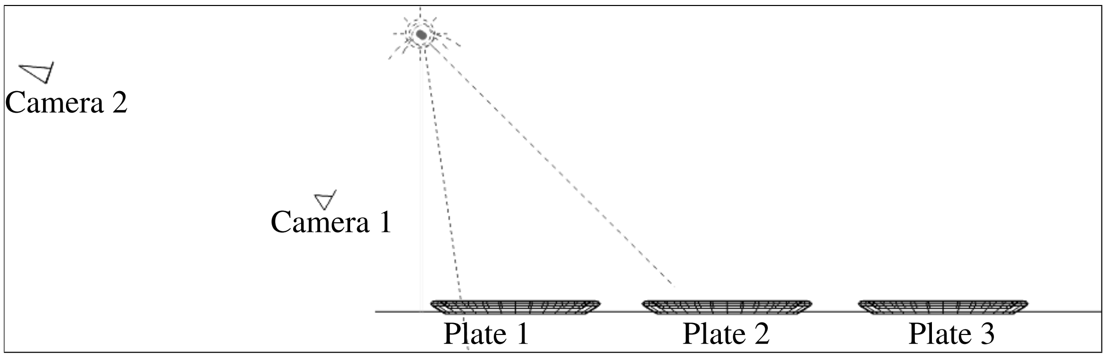


Figure 4.5: A side view of the 3D scene.

s^{view} and then the camera pose can be calculated due to a fixed and known 3D size of each plate.

The plates are enumerated starting from the bottom of the image. Table 4.1 provides values of the latent variables read out approximately from the rendered images.

Object	**variables, Image 1**	**variables, Image 2**
Camera	$\alpha = 30°, h = 0.75, f = 35\text{mm}$	$\alpha = 20°, h = 1.5, f = 70\text{mm}$
Plate 1	$\Delta y = +0.00$, $\psi^{view} = 30°$, $s^{view} = 4.5$	$\Delta y = -0.25$, $\psi^{view} = 25°$, $s^{view} = 3.5$
Plate 2	$\Delta y = +0.25$, $\psi^{view} = 15°$, $s^{view} = 2.5$	$\Delta y = +0.00$, $\psi^{view} = 20°$, $s^{view} = 2.5$
Plate 3	$\Delta y = +0.35$, $\psi^{view} = 10°$, $s^{view} = 1.5$	$\Delta y = +0.17$, $\psi^{view} = 15°$, $s^{view} = 2.0$

Table 4.1: Scene configuration (GT Camera) and measured view variables (approximate).

Figure 4.6 shows plots of the votes in the Hough space using variables from Table 4.1 within eq. 4.16 and 4.17, with artificial noise added. The votes intersect each other, and very close the GT camera parameters, even though the values of the variables are approximate. Note the camera elevation and height are estimated correctly, with all the votes passing through the circle-mark representing the GT camera parameters.

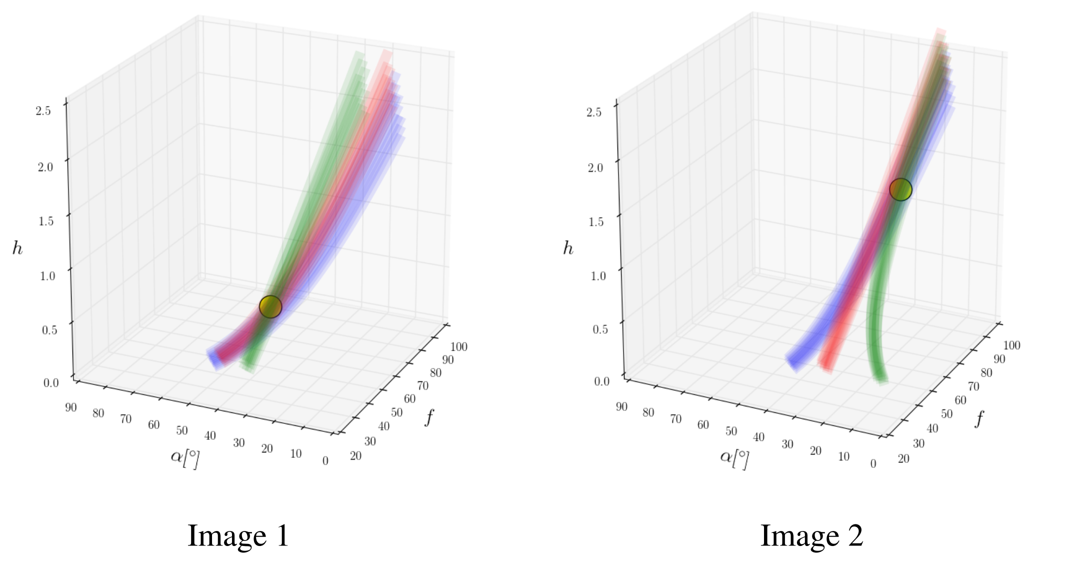


Figure 4.6: The voting in the Hough space. Camera ground-truth latent variables are located at the circle-mark. Lines of each colour represent votes of each plate. They represent votes at the detected distance and angle with the following ranges of the errors, visualizing uncertainty: $[-10\%, +10\%]$ of the camera height h, and $[-2°, 2°]$ of the camera elevation angle α.

Conclusion: we expect the votes for the camera to be noisy 1-dimensional manifolds, where the noise comes from the noise or errors in predictions and noise due to the variable object size.

4.4 Experiments

We consider two cases: one with a single solution, and another with multiple solutions, i.e. 2D and 3D cases. We refer to a '3D case' when we consider predictions in (α, h, ω)-space. When we know or have already predicted the camera AoV and predict in (α, h)-space given the AoV, we refer to it as a '2D case'. They are of a different nature, as the noise-free 2D solution is a point, while for 3D, the solution is a 1D manifold.

In this section we discuss the experimental set-up. First, we provide the details of the Hough space and of the GMM within PHNs. We then introduce a re-projection error used for the evaluation of the MAP, and provide details of computation of the log-likelihood. Finally, we describe the configuration of neural networks and of the

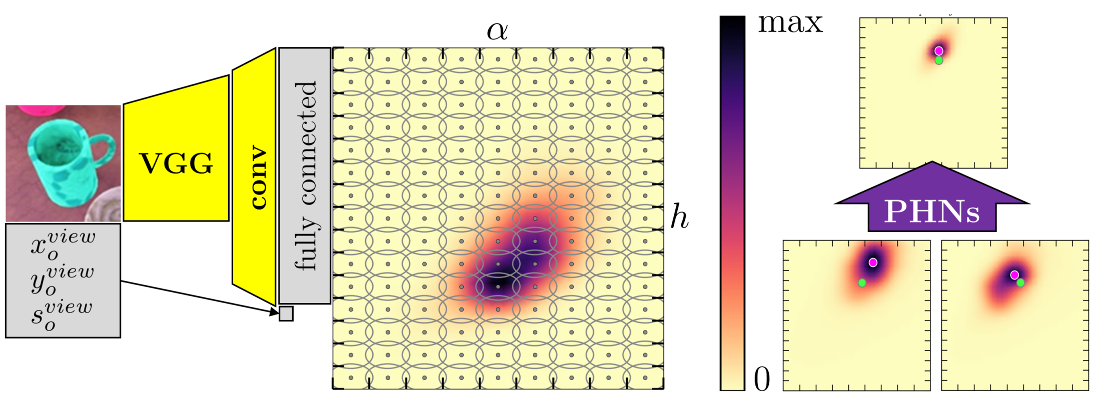


Figure 4.7: **Left:** Probabilistic HoughNets framework (see Section 4.4.5 for details). Given the detected object, the input to a PHN is the image patch plus position (x_o^{view}, y_o^{view}) of the image patch and size s_o^{view} (projection scale) of the object. The PHN consists of VGG layers (fixed), and convolutional and fully connected layers trained on top. The output are the mixing coefficients of the Gaussian mixture model. The density plots represent predictions in (α, h)-space conditioned on ω; the dots are Gaussian means, ellipses show standard deviations; **Right:** combining multiple PHNs; green dot is the GT, red dot is the MAP.

training procedure.

4.4.1 The Hough Space Used in the Experiments

The size of the grid where GMM means are located that represent (α, h, ω)-space is $N_\alpha N_h N_\omega = 9 \times 15 \times 10$, a total of 1350 components. The spacing between components are $(\Delta_\alpha, \Delta_h, \Delta_\omega) = (10°, 10\text{cm}, 4°)$. The parameter γ (see Equation 4.4) of the robust model was set to 0.50 and the prior is uniform in the Hough space. See Figure 4.7 for the visualization of the approach for the 2D case.

We represent the camera LVs as mixture of Gaussians in 3D space, which allows us to illustrate how PHNs can handle complex or multimodal distributions. In our case we decompose the PHN using the chain rule (see sec. 4.2.2) into two PHNs: H^1 predicts $p(\alpha, h|\mathbf{x}_i, \omega)$, as a GMM with means located in the grid of centres of size $N_\alpha N_h$, and H^2 predicts $p(\omega|\mathbf{x}_i)$ with a grid of size N_ω.

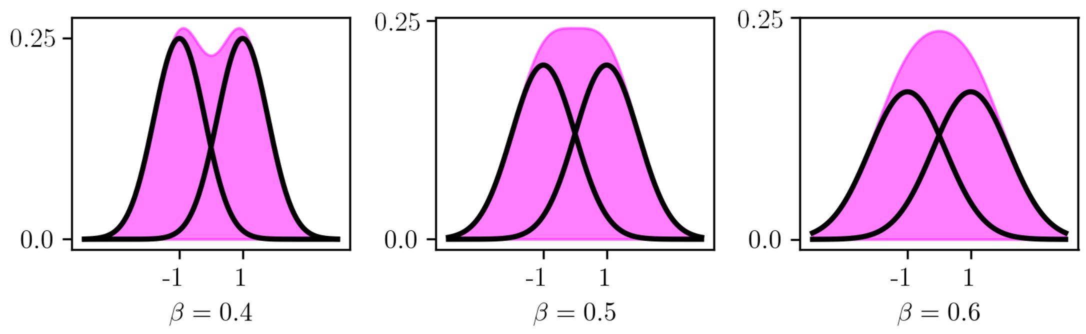


Figure 4.8: A simple one-dimensional PHN composed of two Gaussians (in black colour, with the mean at -1 and 1) for three different standard deviation scaling factors β. The mixing proportions of the Gaussians are equal, together they sum up to the magenta distribution. In the leftmost plot $\beta = 0.4$ does not create a mode between the Gaussians, $\beta = 0.5$ produces a flat peak, $\beta = 0.6$ produces a distribution with the mode between the two components.

The covariance matrices of the Gaussians for H^1 and H^2 are respectively:

$$\mathbf{\Sigma}_1 = \beta^2 \mathrm{diag}(\Delta_\alpha^2, \Delta_h^2), \tag{4.20}$$

$$\mathbf{\Sigma}_2 = \beta^2 \Delta_\omega^2. \tag{4.21}$$

The standard deviation scaling factor β is a global single value applicable to all PHNs. The β factor defines the standard deviation in a given dimension as a fraction of a distance Δ between Gaussian components, i.e. $\sigma = \beta\Delta$, thus:

$$\beta = \sigma/\Delta. \tag{4.22}$$

Note $\beta = 1$ would correspond to the distance between the components on the grid, and would produce distributions that are too blurred. Setting β to lower values, such as 0.4, would prevent the GMM from obtaining modes between the components, see Figure 4.8. We have found $\beta = 0.6$ to produce smooth distributions which are not blurred. We can note that the borderline β to obtain a mode between the components is approximately 0.5. Figure 4.8 visualizes only a one-dimensional case. The analysis in Section 4.4.2 shows that $\beta = 0.5$ is the exact critical value for the centre point of a Gaussian hypercube.

4.4.2 Critical Value of β for Obtaining a Maximum

The analysis below provides the derivation for which β the local maximum is obtained. This is examined at the centre point of a Gaussian hypercube.

We consider a Gaussian hypercube, composed of 2^D D-dimensional spherical Gaussians ($\mathbf{\Sigma} = \mathrm{diag}(\sigma^2, \sigma^2, ..., \sigma^2)$) located at $\boldsymbol{\mu} = (\pm 1, \pm 1, ..., \pm 1)$. Note for $D = 1$ this results in two one-dimensional Gaussians, analogously to the set-up considered in Figure 4.8.

Let $\gamma = 1/\sigma^2$, so $\mathbf{\Sigma}^{-1} = \mathrm{diag}(\gamma, \gamma, .., \gamma)$. The density and its first and second partial derivatives are as follows:

$$p(\mathbf{x}) = c \sum_{i=1}^{2^D} e^{-\frac{\gamma}{2}|\mathbf{x}-\boldsymbol{\mu}_i|^2}, \tag{4.23}$$

$$\frac{\partial p}{\partial x_j} = -c \sum_{i=1}^{2^D} \gamma(x_j - \mu_{ij}) e^{-\frac{\gamma}{2}|\mathbf{x}-\boldsymbol{\mu}_i|^2}, \tag{4.24}$$

$$\frac{\partial^2 p}{\partial x_j \partial x_k} = \begin{cases} c\sum_{i=1}^{2^D} \gamma^2 (x_j - \mu_{ij})(x_k - \mu_{ik}) e^{-\frac{\gamma}{2}|\mathbf{x}-\boldsymbol{\mu}_i|^2} & \text{for } j \neq k, \\ c\sum_{i=1}^{2^D} (\gamma^2 (x_j - \mu_{ij})^2 - \gamma) e^{-\frac{\gamma}{2}|\mathbf{x}-\boldsymbol{\mu}_i|^2} & \text{for } j = k, \end{cases} \tag{4.25}$$

$$\left.\frac{\partial^2 p}{\partial x_j \partial x_k}\right|_{\mathbf{x}=\mathbf{0}} = \begin{cases} c\sum_{i=1}^{2^D} \gamma^2 \mu_{ij}\mu_{ik} e^{-\frac{\gamma}{2}|\boldsymbol{\mu}_i|^2} & \text{for } j \neq k, \\ c\sum_{i=1}^{2^D} (\gamma^2 \mu_{ij}^2 - \gamma) e^{-\frac{\gamma}{2}|\boldsymbol{\mu}_i|^2} & \text{for } j = k. \end{cases} \tag{4.26}$$

Let $ce^{-\frac{\gamma}{2}|\boldsymbol{\mu}_i|^2} = a$. Note that $\sum_{i=1}^{2^D} \mu_{ij}\mu_{ik} = 0$ due to symmetry, and $\mu_{ij}^2 = 1$, so:

$$\left.\frac{\partial^2 p}{\partial x_j \partial x_k}\right|_{\mathbf{x}=\mathbf{0}} = \begin{cases} 0 & \text{for } j \neq k, \\ a\sum_{i=1}^{2^D} \gamma(\gamma - 1) & \text{for } j = k. \end{cases} \tag{4.27}$$

The Hessian matrix is diagonal with the same values on the diagonal, thus there is always a minimum or maximum in the centre $\mathbf{x} = \mathbf{0}$. The critical value of the inflection is when $\gamma(\gamma - 1) = 0$, i.e. for $\gamma = 1$. Thus the critical σ and β are as follows:

$$\gamma \overset{\text{def}}{=} 1/\sigma^2 = 1 \text{ implies } \sigma = 1, \tag{4.28}$$

hence

$$\beta \overset{\text{def}}{=} \sigma/\Delta = \sigma/(1+1) = 1/2 = 0.5. \tag{4.29}$$

Thus the critical β is equal 0.5 for $D \geq 1$. Summing up, β higher than 0.5 should be used for all PHNs of any dimension.

4.4.3 Evaluation using Re-Projection Error

In addition to average log-likelihood of the camera parameters, we evaluate our camera calibration using the re-projection error at the MAP prediction. This task is carried out by placing a known object (often a checkerboard) with a set of K 3D points in the scene at a known location, and comparing the actual locations of these points in the image to those predicted by the estimated projection matrix.

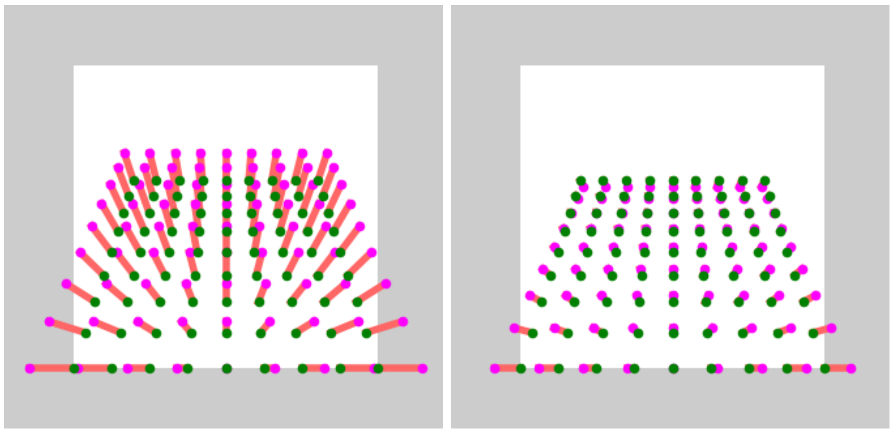


Figure 4.9: Re-projection error, where the green checkerboard is the ground-truth, the magenta one is the prediction, and the pink lines show the errors. The white background is the image frame and the grey one lies outside it. Examples of re-projection error (from left): 6.5% and 2.0% of the image width (for illustration only).

Since we know the projection matrix of both ground-truth camera, $\mathbf{P}^{gt}$, and of the predicted camera, $\mathbf{P}$, we can place a checkerboard (virtually) in the scene, namely in the front of the view with a maximum size that fits the ground-truth image. Thus we know the exact positions of the grid-points in the world coordinates, $\mathbf{W}$. We also know the projection of the grid-points in the image frame using the ground-truth camera, $\mathbf{w}^{gt} = \mathbf{P}^{gt}\mathbf{W}$ and the ones obtained using the camera predicted by PHNs, $\mathbf{w} = \mathbf{PW}$. The re-projection error is simply the RMSE of a deviation of the checkerboard grid-points in both images where $\mathbf{w}^{gt}$ and $\mathbf{W}$ are fixed, i.e.:

$$E(\mathbf{P}) = \sqrt{\frac{1}{K}\sum_{k=1}^{K} |\mathbf{w}_k - \mathbf{w}_k^{gt}|^2}. \tag{4.30}$$

We express the re-projection error in the image frame units: the RMS deviation is given in a percentage of the image frame width. Figure 4.9 shows the examples of overlaid checkerboards using ground-truth and predicted cameras.

4.4.4 Integration for Log-Likelihood Computation

To perform inference to find the MAP value of the joint posterior $p(\mathbf{z}|X)$ as in Equation 4.2 we do not need to find the normalizing constant. However, we need it to compute the log-likelihood for the purposes of evaluating the framework.

We wish to compute the normalizing constant $1/Z(X)$ for the RHS of Equation 4.2. We estimate $Z(X)$ by numerical integration of the predicted density in the $\mathbf{z}$-space.

The function was implemented in C to make the computations faster, and we experimented with integral computation using classical quadrature. We found that in the 3D case it was not converging at all in a reasonable time even for a single prediction, and it would be much too slow to evaluate all test examples. This integration method would be even slower for dimensions above 3. Therefore, we evaluate the 2D and 3D cases integral using Monte Carlo integration by sampling from the Hough space.

One can sample uniformly in Hough space, but for faster convergence we use Importance Sampling MC integration, where the auxiliary density is obtained as follows: we first evaluate the function on a hypercube centered around each Gaussian. We then define a d-dimensional histogram (normalized to sum to 1) with the weight in each cell being mean value of the function around each Gaussian centre. This leads to sampling a few orders of magnitude more frequently in the regions of a high density.

Importance Sampling (IS) results in a significant speed-up relative to uniform sampling. We conducted an additional comparison for the 2D case, where we can evaluate the ground-truth integral to a high precision using a Python method `scipy.integrate.dblquad` that executes a quadrature from the Fortran library QUADPACK. Figure 4.10 shows the relative error plotted against the number of function evaluations on a log-log scale. We can note that for 1% relative error, we need to evaluate the function only 5k times when using IS, as opposed to around 500k in the uniform approach. The slopes of the lines are close to $-\frac{1}{2}$, as is expected for Monte Carlo integration where the error decreases as $N^{-\frac{1}{2}}$, where N is the number of function evaluations. As can be seen from the plot, the error after the same number of function evaluations is one order of magnitude lower for IS over uniform, and as the error decreases with a square root of the number of function evaluations, integration is two orders of magnitude faster. This allows us to accurately estimate a single integral in a few seconds, rather than minutes.

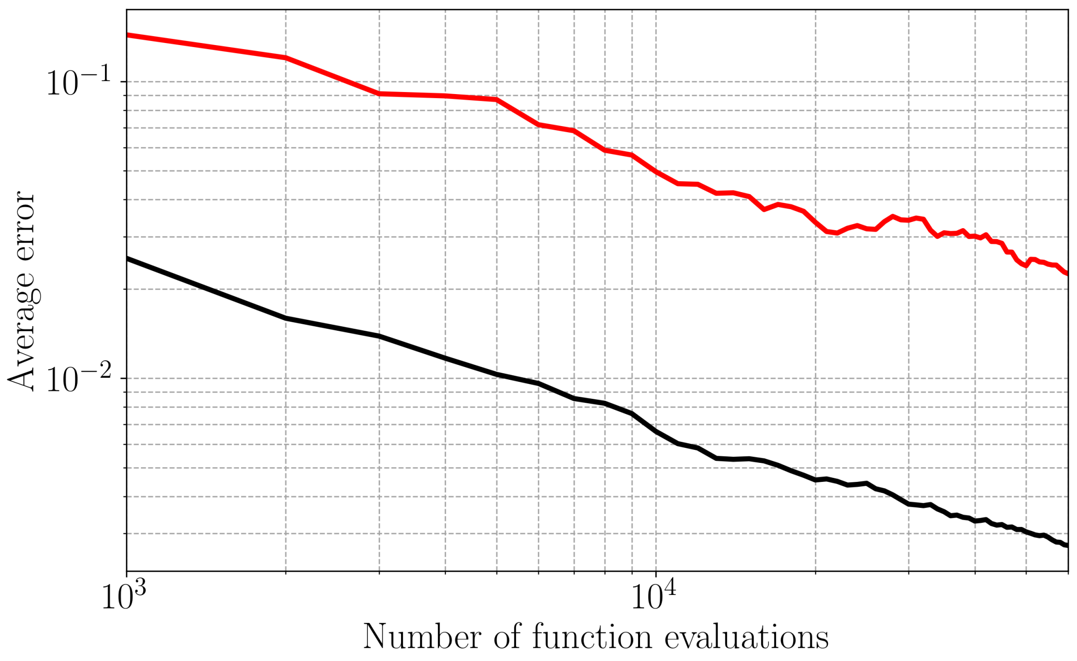


Figure 4.10: Average error over the test set using a uniform sampling (red) and IS (black) in the 2D case.

4.4.5 Neural Networks Set-up

Table 4.2 shows the network configurations and learning rates used for training. We use all 13 convolutional layers of VGG-16 as the fixed core, and the set-up is the same as in Section 3.4.2. Again, we concatenate the input to the first fully connected layer with $(x_o^{view}, y_o^{view}, s_o^{view})$ of a patch, and for the H^1 network, also with the standardized value of ω that it is conditioned on.

The networks are implemented in Lasagne/Theano, and trained using the Adam optimization algorithm (Kingma and Ba, 2015) with the learning rate of 0.001. Both the Adam optimizer and use of tanh activations appeared to be crucial for a proper convergence. When keeping $\beta = 0.6$ fixed, training finds a local optimum where only a subset of the Gaussians are activated, usually these near to a large number of training points. A better convergence was obtained when the network was pre-trained using a lower standard deviation scaling factor β (as used in eq. 4.20 and 4.21) to encourage the network to activate all Gaussian components. During the first half of the training we divide the standard deviations by two, thus we first set $\beta = 0.3$, and in the second half of the training we use $\beta = 0.6$.

Resolution (outgoing)	PHNs	
	H^1	H^2
$128 \times 128 \times 3$	RGB Image	
$16 \times 16 \times 512$	VGG-16 (all 13 conv. layers – fixed)	
$11 \times 11 \times 50$	C-50-6	C-50-6
$6 \times 6 \times 50$	C-50-6	C-50-6
$1 \times 1 \times 50$	C-50-6	C-50-6
	F-50	F-30
	F-30	F-30
	Softmax-135	Softmax-10

Table 4.2: The configurations of the PHN networks, these are 19 layers deep. Layer types are: C – Convolutional, F – Fully connected, and Softmax. The layers are described as follows: F - number of units; C - number of filters - filter size.

4.5 Results

We first provide illustrative examples in Section 4.5.1, with exemplary density plots of predictions in the Hough space. We analyse the properties of the combinations of predictions for both 2D and 3D cases.

Then we describe the set-up of the quantitative evaluation for the realistic synthetic dataset in Section 4.5.2. To this end we introduce several baselines and CNN-based comparators that make use of the same hidden layers architecture as the PHNs.

The main results are given in Section 4.5.3. We show that the PHN predictions outperform all the other approaches. Since all the methods use the same network architecture, PHNs' advantage lies in the method for combining of multiple probabilistic votes. Finally, in Section 4.5.4 we investigate in detail how the Single-PHN predictions and their mean error (per image) compare with their Multi-PHNs combination.

4.5.1 Examples

2D case Figure 4.11 shows two example scenes, for each scene we show single object predictions in the 2D Hough space, and the Multi-PHNs prediction (bottom right in each panel). Notice that in the Multi-PHNs plots the uncertainty has significantly decreased compared to single observation plots, so the model works as desired. Even in the cases of lower likelihood, the MAP is very close to the ground-truth. In the bottom example note that the outlying prediction ‘Single 1’ (due to a false positive detection of an object) does not corrupt the final result, due to the outlier model.

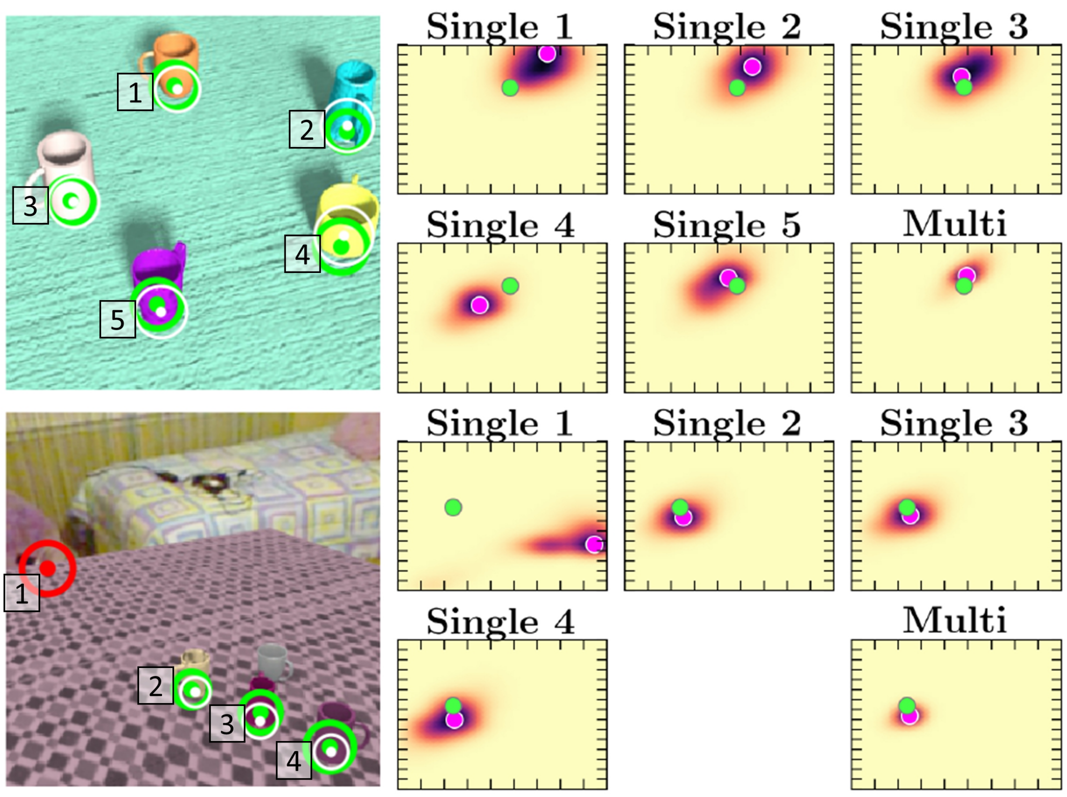


Figure 4.11: Two examples of PHNs prediction; the density sub-plots are in the (camera elevation, camera height)-space. Each density sub-plot is the prediction for a single observation, apart from the bottom-right sub-plot, which shows the joint density of the multiple observations. The ground-truth is denoted by a green circle, the MAP by a magenta circle.

3D case Figure 4.12 presents examples of prediction for the 3D case. We may notice in the plots that predictions are significantly more accurate after observing all objects. Note there is always a high density at the ground-truth. Sometimes a single camera AoV cannot be determined and a prediction is the whole or a part of the manifold (top row), then ground-truth may be farther from the MAP, but the MAP camera is still located on the manifold, so it results in a similar image, just using a different AoV and camera pose (e.g. a camera located two times higher, but with two times larger zoom). In the bottom row the object configurations more tightly constrain the posterior.

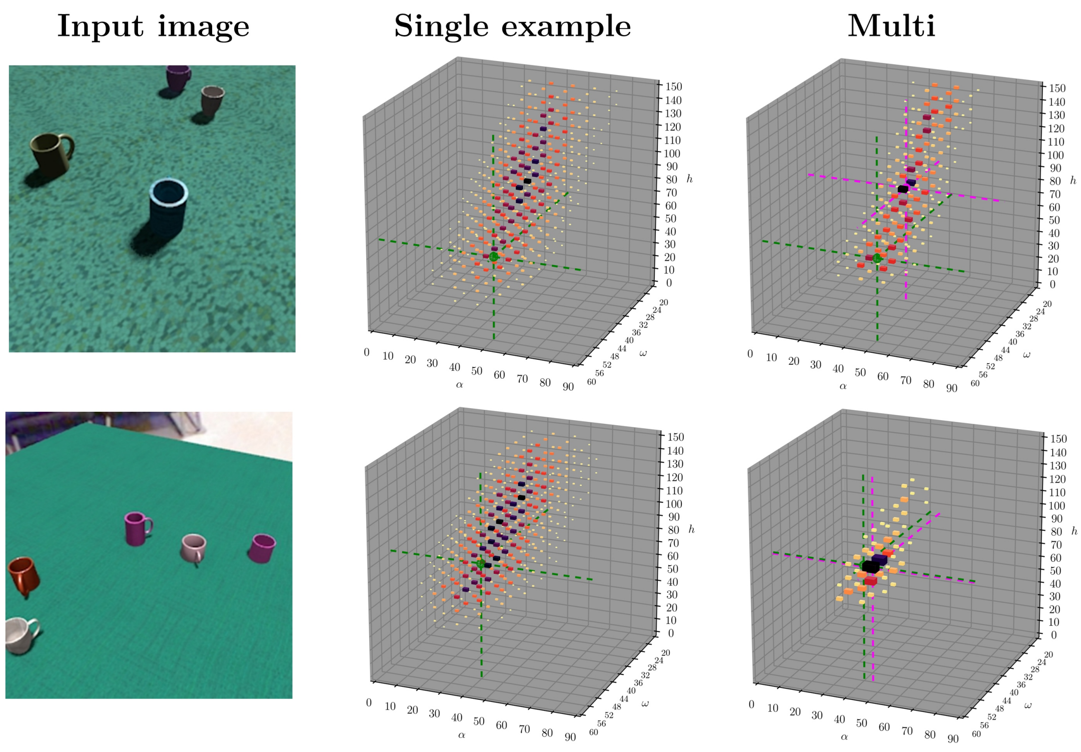


Figure 4.12: Examples of prediction. For the observed image (left), the plot in the middle shows a randomly selected example of a prediction for a single observation, the right plot shows the joint density with multiple observations. The density is represented as a 3D Hinton plot: the space is divided into voxels and each cube lies in the centre of a given voxel. The cube volume and colour-coding represents the amount of the density mass within a single voxel. The ground-truth is denoted by a green ball located at the intersection of green guiding-lines. For 'Multi', the magenta ball is the MAP.

Qualitative Examples for Real Images The examples of predictions for real images are given in Chapter 3, in Figure 3.12 and Figure 3.13, for which we considered the full 3D case set-up. Note the predicted view-points are very accurate, and include both low and high camera elevation view-points, such as the ones shown in the right column of Figure 3.13 (for instance see the first three rows).

4.5.2 Quantitative Evaluation

We use the same training dataset of 7 thousand images as used in Chapter 3. We perform a quantitative evaluation of PHNs on the same synthetic test set of two hundred images. We evaluate the quality of the predictions via the average predictive log-likelihood, and through the average re-projection error.

Baseline As a simple baseline ('Baseline') we use the prior density $p_0(\mathbf{z})$, and the mean of $\mathbf{z}$ on the training set as a point estimate.

CNN predictor To demonstrate that object-based PHNs are superior to standard CNNs, as a non-trivial comparison we use a CNN predictor which takes the whole image as input and predicts $\mathbf{z}$. This is based on the VGG-16 architecture using the same configuration of all the hidden layers as the PHN network, where there are sigmoid outputs (scaled to match the Hough space size) for each camera LV. We use the predicted values to evaluate the Re-projection error. To evaluate the log-likelihood for this CNN $p(\mathbf{z}|X)$ is modelled as a full covariance Gaussian estimated from the error residuals, robustified by including a term $(1-\gamma)p(\mathbf{z})$ as we do for PHNs to avoid paying a high penalty for outliers. We use separate CNNs and covariance matrices of error residuals for 2D and 3D cases.

Object-based CNN predictor To demonstrate that PHNs are superior to CNNs even if also applied in the per-object manner, we introduce a strong comparator that makes predictions per object and aggregates the votes using the median function. The CNNs hidden layers are again the same as for PHNs, and two separate networks are trained to regress the parameters for 2D and 3D cases, and the median of the single predictions ('Single-Regression') per variable is applied to aggregate the votes to produce a 'Multi-Regression' prediction.

For these CNNs (also robustified), $p(\mathbf{z}|X)$ is modelled as a full covariance Gaussian in 2D and in 3D. Note for Single-Regression, the error residuals are of the errors made by the Single-Regression. For Median-Regression the errors are lower, as these are obtained after the aggregation, hence with tighter covariance matrices. Figure 4.13 compares single-Regression covariance matrix (of error residuals, left column) vs Multi-Regression (right column). In both plots the Gaussian tries to cover the whole manifold, for the Median case it is narrower.

Case	Evaluation metric	Baseline	CNN	Single-Regr.	Median-Regr.	Single-PHN	Multi-PHNs	Multi better
2D	Log-likelihood	−9.51	−8.36	−7.58	−6.96	−7.65	**−6.18**	94%
	Re-projection err.	9.62	4.95	3.79	2.71	3.85	**2.41**	91%
3D	Log-likelihood	−13.20	−12.79	−12.52	−12.35	−11.33	**−10.18**	77%
	Re-projection err.	9.62	5.64	4.82	4.02	4.91	**3.30**	86%

Table 4.3: Results: average log-likelihood and re-projection error for Baseline, CNN, Single-Regr., Median-Regr., Single-PHN and Multi-PHNs. Re-projection error is given in % of the image width. The last column shows for each evaluation metric the percentage of images where PHNs predictions after observing multiple detections are better than the average of single PHN predictions.

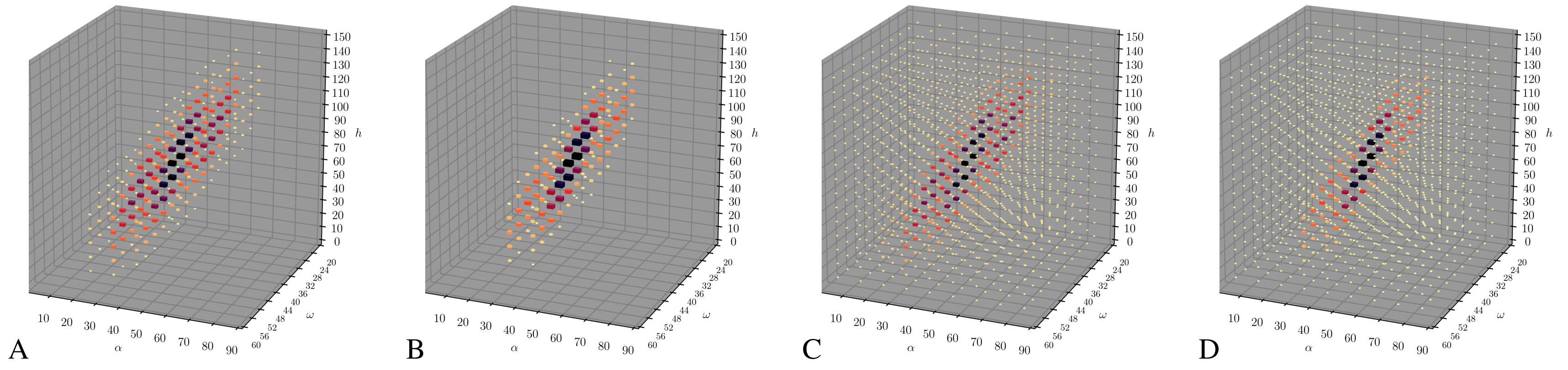


Figure 4.13: Covariance matrices for Single-Regr. (A) and Median-Regr. (B), and the robustified versions (C and D, respectively).

4.5.3 Results: Comparison of the Methods

Table 4.3 shows the quantitative results of all the compared methods. For log likelihood a higher value is better, while for re-projection error lower is better. For the Single-Regression and Single-PHN columns the results are averaged over all detections in a scene, as well as over scenes.

For both the 2D and 3D cases PHNs clearly make better predictions after observing multiple detections. The last column "Multi better" shows for each evaluation metric the percentage of images where PHN predictions after observing multiple detections are better than the average of single PHN predictions.

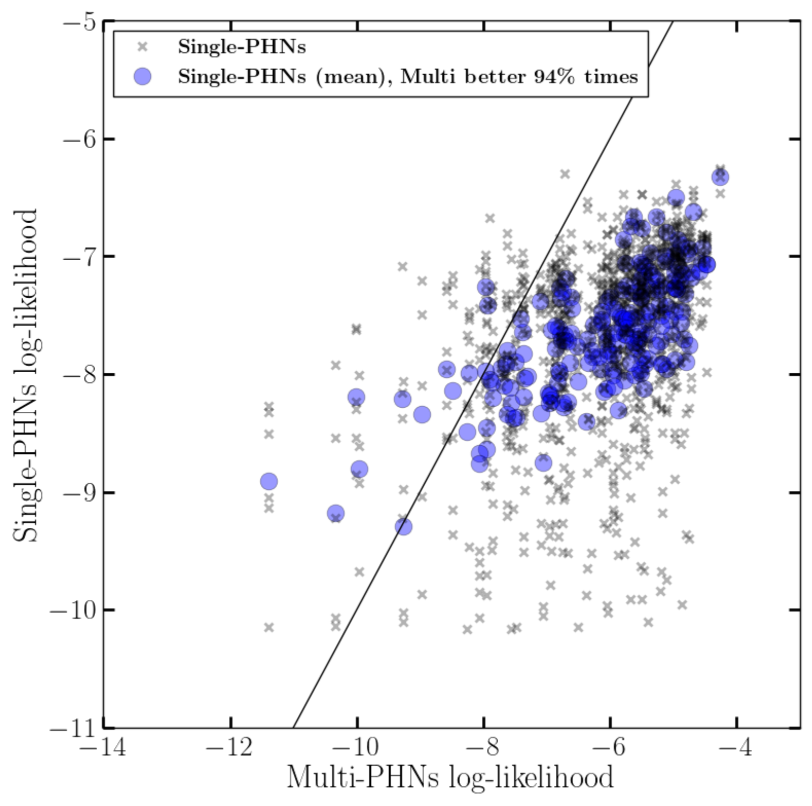
Note that the Single-PHN method that processes a single detection outperforms the CNN method that takes as input the whole image. The re-projection error (3D case) is 4.91 (Single-PHN) vs 5.64 (CNN), and is noticeably lower (3.30) given all the detections.

PHNs are also better than the strongest comparator, Median-Regression, here the re-projection error is approximately 11% lower for the 2D case, and 18% lower for the 3D case. Note Single-PHN re-projection error results are very similar to Single-Regression due to the same input and neural network architecture. Thus the PHNs' advantage lies in combining of multiple probabilistic votes. For log-likelihood, PHNs results are also better than Median-Regression for 2D case, and significantly better for the 3D case.

4.5.4 Detailed PHNs Results

Figure 4.14 (log-likelihood) and Figure 4.15 (re-projection error) give a detailed comparison of Single-PHNs vs Multi-PHNs. These are presented for both the 2D (top) and 3D (bottom) cases.

Blue dots indicate the mean Single-PHN prediction for a given image, this is to be compared with the Multi-PHN score (horizontal axis). For the log likelihood case higher is better, so when the Multi-PHN wins the blue dots lie *below* the diagonal line. For the re-projection error lower is better, so when the Multi-PHN wins the blue dots lie *above* the diagonal line. For each observed image we show all Single-PHN predictions (grey crosses) as well as their mean (blue dot) so that the variability can be seen.



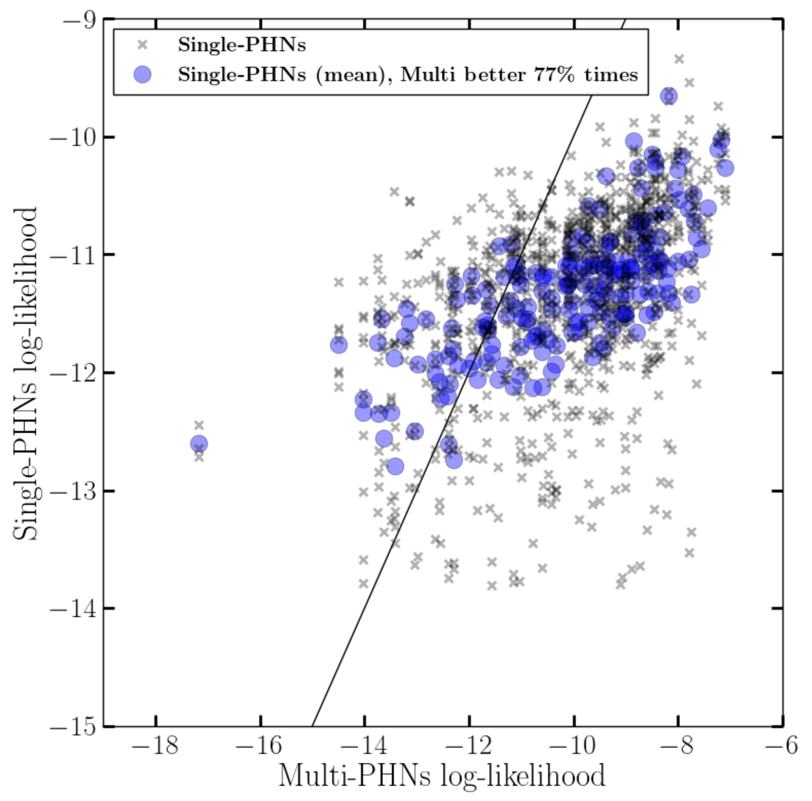
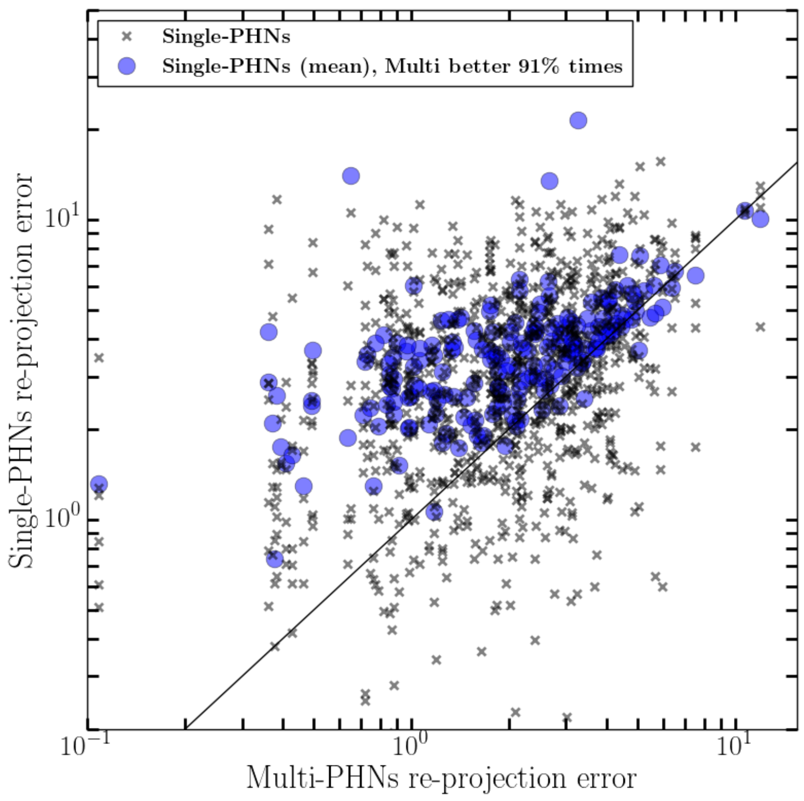


Figure 4.14: PHNs predictions for 2D case (top) and 3D case (bottom), of the log-likelihood (higher better, log-log axes). Multi-PHN wins when the blue dots lie *below* the diagonal line.



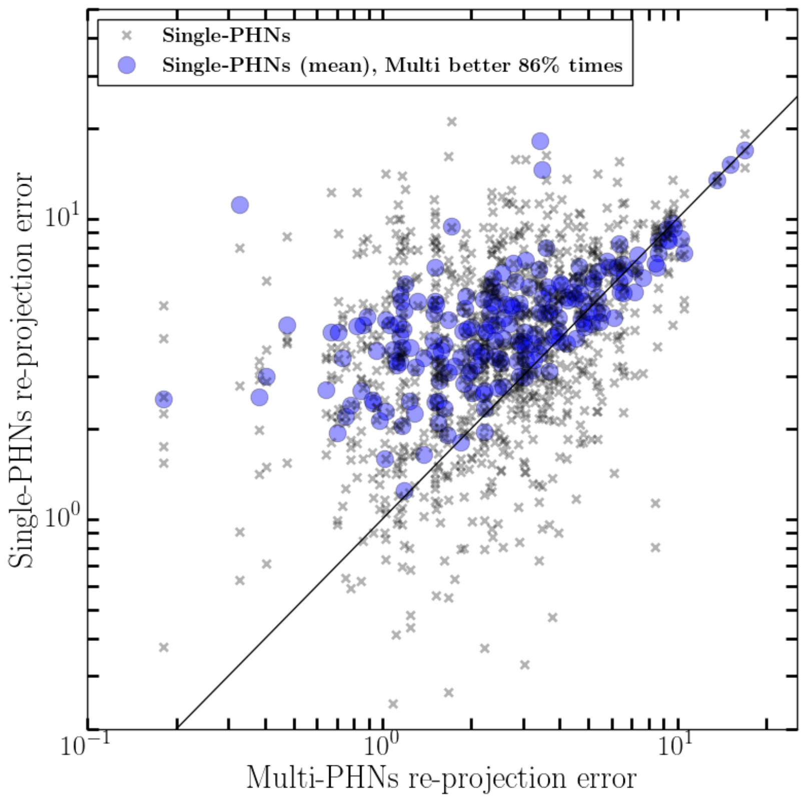


Figure 4.15: PHNs predictions for 2D case (top) and 3D case (bottom) of the re-projection error (lower better, log-log axes). Multi-PHN wins when the blue dots lie *above* the diagonal line.

4.6 Discussion

In this chapter we introduced the Probabilistic HoughNets framework for combining probabilistic votes to infer the camera parameters given the votes cast by the detected objects, both for uni-modal and multi-modal cases. Each detection provides one vote in the form of a noisy low-dimensional manifold in the Hough space, and by intersecting the votes probabilistically we reduce the uncertainty on the camera parameters. Importantly, the way in which the detections cause voting in the Hough space is learned from data. An outlier model allows for exclusion of corrupted predictions so as they do not interfere with the final result.

Probabilistic HoughNets by definition can cast multimodal votes, and we have demonstrated that the composition of these leads to an overall improvement in performance. For example, as shown in Figure 4.15, the re-projection errors of the combined predictions are usually lower than the corresponding error means of the single PHN predictions. For the uni-modal 2D case 91% of the combined predictions are better, and these are 86% for the multi-modal 3D case. The PHNs require extra computing time for the inference (computing MAP), for the GMM of 1350 components used in the experiments this is approximately 1 second per object (primarily for automatic-differentiation within BFGS hill climbing). The inference time for a whole image scales linearly with the number of the GMM components, and thus also linearly with the number of the detected objects. For typical images with 6 objects, this is approximately 6 seconds (standard CPU, single core).

In Table 4.3 we demonstrate the better performance of the PHNs compared to other methods. To this end we have performed a comparison with several CNN-based approaches. We demonstrate that the PHNs outperform the strongest competitor, Median-Regression. For the PHNs the re-projection error is approximately 11% lower for the 2D case, and 18% lower for the 3D case than for Median-Regression. Since all the methods use the same network architecture, PHNs' advantage lies in the method for combining of multiple probabilistic votes.

Chapter 5

Learning Direct Optimization for Scene Understanding

Given an initialization of a scene graph, its refinement typically involves computationally expensive and inefficient search through the latent space. Since optimization of the 3D scene corresponding to an image is a challenging task even for a few latent variables (LVs), previous work for multi-object scenes considered only refinement of the geometry, but not the appearance or illumination.

The standard way for the refinement is to measure the error E between the predicted render and the observed image, and use an optimizer to minimize the error. However, it is unknown which error measure E would be most effective for simultaneously addressing issues such as misaligned objects, occlusions, textures, etc.

To overcome these issues, we develop a framework called ‘Learning Direct Optimization’ (LiDO) for optimization of the latent variables of a multi-object scene. Instead of minimizing an error metric that compares observed image and the render, this optimization is driven by neural networks that make use of the auto-context in the form of a current scene graph and its render to predict the LV update. This allows for a smarter refinement, for example that is robust to occlusions, and can ignore the noise such as a mismatch in texture while refining the object pose. Our experiments show that LiDO generally produces better solutions in shorter time than standard optimizers, and that it is better able to handle mismatch between the data generator and the fitted scene model.

5.1 Introduction

We develop a Learning Direct Optimization method for the refinement of a latent variable model that describes input image $\mathbf{x}$. In our system we use *initialization networks* similar to the ones in Chapter 3 to predict the starting configuration of the scene graph $\mathbf{z}_0$ based on $\mathbf{x}$, which also serves as the initialization for all the compared methods.

The representation of the LVs consists of global and object LVs and is of variable dimension, i.e. $\mathbf{z} = (\mathbf{z}^{\text{Global}}, \mathbf{z}_1^{\text{Object}}, \dots, \mathbf{z}_P^{\text{Object}})$, thus initialization networks can initialize an arbitrary number of objects.

Due to the interpretable representation, one could easily edit the scene, e.g. refine object positions or their colours, or interact with objects, their properties and relations, see Figure 5.1. Note that optimization of the 3D scene corresponding to an image is a challenging task even for a few LVs.

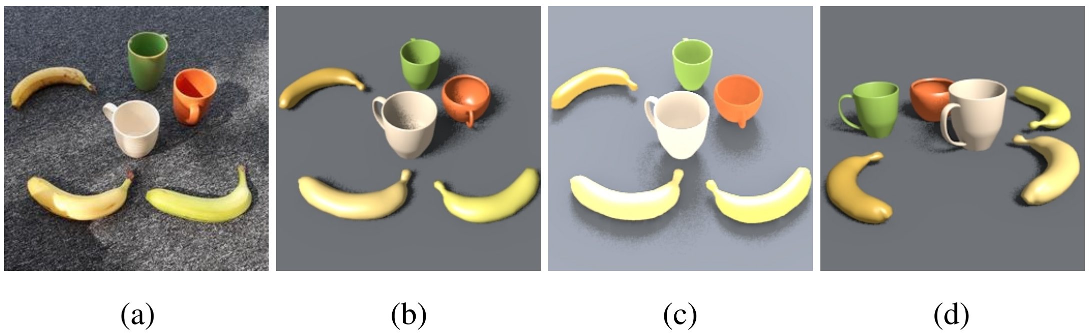


Figure 5.1: (a) Real input image, (b) the reconstructed 3D scene, (c) the scene under modified illumination and (d) from a different viewpoint.

Given a current estimate of $\mathbf{z}$ we can render a prediction of the image $\mathbf{g}(\mathbf{z})$, which can be compared to the image $\mathbf{x}$. The standard way to proceed is then to measure the error $E(\mathbf{x}, \mathbf{g}(\mathbf{z}))$ between the two, and use an optimizer to minimize the error. In contrast, the LiDO approach trains a Prediction Network to predict an update directly to correct $\mathbf{z}$, rather than minimizing the error with respect to $\mathbf{z}$. The Learning Direct Optimization approach is based on the idea that a comparison between $\mathbf{x}$ and $\mathbf{g}(\mathbf{z})$ can provide good clues as to how $\mathbf{z}$ should be updated. We can *train a network* to predict an update for $\mathbf{z}$ rather than requiring an error measure E to be defined in the image space, and then minimizing it.

The structure of this chapter is as follows: in Section 5.2 we explain LiDO and how it contrasts to error-based optimization, and provide a list of our contributions. In Section 5.3 we discuss related work. In Section 5.4 we describe the latent variables, and the initialization networks in Section 5.5. Section 5.6 gives details of the experimental datasets. Section 5.7 provides the details of the set-up of error-based optimization, and Section 5.8 gives the details of LiDO set-up and network architecture. Section 5.9 describes the evaluation measures. Finally, the results are presented in Section 5.10, with the discussion provided in Section 5.11.

5.2 Learning Direct Optimization

Figure 5.2 gives an overview of the Learning Direct Optimization framework.

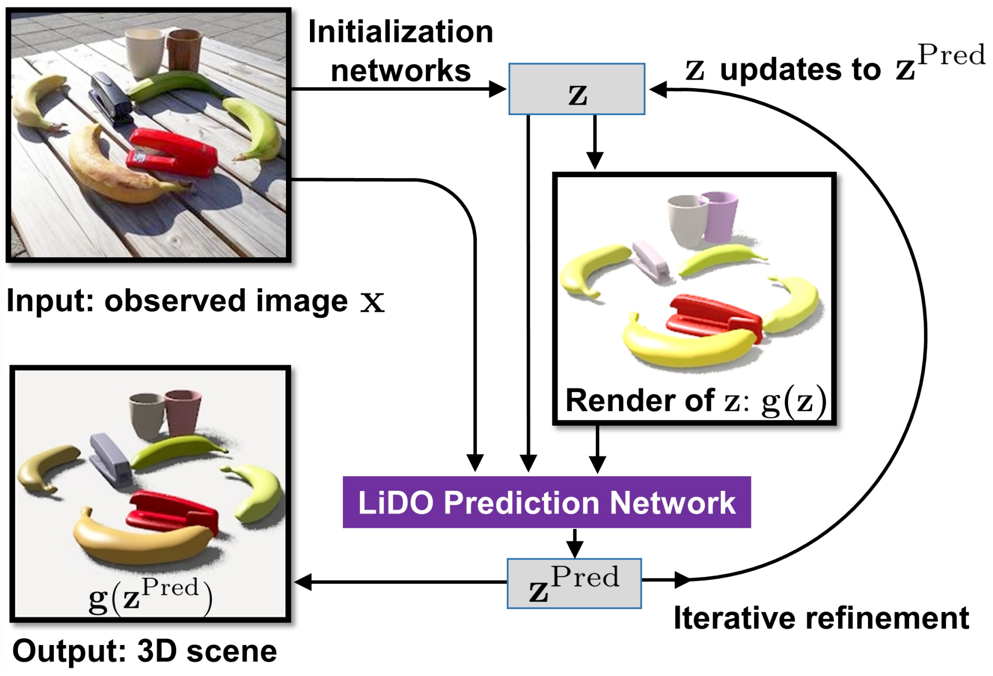

Initialization networks
$\mathbf{z}$
$\mathbf{z}$ updates to $\mathbf{z}^{\text{Pred}}$
Input: observed image $\mathbf{x}$
Render of $\mathbf{z}$: $\mathbf{g}(\mathbf{z})$
LiDO Prediction Network
$\mathbf{z}^{\text{Pred}}$
$\mathbf{g}(\mathbf{z}^{\text{Pred}})$
Iterative refinement
Output: 3D scene

Figure 5.2: Learning Direct Optimization: given the observed image $\mathbf{x}$, the initialization of the latent variables (LVs) $\mathbf{z}$ is obtained – then $\mathbf{z}$, the predicted image $\mathbf{g}(\mathbf{z})$ and the observed image $\mathbf{x}$ serve as the input to the LiDO Prediction Network. The LVs are then updated according to the prediction and a new render is produced. The LVs are then refined iteratively driven by the Prediction Network.

The LiDO method trains the *Prediction Network* on data where the current state $\mathbf{z}$ does not match the ground truth $\mathbf{z}_{GT}$. This was obtained in two ways: (i) from the initialization network where, based on $\mathbf{x}$, $\mathbf{z}_0$ and $\mathbf{g}(\mathbf{z}_0)$, one can predict $\mathbf{z}_{GT}$, and (ii) by perturbing $\mathbf{z}_{GT}$ to produce $\mathbf{z}'$, and learning to predict $\mathbf{z}_{GT}$ given $\mathbf{x}$, $\mathbf{z}'$ and $\mathbf{g}(\mathbf{z}')$. Note that the training data requirements for a Prediction Network are similar to those needed to train the initialization networks, and reuse the same data generator, so it has minimal marginal cost.

Given our initial prediction $\mathbf{z}_0$, a standard optimization would make steps based on some error E where $E(\mathbf{x}, \mathbf{g}(\mathbf{z}_t))$ measures the error between the image $\mathbf{x}$ and the current prediction $\mathbf{g}(\mathbf{z}_t)$. To perform the optimization, one can use a gradient-based optimization (GBO). However, due to the difficulties in obtaining gradients from a renderer, much work for minimizing $E(\mathbf{x}, \mathbf{g}(\mathbf{z}))$ has used gradient-free local search methods such a Simplex search, coordinate descent, genetic algorithms (Stevens and Beveridge, 2001), or the COBYLA algorithm (as used in Izadinia et al. 2017).

However, it is unknown which error measure E would be most effective for simultaneously addressing issues such as misaligned objects, occlusions, textures, etc. In contrast, Learning Direct Optimization procedure takes as input $\mathbf{x}$, $\mathbf{z}_t$, and the current prediction $\mathbf{g}(\mathbf{z}_t)$, as shown in Figure 5.2, and is trained to predict $\mathbf{z}_{t+1}^{\text{pred}}$, the true latent variables corresponding to $\mathbf{x}$, as described in Section 5.8.

The key insight is that comparison of $\mathbf{x}$ and the render $\mathbf{g}(\mathbf{z}_t)$ may yield much more information than simply the value of the error measure E. For example, if $\mathbf{x}$ contains a mug, and the prediction of $\mathbf{z}_t$ has the overall size and position of the mug correct, but the pose is incorrect so that the mug handle is predicted in the wrong place, a comparison of the two images (e.g. by subtraction) will show up a characteristic pattern of differences which can lead to a large move in $\mathbf{z}$ space. In contrast, if the handle positions are far apart, there may be no gradient information pushing $\mathbf{z}$ in the correct direction. Another problem is that the optimization may be misled when the observed image contains noisy features in a form of object textures, shadows, etc., while the LiDO Prediction Network can learn to ignore such distractions.

Our contributions are:

1. We develop a general framework for the joint refinement of all the LVs in a multi-object scene, including the shape and appearance of the objects, illumination and camera variables, without the need to choose a specific error metric E to measure of the mismatch between the input image and the predicted image.

2. We show that LiDO generally produces better solutions in shorter time and is

more stable than standard optimizers, since the $\mathbf{z}$-update directly targets the optimal $\mathbf{z}$ rather than simply moving downhill.

3. We show that LiDO is better able to handle mismatch[1] between the data generator and the fitted scene model, in terms of synthetic vs. real images mismatch, object shape mismatch, and mismatch due to the texture and other nuisance variables.

5.3 Related Work – Refinement via VIG

The shape and appearance refinement is a long-standing problem, see e.g. Cootes et al. (2001) for refinement of a model defined in pixel space. For 3D models, the standard practice for refinement of the initialized scene graph is to minimize some error function E, where the reconstruction loss is based on a summary statistics of the image pixels. Thus refinement can be carried out by sampling from or optimization of the posterior on $\mathbf{z}$. The error could, for example, measure the discrepancy in pixel space, or in some other feature space like the representation obtained in higher layers of a neural network (see e.g. Kulkarni et al. 2015c). In contrast, LiDO directly predicts updates for all different kinds of LVs together, without the need to chose a specific error metric E.

Refinement within the VIG paradigm has been considered by several works focusing on the specific problem of face explanation. This is a significantly simpler problem for optimization than multi-object scene explanation, as the face (a single object) is always present in the centre of the image and can be modelled by a single deformable mesh. Yildirim et al. (2015) use a pixel-based error measure when sampling parameters of a 3D face model; and Schönborn et al. (2017) develop a sampling procedure over the probabilistic parameters of the face shape and appearance model. Hu et al. (2017) split the problem into simpler sub-tasks and sequentially optimize the pose, shape, light and texture parameters.

For multiple objects the problem is much more challenging, even in 2D, as illustrated by Jampani et al. (2015) who compare various compute-intensive MCMC methods for the relatively simple problem of fitting multiple colourful 2D squares. For 3D

[1]"mismatch": the observed data at the training time is different or simpler than the observations at the test time.

objects, most of the methods match only the 3D geometry to the image, and this is also achieved via slow sampling procedures. For instance Satkin et al. (2015) match 3D CAD models based on a set of various similarity measures calculated using predicted surface normals, detected edges, rendered object mask etc.; the IM2CAD method by Izadinia et al. (2017) aligns object shapes to the observed image by minimizing the distance in the VGG feature space; and Zou et al. (2019) optimize object poses to fit to the depth channel input by enumerating a large number of object shapes and poses. Note these methods optimize only the geometry, and none of them model the appearance or illumination parameters (for visualizations, objects are given a colour in a post-processing step).

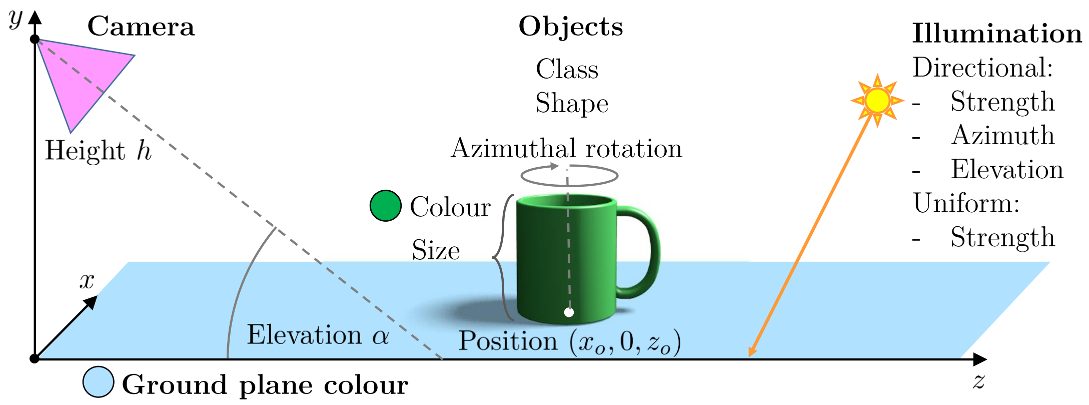
Finally, some works consider toy synthetic scenes, but these approaches have only been demonstrated for scenes containing known objects with fixed sizes and appearance (e.g. Eslami et al. (2016) who consider three objects of a fixed colour, and Wu et al. (2017a) who consider scenes with objects from the Minecraft game), and hence have not demonstrated applicability to real images.

LiDO not only fits shapes and poses to the image, but makes use of a whole render of a reconstructed 3D scene, and refines all scene graph LVs jointly. This idea of making use of the current render of the scene, falls into the area of “auto context”, which relates to feeding the output of a learning machine to the input to improve results and make use of context information. This was studied e.g. by Tu and Bai (2009) who used a mask of current pixel labelling to help to improve the segmentation. The recent work by Manhardt et al. (2018) and the DeepIM method (Li et al., 2018b) describe a special case of LiDO as applied to object 6D pose estimation (3D translation plus 3D rotation). In these papers a neural network makes iterative updates to the pose parameters, based on the input image and a render of the current estimate, but only for a known, specific object of a fixed size. In addition, LiDO can handle novel instances at test time, allowing for variable object size, shape, and texture.

5.4 Latent Variables

Our work below considers high resolution scenes with a number of objects (from a known set of object classes) on a ground plane (table-top). The rendered objects and the ground plane have random noisy textures, and the background is a real indoor image. See Figure 5.3 for the diagram of the latent variables and example images.

The object LVs are as follows: for each object o its associated LVs are its class c_o, position $(x_o, 0, z_o)$ on the ground plane, size s_o, angle of azimuthal rotation ϕ_o, shape (1-of-K encoding) and colour (RGB).



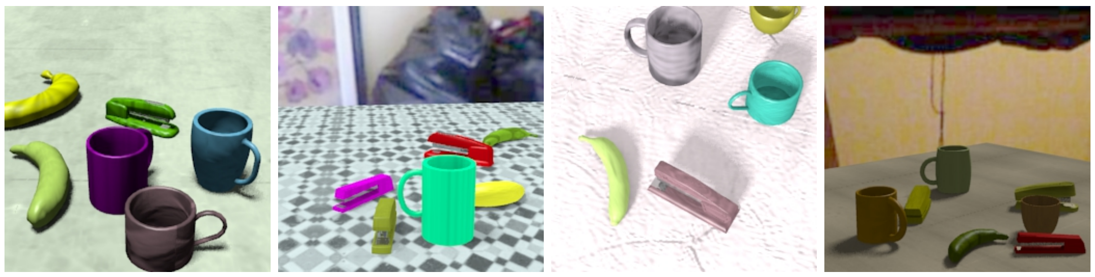


Figure 5.3: **Top**: Diagram of the latent variables; **bottom**: examples from the Synthetic dataset, featuring a variablity in the objects present, their poses, appearance, as well as variable illumination and viewpoints.

The global LVs are as follows: ground plane RGB colour, camera LVs and illumination LVs. The camera is taken to be at height $y = h$ above the origin of the (x, z) plane, and to be looking at the ground plane with angle of elevation α, with fixed camera intrinsic parameters. The illumination model is uniform lighting (LV: strength) plus a directional source (LVs: strength, azimuth and elevation of the source).

The 1-out-of-K object shape encoding is a simple yet effective baseline. As the predictions are made *per detected object and per object class*, one could extend this to use e.g. shape and texture morphable models like Blanz and Vetter (2003) or later work. However, note that the contribution of our system is demonstrating strong performance on optimizing *multiple objects* in a complex scene (plus camera, illumination), not just one object.

5.5 Initialization Networks

Our approach makes use of the initialization networks to obtain $\mathbf{z}_0$. Existing methods for initialization were unsuitable because these methods do not predict several of the LVs that we consider, such as object colours, object contact points and illumination. Therefore, we develop our own initialization networks.

The steps in obtaining an initial scene description $\mathbf{z}_0$ are:

1. Detect objects: class, contact point[2] and size; extract 128×128 pixels image windows $\mathbf{P}_0^{\mathbf{x}}$ from input image $\mathbf{x}$ at the contact points.
2. For each image window $\mathbf{P}_{0;p}^{\mathbf{x}}$ with $p = 1, \ldots P$ predict global LVs and object LVs: $\mathbf{z}_{0;p} = (\mathbf{z}_{0;p}^{\text{Global}}, \mathbf{z}_{0;p}^{\text{Object}})$ – global LVs are predicted by each object.
3. Aggregate votes for global LVs to obtain: $\mathbf{z}_0 = (\mathbf{z}_0^{\text{Global}}, \mathbf{z}_{0;1}^{\text{Object}}, \ldots, \mathbf{z}_{0;P}^{\text{Object}})$.

The above steps make use of the detector and LV initialization networks described below, for each image windows are of size 128×128 pixels. All the convolutional networks are trained on top of all the 13 convolutional layers of VGG-16 network (Simonyan and Zisserman, 2015), so as to afford transfer to work on real images.

Object detector: The detector is trained to predict whether a particular object class is present at a given location, together with object size. Trained object detectors are run over the input image to produce a set of detections, which are then sparsified using non-maximum suppression (NMS).

The detector is trained on 30,000 positive image windows with object contact point centred (with small noise of ± 8 pixels added), and 90,000 negative images: 30,000 random image windows, 30,000 image windows with the centre nearby the contact point of other objects, and 30,000 random crops from the ImageNet dataset (Russakovsky et al., 2015). The detector is run on 10,000 images to produce the training dataset for the initialization networks. Afterwards, we apply the LV initialization networks on another 10,000 images to produce the dataset for LiDO (the first source).

LV initialization networks: We extract an image window centred at each object detection, and use this to predict the ground truth latent variables. The networks are

[2] To recall, the contact point is the origin of the object at which it is placed on the ground plane, around which the azimuthal rotation is specified.

applied individually to each detected object window. All the objects predict their own LVs as well as all global LVs. All the global LVs are trained/predicted per object, then combined; for robustness, the aggregation is done using the median function [3].

The outputs of the above stages are assembled into a scene graph. The contact points of the detected objects are back-projected into a 3D scene given the predicted camera to obtain the 3D positions. The object scaling factors are obtained from the predicted object size and the actual distance from the camera to the object after back-projection.

5.5.1 CNN Architectures of Detector and LV Initialization Networks

Table 5.1 shows the network configurations and learning rates used for training. We use all 13 convolutional layers of VGG-16 as the core on 128×128 pixel input. We use only the first three pooling layers, and the VGG weights are kept fixed. Since the original pixel values are integers in $[0,255]$, while the VGG expects zero mean pixel intensity, we subtract the mean. The region outside the image frame is given as value 0.

VGG layer activations are *ReLU*, layers on top of VGG use *tanh* activations. We do not use padding in our VGG layers. The fully connected layers of the detector networks are implemented as filter 1×1 convolutional layers, so they can be efficiently applied in a sliding window manner.

For azimuthal rotation, we predict the rotation discretized into 18 bins of 20 degrees (the LiDO network predicts the updates in a similar manner, discretized into smaller, 1 degree bins). This allows to make multimodal predictions and thus the networks can handle the symmetries. For example for a cube object, the network should predict 4 bins every 90-degrees with similar probability, initialize at one of these, and then refine the remaining misalignment.

The implementation is in Python (Theano) and we use the Adam (Kingma and Ba, 2015) optimizer with L2 or categorical cross-entropy loss to train the networks. Each

[3] Note here for all the global LVs we use the median function to aggregate the votes, also for the camera LVs for which we previously used PHNs in Chapter 4. This is because we no longer are interested in a one-shot compute intensive inference, but instead consider an optimization task with an auxiliary renderer that provides auto-context feedback. As shown in Table 4.3, in the “2D re-projection error” row, the Median-Regression method is only slightly worse (error: 2.71 vs 2.41) for the unimodal case that we consider here, and with more iterations, the refined camera can be more accurate. Therefore, we use the much faster median aggregation method for all the global LVs, which we apply to produce a large dataset of the initialization mistakes being made. We can also very efficiently calculate the global LVs multiple times during all the iteration steps of the refinement.

Detector	
Class	Size
Input $128 \times 128 \times 3$ (Image)	
VGG-16 (all 13 convolutional layers)	
$3 \times$ C-50-6 *(separate per network)*	
Fd-200	Fd-200
Softmax-4	Sigm-1

LV Initialization Networks			
Shape	Azimuth (Ob/Lighting)	Lighting	Camera
Input $128 \times 128 \times 3$ (Image)			
VGG-16 (all 13 convolutional layers)			
$3 \times$ C-50-6 *(separate per network)*			
Fd-50	Fd-50	Fd-100	Fd-50
Softmax-6/15/8	Softmax-18	Sigm-3	Sigm-2

Learning rates					
Class	Size	Shape	Azimuth	Lighting	Camera
0.001	0.0002	0.001	0.0003	0.0001	0.0001

Table 5.1: The configurations of the detector (top) and LV initialization networks (middle), and the learning rates (bottom). Layer description (where N denotes the number of units and K the filter size), is as follows: 1) *Convolutional layer*: C-N-K; 2) *Fully connected layer, with its input concatenated with the detector output (position of the detection plus object size)*: Fd-N; 3) *Sigmoid (fully connected) layer*: Sigm-N; 4) *Softmax (fully connected) layer*: Softmax-N. Colour networks (for objects and for ground plane) are simple 3-layer CNNs with leaky rectify activations: Input, C-27-6 (stride 6, dropout $p = 0.5$), Fd-40, F-40, Sigm-3; trained with 0.0001 learning rate.

LV belongs to a specific LV-set responsible for a given property, and we train one LV initialization network per global LV-set and one LV initialization network per each object class (stapler, mug, banana) for object LVs. We use dropout in the detector networks with $p = 0.5$ in all the 3 convolutional layers (on top of VGG ones), and after the first one for the initialization networks.

5.6 Stochastic Scene Generator and Experimental Datasets

This section present the details of the Stochastic Scene Generator (Section 5.6.1), the training and test datasets (Sections 5.6.2, 5.6.3), and the quality of the initialization (Section 5.6.4).

5.6.1 Stochastic Scene Generator

To obtain a suitable dataset, we use our Stochastic Scene Generator as described in Section 3.2, extended to handle multiple object classes. The scene generation procedure makes use of adjusted parameters ranges and an improved method for sampling the colours. We outline below the details of the new dataset.

For each image we sample the global LVs and the LVs of up to 7 objects which lie on the ground plane. We consider three object classes: mugs, bananas and staplers. For each object we sample its class, shape (one-of-K = 6/15/8 shapes respectively[4]), colour, rotation, and scaling factor. Since our ultimate goal is understanding of real images, the synthetic images are generated with a rich realistic Blender[5] renderer, where we have added shadows, realistic backgrounds and textures on the objects. The textures serve as a noise to allow LiDO to work with richer real images that may feature different kinds of surface patterns, shadows and noise. Since we do not model the textures, the mean effect of texture is absorbed into the ground truth (GT) colour. We define the GT colour to be the one that the best matches the texture, i.e. the mean colour of the

[4]The shapes were obtained from ShapeNet: `https://www.shapenet.org/`; and then aligned in 3D to have the same position, size, and rotation.

[5]`https://www.blender.org/`

texture. For example for the white object in black dots, the ground truth colour will be light-grey.

We sample the camera height and elevation uniformly in the appropriate ranges: $\alpha \in [0^\circ, 75^\circ]$, $h \in [5, 75]$ cm, the angle of view is fixed and set to a typical value of 60°. Illumination is represented as uniform lighting plus a directional source, with the strength of the uniform light $\in [0, 1]$, the strength of the directional light $\in [0, 2]$, with azimuth $\in [0^\circ, 360^\circ]$ and elevation $\in [0^\circ, 90^\circ]$ of the directional light. For each object we sample its class (stapler, mug or banana), shape, colour, rotation, and scaling factor so that stapler length is $\in [12, 16]$cm, mug diameter in $\in [7, 10]$cm, banana length $\in [15, 20]$cm.

Below we describe the process of sampling realistic colours for our scenes. Initially we experimented with sampling from a uniform distribution but it often results in pastel colours, close to gray. Therefore we use a collection of 17 predefined CSS/HTML colours and sample a pair of them with a random mixing proportion. This samples a variety of colours with frequent strong colours (where the RGB value is either close to 0 or 1), as these are common choices for everyday objects. Afterwards, we add a uniform noise of ± 0.2 to the RGB coefficients and clip if necessary. We use this scheme to sample colours of staplers and mugs, for bananas we fix one of the components to be yellow, for ground plane colours we fix one of the components to be white so as to obtain bright colours more frequently.

Background images are taken from the NYU Depth V2 dataset (Silberman et al., 2012). In addition random textures are applied to objects by converting a set of textures to greyscale and applying them at a random scaling on the surface via multiplication of the initial colour and the texture intensity.

5.6.2 Training Datasets

We train the initialization networks on a dataset of 10,000 images with over 55,000 objects. To train the Prediction Network we use data from two sources. The first (another 10,000 images and over 55,000 objects) is obtained from the $\mathbf{z}_0$ outputs of the initialization networks. We paired the detected and GT objects based on the distance of the object contact points to the closest one of the same class within the radius of 10% of the image width (15% for real images since manual annotations are more noisy than the perfect synthetic ones). The second source was a dataset generated by adding a small amount of noise to 10,000 GT images (over 55,000 objects) to allow LiDO to deal with small errors in further iterations. The noise was uniform for the continuous LVs: $\pm$ the median error made by the initialization networks per LV. We also replaced each GT CAD shape by a random one to train LiDO to work well in the case of shape-mismatch. Thus, the LiDO training dataset consisted of 110,000 object examples.

5.6.3 Test Datasets

For all the neural networks we used train-validation-test splits of the synthetic dataset, using separate scenes for the Initialization Networks and LiDO Prediction Network. Furthermore, we used a separate validation set for optimization tasks to choose the hyper-parameters of the LiDO and baseline methods, plus a final optimization test set to evaluate them (each of 200 images, with over 1k objects).

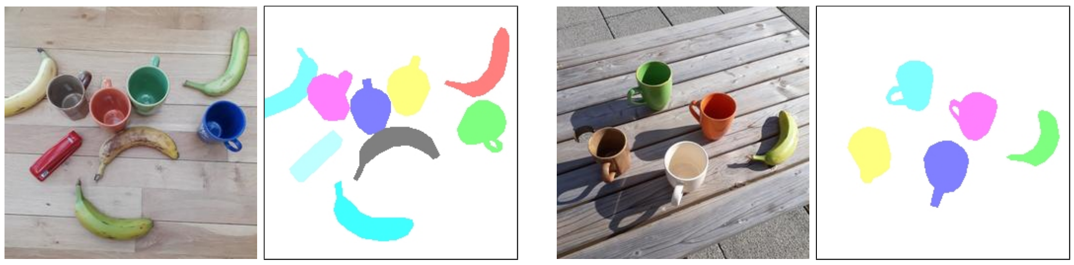


Figure 5.4: Two examples from the Real dataset (images and instance segmentation masks).

As our aim is to understand real images, we apply the same methods to a dataset consisting of 135 real images with over 750 objects total. The manual annotations are used only for the evaluation, not for making the predictions. The real images were annotated with object masks and contact points to allow quantitative testing of the methods' performance, as the quantitative evaluation here can be done only in the pixel space. We captured the real images to feature a number of objects of the considered classes at a variety of lighting, viewpoint and object configuration conditions, see Figure 5.4. For each object we annotated its class, instance mask, and the contact point using LabelMe software (Russell et al., 2008). Our system renders objects on an infinite ground plane. Since the ground plane is finite for real images, we use a GT ground plane mask that is defined as the ground plane up to a horizontal line located at the contact point of the farthest GT object. For real images we annotated the GT ground plane mask, since sometimes the mask might not be the full plane.

5.6.4 Initialization Network Performance for the Test Datasets

For the three-class synthetic dataset, objects were accurately detected with 94.6% precision, 94.0% recall (94% of objects are detected, 94.6% of all detections are correct), and for these 57% of the object shapes were predicted correctly. For real images, the results were: 97.8% precision, 94.0% recall, showing that the initialization networks worked similarly well for real images. All the methods that we compare start from the same initialization with the same set of the detected objects, and unpaired objects are treated as false positives/negatives. The set of the instantiated objects and their shapes are kept fixed, as these are discrete variables which are not changed during optimization with the above methods.

5.7 Experimental Set-up of Error-Based Optimization

We compare LiDO to two optimization methods: gradient-based optimization (GBO) with the best performing optimizer, and the most effective gradient-free method which was Simplex search (denoted Simp). Note that these baselines were selected as the best-performing methods out of many tested. Note other VIG frameworks (IM2CAD, 3DParsing) do not optimize the appearance and illumination and are not applicable to this domain. Since these are standard search methods which require a large number of function evaluations, we use a very fast OpenGL renderer. LiDO was also configured to use OpenGL as the internal renderer during optimization.

For error-based optimization, we compute the match between the actual and rendered image pixels (RGB intensities being between 0 and 1) using a robustified Gaussian likelihood model with standard deviation $\sigma = 0.1$ and inlier probability $\alpha = 0.8$, as in Moreno et al. (2016, eq. 3). The observed image $\mathbf{I}^O$ and the rendered image $\mathbf{I}^R(\mathbf{x})$ have P pixels and are represented as a vector of a length $3P$, with P values for each RGB colour channel. Then, for each pixel-channel i, $p(\mathbf{I}_i^O|\mathbf{z})$ is given by:

$$p(\mathbf{I}_i^O|\mathbf{z}) = \alpha\mathcal{N}(\mathbf{I}_i^O;\mathbf{I}_i^R(\mathbf{z}),\sigma^2) + (1-\alpha)\,\mathcal{U}(\mathbf{I}_i^O). \tag{5.1}$$

For GBO we use a differentiable OpenGL renderer[6] based on OpenDR: Differentiable Renderer (Loper and Black, 2014), extended to simplify rendering multiple objects. The approximate derivatives of the likelihood computed by OpenDR are fed to an optimizer. To facilitate refinement we use anti-aliasing with 8 samples per pixel to make the gradients more accurate and the likelihood function smoother.

We performed experiments with several optimizers and most of them converged poorly (e.g. L-BFGS-B). We found Truncated Newton Conjugate-Gradient (TNC) to considerably outperform other gradient-based methods, with the Nonlinear Conjugate Gradient optimizer[7] being the only other one that usually converged well (yet worse than TNC, so we use TNC for GBO).

For gradient-free methods, we found the Simplex (Nelder-Mead) optimizer worked well and significantly better than COBYLA, Simplex also often performed better and faster than GBO.

Setting proper bounds is crucial for proper optimization as the stepsize is scaled by the distance between LV boundaries, hence for all the LVs we set the bounds to the respective ranges as used in the scene generator.

Following the work of Moreno et al. (2016), we fit subsets of the search variables sequentially. The LVs are fit in the following order: ground plane colour, object colours (each object separately), object poses (each object separately), illumination and the camera. We experimented with fitting each object's LVs together, all the object LVs together, and also all LVs together, but it worked a lot less well and overall slower, because the number of variables is larger and likely the optimization landscape is thus more complex.

5.8 Experimental Set-up of Learning Direct Optimization

We ran the initialization networks on the synthetic dataset and took object detections and the associated errors as the new dataset for training LiDO. For an image $\mathbf{x}$ with ground truth $\mathbf{z}_{GT}$ we obtained a set of image windows $\mathbf{P}_0^{\mathbf{x}}$ extracted at the object de-

[6]https://github.com/polmorenoc/inversegraphics
[7]http://learning.eng.cam.ac.uk/carl/code/minimize

tections and the corresponding rendered image windows $\mathbf{P}_0^R$ from the render $\mathbf{g}(\mathbf{z}_0)$. From each pair of image windows $\mathbf{P}_{0;p}^{\mathbf{x}}$ and $\mathbf{P}_{0;p}^R$, $p = 1, \ldots, P$ and the corresponding $\mathbf{z}_0$ the *Prediction Network* CNN is trained to predict the object-specific GT variables $\mathbf{z}_{GT;p}^{\text{Object}}$ and the global GT variables $\mathbf{z}_{GT}^{\text{Global}}$. Example errors are that an object could be larger, the camera located higher.

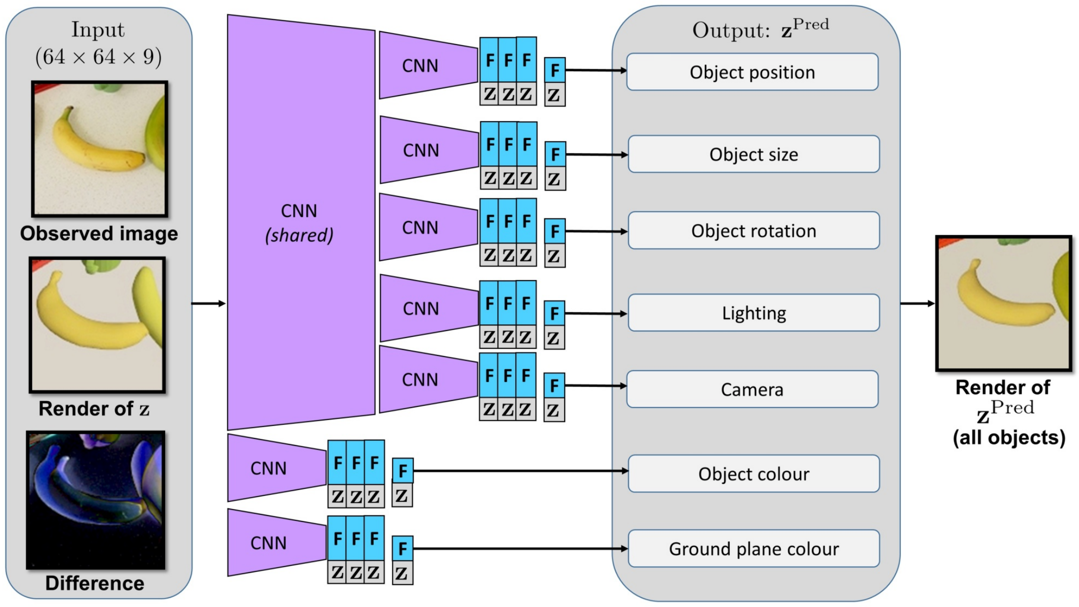


Figure 5.5: Architecture of the LiDO Prediction Network. Image input is processed by CNNs per each detected object, this is followed by fully connected layers, denoted by "**F**". In addition each fully connected layer takes as input "**z**" – the current estimate of the LVs. Each sub-network predicts the new configuration of each LV-set. The predicted LVs can be then used to render an updated scene.

Figure 5.5 shows the diagram of the neural network architecture. The Prediction Network takes as input the current estimate of $\mathbf{z}_p$, and images: both the observed and the rendered image windows (128×128 pixels, down-sampled to 64 by 64 resolution), plus their difference[8]. The RGB channels of the three images (observed, rendered, difference) are stacked together giving an input size of $64 \times 64 \times 9$. This image input is followed by a number of convolutional layers shared across all the LVs. Shared layers are followed by LV-set specific convolutional layers, and finally a few fully-connected layers, each concatenated with the current estimate of $\mathbf{z}_p$. For each object window, the current estimate of $\mathbf{z}_p$ (standardized) input consists of: object LVs (discrete class and

[8]Initially we performed experiments where the input did not include the image difference, and the network was able to learn appropriate filters. However, providing the difference led to a much faster learning, and eventually converged to a slightly more accurate configuration.

shape one-hot-encoded), global LVs (as predicted by the object, denoted G_O), global LVs (as used in the render after voting of all the objects, denoted G_V), plus their difference $G_O - G_V$. All the LV values (current and outputted $\mathbf{z}$) are standardized across the dataset. The whole network for all the LVs is trained together.

Table 5.2 shows the detailed network configurations used for LiDO. The implementation is in Python (TensorFlow). Again, the region outside the image frame is given as value 0. The [0,255] image dataset values had their mean subtracted and were divided by 100. To allow convergence, the object pose is trained in the object's current coordinates/frame (to predict the change in object position and rotation from the current value).

CNNs are trained together for all the LVs. We use an L1 loss (mean absolute error) plus L2 loss (mean squared error) for all the continuous LVs (indexed by i), and categorical cross-entropy for discrete ones (indexed by d, class indexed by c). We use the L1 loss in addition to the L2 loss, as when making predictions multiple times during refinement small errors aggregate, and L1 appropriately punishes small errors during training. To calculate the overall network loss we sum up each (per each LV) L1+L2 loss and the cross-entropy loss (for which we used a scaling factor $s = 0.2$):

$$Loss = \sum_i (|z_i^{GT} - z_i| + (z_i^{GT} - z_i)^2) - s \sum_{d,c} [z_d^c = z_d^{GT}] \ln p(z_d^c) \tag{5.2}$$

We use the Adam optimizer with learning rate 0.0003. We trained the Prediction Network for 20 epochs, this took 15 minutes/epoch on a single GPU. Note that for LiDO dataset we reuse the same scene generator, and that training time is no more than the training time of the initialization networks.

Afterwards, we run the refinement for T iterations as summarized below (every iteration extracts a new set of the image windows, $\mathbf{P^x}$ and $\mathbf{P}^R$, taken from the input image and the current render at the current estimate of the object contact points):

```
for $t \in 0..(T-1)$:
    1. Take image windows $\mathbf{P}_t^{\mathbf{x}}, \mathbf{P}_t^R$ from the input image $\mathbf{x}$ and the render $\mathbf{g}(\mathbf{z}_t)$
    2. Predict $\mathbf{z}_{t+1;p}^{\text{pred}}$ given each $\mathbf{P}_{t;p}^{\mathbf{x}}, \mathbf{P}_{t;p}^R, \mathbf{z}_{t;p}$ using the Prediction Network
       (clip if outside of the range, e.g. colour not in [0,1]).
    3. Update $\mathbf{z}_{t+1;p} = \mathbf{z}_{t;p} + \mu_t(\mathbf{z}_{t+1;p}^{\text{pred}} - \mathbf{z}_{t;p})$.
```

4. Aggregate global LVs to produce $\mathbf{z}_{t+1}$ (in the same manner as in Section 5.5).

Setting the step size $\mu_t = 1$ would move from $\mathbf{z}_t$ to $\mathbf{z}_{t+1}^{\text{pred}}$, but we have found that in the case of multiple updates, using a $\mu_t < 1$ which decreases with t leads to better performance than keeping μ_t fixed. We set $\mu_t = 1/(t+a)$. The hyper-parameter $a = 2$ was selected using the validation set split. In the experiments below we run the LiDO iteration for a fixed number of $T = 30$ steps to show the convergence curve, but it would be easy to use a termination condition $|\mathbf{z}_{t+1} - \mathbf{z}_t| < \varepsilon$.

LiDO Prediction Network					
Position	Size	Azimuth	Lighting	Camera	Colour (Ob/Gr)
Input 64 × 64 × 9 (2 images plus their difference, stacked)					
C-32-3 *(shared)*					C-32-3 (stride 2)
C*-64-3 *(shared)*					C-64-3 (stride 2)
MaxPool-2 *(shared)*					
C*-128-3 *(shared)*					C*-128-3 (stride 2)
MaxPool-2 *(shared)*					
C*-64-3 *(separate per network)*					
MaxPool-2 *(separate per network)*					
C-32-3 *(separate per network)*					
3 × Fz-40 *(separate per network)*					3 × Fz-40
Fz-2	Fz-1	Softm.(Fz-360)	Fz-5	Fz-2	Fz-3

Table 5.2: The configurations of the LiDO Prediction Network, the whole network is trained together. All the LV values (current and outputted $\mathbf{z}$) are standardized across the dataset. Layer description (where N denotes the number of units and K the filter size), is as follows: 1) *Convolutional layer*: C-N-K; 2) *Fully connected layer: F-N*; 3) *Fully connected layer, with its input concatenated with the current LVs* $\mathbf{z}$: Fz-N; 4) *Max-pooling layer*: MaxPool-K; 5) *Softmax output layer on top of a linear layer X*: Softm.(X). Non-colour networks: the first 5 layers are shared across all the 5 sub-networks, all the sub-networks on top of them have the same set-up. Colour networks are simpler and have fewer layers, and use *ReLU* activations, while non-colour networks use *tanh* activations. For lighting we predict: uniform component strength, directional component: strength, elevation, sin(azimuth), cos(azimuth). We use dropout with $p = 0.5$ after the convolutional layers denoted with *.

When producing the OpenGL renders for the LiDO prediction, we needed to take care because LiDO has been trained to predict $\mathbf{z}$'s that specify a scene for the Blender renderer. While the geometry LVs (objects present, their shape/poses, camera etc.) are common for both the renderers, the optimal colours in OpenGL differ in brightness to Blender, since the OpenGL renderer cannot produce shadows, and has to explain shadowing (e.g. that mugs are dark inside when the light comes from the side) with a lower colour brightness. To render an OpenGL image for the LiDO prediction with Blender colour LVs, we adjust the brightness colour of each object and the ground plane. We do so by scaling the RGB colour by the ratio of the means of brightness calculated at the pixels of the predicted object mask, for LiDO's OpenGL render and the observed image. We can do this as it uses only the *predicted* masks, e.g. this would be equivalent to the final iteration of minimizing the MSE w.r.t. the colour LVs.

5.9 Experimental Evaluation Measures

This section gives the details of the evaluation measures of our interest, in the latent space (Section 5.9.1) and in the image space (Section 5.9.2).

5.9.1 Evaluation of the LVs

For the synthetic dataset, we evaluate the improvement in the LVs for all the methods. We consider suitable evaluation measures specific for the seven different LV-sets, as outlined below.

Object LVs: Object position error is a distance between object central contact points in the image. Object size is the size of the projected object in the image frame, the error is the relative size difference. For azimuthal rotation we measure the absolute angular difference between the prediction and ground truth, but with wrap-around, so the maximum error is 180°. The error metric of the object colour is the RMSE of normalized RGB components (computed as $R/(R+G+B)$ etc.).

Global LVs: Ground plane colour is evaluated as for object colour above. The lighting is projected onto a sphere and evaluated at 313 points uniformly-distributed on the sphere, then normalized; the error is RMSE. To assess the camera error, we place a (virtual) checkerboard in the scene, and compute the RMSE of the errors between the

GT and predicted positions of the grid points in the image, as described in Section 4.4.3.

The multiplicative interaction between illumination and colour introduces a problem when evaluating them separately; by using the normalized metrics above we overcome this issue. The joint result of both factors is directly available via pixel intensities (and is compared via MSE).

5.9.2 Evaluation in the Image Space (2D Projection, Pixels)

We compare the observed and predicted images using the Intersection-over-Union (IoU) of the predicted and GT masks (of objects and ground plane), and MSE of pixel intensities calculated at the GT mask (of objects and ground plane). We can evaluate these measures for both synthetic and real datasets. Note that the IoU of the ground plane assesses differences in the present, missing and superfluous objects. The background (the part of the image not belonging to the ground plane or the object masks) is excluded from the explained pixels. Note that each MSE is calculated at the same pixels for all the methods, as these are calculated only at the GT masks.

5.10 Results

This section provides the results of all the methods: Section 5.10.1 presents the evaluation in terms of the latent variables and shows illustrative examples for the synthetic dataset; Section 5.10.2 presents the evaluation in the image space for the synthetic dataset; and Section 5.10.3 provides the results and illustrative examples for the real dataset.

We ran the methods long enough to allow convergence: 50 iterations for GBO, 100 iterations for Simplex, and 30 for LiDO, see Figure 5.6. All the times shown are for a 4-core CPU for all the methods, to make the comparison fair LiDO is also executed on a CPU, including the CNNs[9].

[9] Although the CNNs run on GPUs were a few times faster, this did not affect the overall speed significantly since rendering and other modules take most of the time.

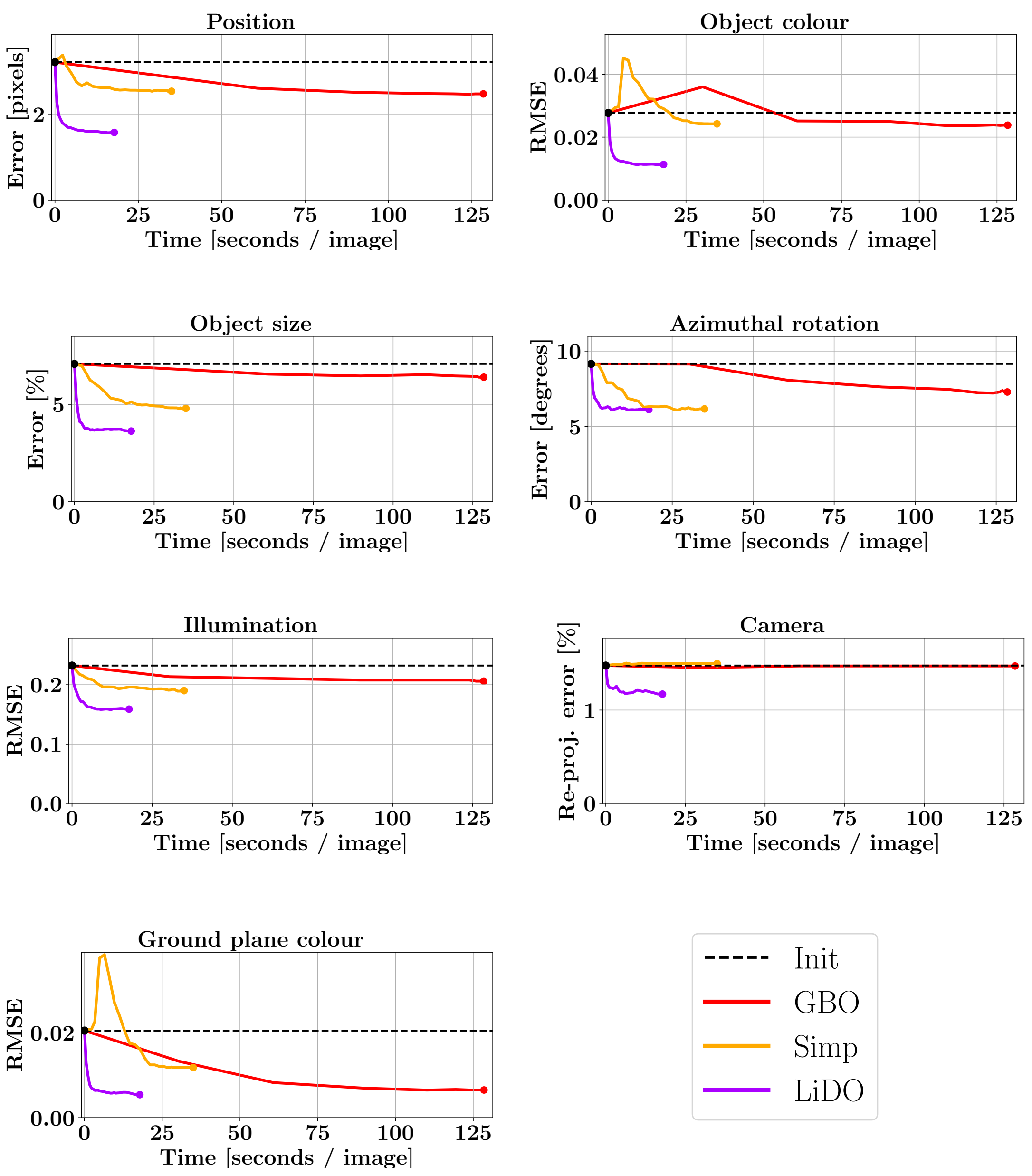


Figure 5.6: Median errors vs time in seconds (top: object LVs, bottom: global LVs). All three methods (GBO, Simp, LiDO) start from the initialization error (Init) located at the black dashed line. Note the rapid convergence of LiDO.

5.10.1 Results: Evaluation of the LVs on the Synthetic Dataset

Table 5.3 (left) shows the percentage improvement of the median error of each of the methods (GBO, Simp, LiDO) over the median error of the initialization. For all seven evaluation measures LiDO outperforms both GBO and Simp. For four out of seven LV sets the LiDO improvements are at least two times higher than the competitors (on

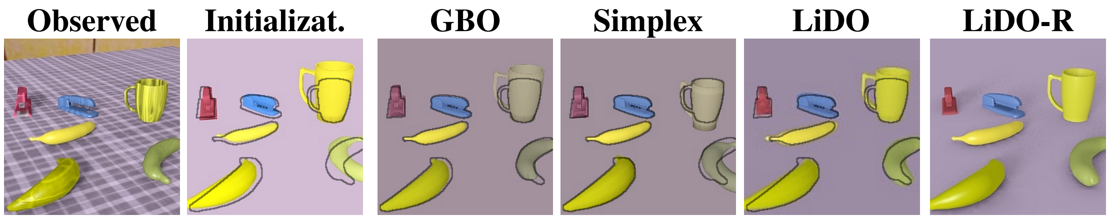
	Initialization		Improvement[%]			Impr. hard cases[%]		
LVs name	Err.	Unit	GBO	Simp	LiDO	GBO	Simp	LiDO
Object position	3.23	pix	22.9	21.0	**51.0***	76	71	**92**
Object colour	0.027	RMSE	14.0	12.4	**59.1***	70	67	**95**
Object size	7.08	%	9.7	32.2	**48.7***	72	70	**87**
Object azimuth	9.15	deg.	20.3	32.6	**33.1**	62	62	**67**
Illumination	0.23	RMSE	11.3	18.1	**31.6***	75	80	**86**
Camera	1.47	r.e.	0.3	-1.3	**20.7***	37	45	**80**
Ground colour	0.021	RMSE	68.2	42.4	**73.6**	80	66	**92**

Table 5.3: Results of the Initialization (left), "Improvement" (centre) and "Improved hard cases" (right). **Initialization:** median errors as per the evaluation measures given in Section 5.9.1, units are: pix – pixels, deg. – degrees, r.e – re-projection error;
Improvement: each value indicates how much (in %) the median error across observations was lower after the refinement compared to the median error of the initialization. Ground truth would give 100% improvement, * denotes a statistically significantly better method.
Improved hard cases: for "hard cases" we considered the worst 50% of the initializations, individually per each LV-set. We show percentage of the hard cases that have been improved (see also plots in Figure 5.8).

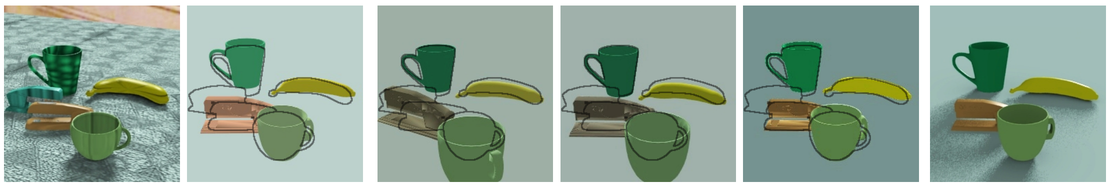
illumination, camera, object: colour, position). We calculate all the different metrics as in Section 5.9.1, e.g. deviation in pixels for object position or angle in degrees for rotation. The absolute values of these errors are given in Figure 5.6 as per the y-axis labels. However, since all the LVs are in different units, we compare the percentage improvement over the initialization.

To assess the statistical significance we conduct a paired test on the errors derived from each image (for global LVs) or object (for object LVs), using the Wilcoxon signed-rank test, at the significance level 0.05. For these LiDO outperforms GBO for 6 out of 7 LVs, and Simp also for 6 out of 7 LVs. This is because GBO does well with ground-plane colour since the objective it minimizes are the differences in pixel intensities between the input image $\mathbf{x}$ and the render $\mathbf{g}(\mathbf{x})$, and Simp performs similarly to LiDO for the azimuthal rotation, but much worse for all other LVs.

Figure 5.6 shows the evolution of the median errors over time for the seven error mea-



Good initialization: given a good initialization all the methods usually converge well, e.g. 4 leftmost objects, for the two rightmost objects (mug and banana) where the initialized masks are less accurate, only LiDO fits the colours properly.



Textures and shadows: There are two staplers in the input image and the blue one was not detected as it is hardly visible. For GBO and Simp the pink stapler converges wrongly, and the same happens for the front mug, which enlarges to explain the shadow. LiDO is robust to such distractors, note for the stapler and the front mug the predicted CAD shapes are different than observed.

Figure 5.7: Example runs for Synthetic dataset, showing from left: the observed input image, the initialization (OpenGL), and images after refinement for GBO, Simplex and LiDO using OpenGL renderers, and LiDO using Blender renderer (LiDO-R). We overlay black contours of the ground truth object masks on top of each OpenGL image to ease the comparison.

sures; it is notable that LiDO obtains a lower error in much shorter time; for error-based methods since we do iterations sequentially for each object, we report the time for each iteration as the average time of reaching it.

Figure 5.7 shows example runs, showing both success and failure cases, see textual descriptions under each image set. More examples of the fitting are given in Section 5.10.4.

For evaluating the robustness and convergence of the optimizers, the most interesting cases are the ones for which the initialization is not accurate. We performed an additional study to measure the performance for difficult cases. For such “hard cases” we considered the worst 50% of the initializations, individually per each LV-set, these ranged from average to poor initializations. Since we have 200 test images, for each LV-set we used the worst 50% of initializations so as to keep the sample sizes suffi-

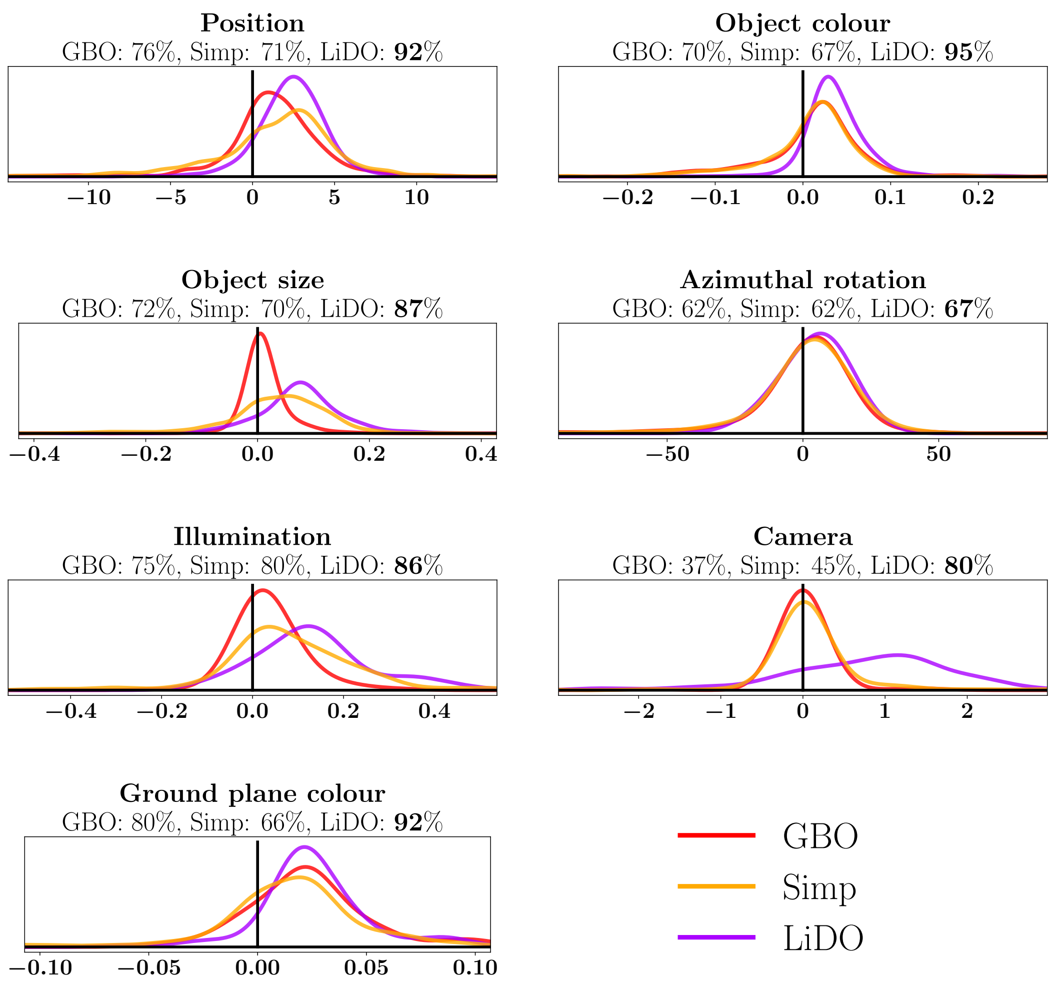


Figure 5.8: We show percentage of the hard cases that have been improved (top of each sub-plot), and the diagram of the distribution of the differences of the initial and final errors ($\mathbf{e}_{init} - \mathbf{e}_{final}$) of the methods for each of the seven LV-sets for hard cases – kernel density estimate. Improved cases are on the right of the 0-line, the less mass on the left the better. For all the LV-sets LiDO performs better than the baselines, note LiDO usually has much less mass on the left side of the plot than the baselines.

ciently large, this led to 532 object LVs and 100 global LVs.

We calculate the percentage of the hard cases that are improved, these are shown in Table 5.3 (right). LiDO performs well and much better than the other methods; for 5 out of 7 LV-sets more than 85% cases are improved. For GBO and Simp none of the individual results exceeds 80%.

We can get a more fine-grained view of the performance by considering the distribution of the changes of the errors, see Figure 5.8. For each LV-set and for each method we show the kernel density estimate (KDE)[10] of the distribution of the differences $\mathbf{\Delta} = \mathbf{e}_{init} - \mathbf{e}_{final}$ of the initial and final errors. Values on the right of the vertical line indicate an improvement. For 6 out of 7 LV-sets, LiDO has a much lower amount of mass on the left-hand-side than the baselines. For example, for the Position LVs it is only 8% (100% − 92%), while these are 24% and 29% for GBO and Simp respectively. For the Camera LVs only LiDO makes noticeable improvements. For Azimuthal rotation the performance of all the methods is similar.

5.10.2 Results: Image-Space Evaluation – Synthetic Dataset

Results for the Synthetic dataset for image-space measures are given in Table 5.4. To allow an equal comparison of three optimizers, all three methods use the OpenGL renderer.

Measure	INIT	GBO	Simp	LiDO	
IoU [ob]	66.2	73.1	74.9	**78.9***	$\uparrow$
IoU [gr]	86.4	88.7	88.8	**91.3***	
MSE [ob]	54.1	29.2	26.4	**21.8***	$\downarrow$
MSE [gr]	34.1	12.8	13.6	**11.9**	

Table 5.4: Results of the pixel evaluation for synthetic dataset. Mean IoU (in %) and MSE ($\times 10^3$) for the objects [ob] and ground plane [gr]. The arrows indicate whether higher or lower values are better. * denotes a statistically significantly better method, using the Wilcoxon signed-rank test, at the significance level 0.05.

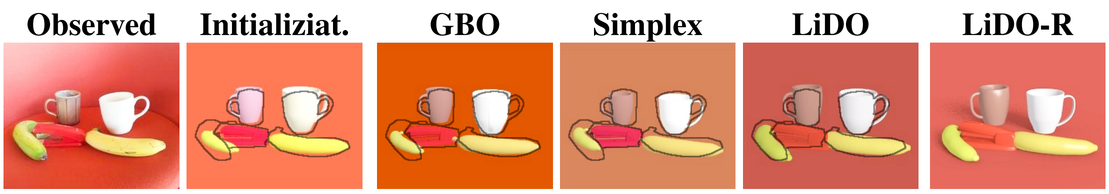
For all the measures LiDO outperforms the other methods, and particularly LiDO works much better for IoU measures. Note that due to unrealisable textures and shadows, the minimal (OpenGL GT) MSE errors are above 0, these are 11.7 for objects, and 8.4 for the ground plane.

[10]KDE is a smoothed histogram, usually using a Gaussian blur. For KDE kernel, we used Gaussian with a standard deviation of $\mathbf{\Delta}_{\texttt{max}}/20$ for object LVs and $\mathbf{\Delta}_{\texttt{max}}/10$ for global LVs – the different values due to the different sample sizes.

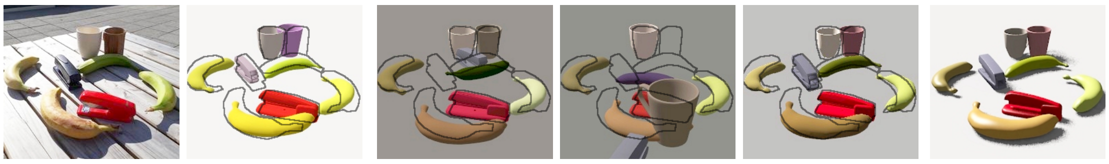
5.10.3 Results: Image-Space Evaluation – Real Dataset

Figure 5.9 shows example runs and explanatory text for real image examples. In general LiDO obtains better results in a shorter test-time than the alternatives, and usually converges to a better configuration. LiDO also has the advantage that it can be trained to handle model mismatch, as shown in the real dataset experiments.

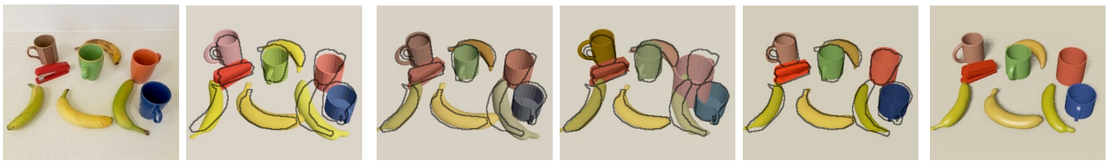
More examples are given in Section 5.10.5, and the video of the fitting at:
`https://youtu.be/Axc0G8IggVU`.



The left-hand banana size/pose is wrong in the Init, only LiDO fits it properly, overall good performance of all the methods, e.g. the left-hand mug obtains brown colour.



Difficult scene, here GBO and Simp diverge objects, LiDO works well (see the gray stapler and the banana near the mugs).



All the methods update the colours and poses of most of the objects, yet LiDO is much more accurate (for example, compare each of the four bananas). The colours of LiDO of all the objects are well predicted (compare output of each method to the observed image).

Figure 5.9: Example runs for Real dataset, the order is the same as in Figure 5.7. Also note that for each image the camera viewpoint is initialized accurately, and how similar the Observed and LiDO-R images are. Obtaining an exact match to the ground truth outline may be impossible because we only have a fixed set of shapes to choose from, none of which may match the actual object shape.

Results for Real Dataset are given in Table 5.5. Since real images are more noisy and difficult, GBO and Simp work poorly for IoU (there is a very minor improvement for objects, and no improvement for the ground plane). All the methods improve the pixel colours (MSE), but note this is because the pixel match is an explicit error measure for GBO and Simp. LiDO, which has been trained on synthetic data, transfers to work better with real images for all four measures.

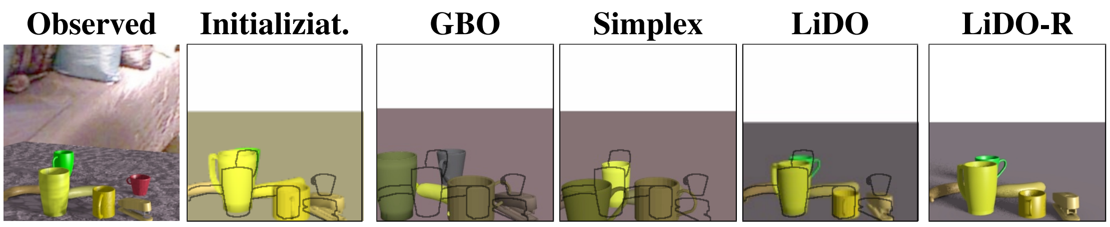
Measure	INIT	GBO	Simp	LiDO	
IoU [ob]	60.9	63.2	61.5	**71.4***	$\uparrow$
IoU [gr]	87.6	87.3	86.1	**91.0***	
MSE [ob]	79.1	42.6	46.2	**35.9***	$\downarrow$
MSE [gr]	69.1	27.5	30.5	**19.2***	

Table 5.5: Results of the pixel evaluation for real dataset. Mean IoU (in %) and MSE ($\times 10^3$) for the objects [ob] and ground plane [gr]. * denotes a statistically significantly better method.

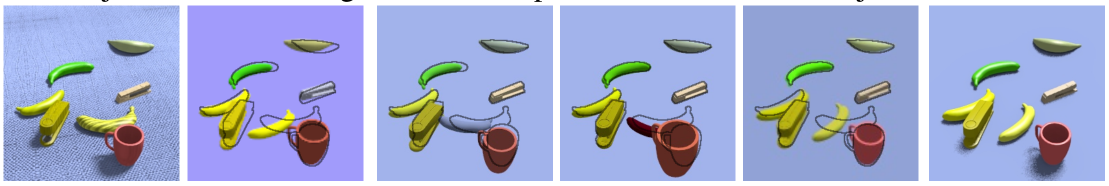
The results in Tables 5.4 and 5.5 afford a direct comparison of the optimizers, all using the OpenGL renderer. However, we can also render LiDO predictions with its “native” renderer Blender (shown as the LiDO-R column in Figures 5.7 and 5.9). We calculated the MSE errors for Blender renderer (IoUs are the same for both renderers since only the appearance changes). These MSE errors were similar to LiDO that used OpenGL renderer (for Synthetic dataset: 21.6 [ob] and 11.3 [gr], for Real dataset: 40.9 [ob] and 23.0 [gr]).

Note the initialized CAD shapes for real images are well matched (see similar object shapes in Figure 5.9), even though these shapes were never observed during training. The objects and global LVs are then refined well. This was facilitated by introducing shape mismatch in the second noisy dataset source of LiDO (see Section 5.6.2).

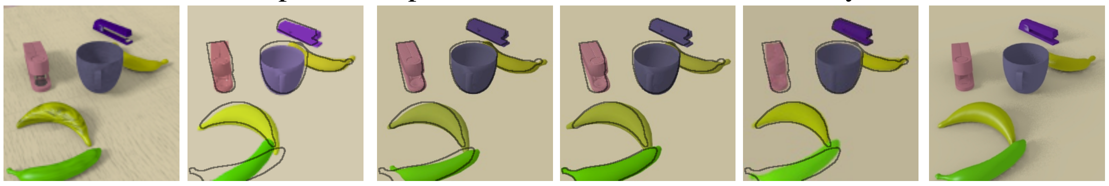
5.10.4 More Examples of Prediction for Synthetic Dataset



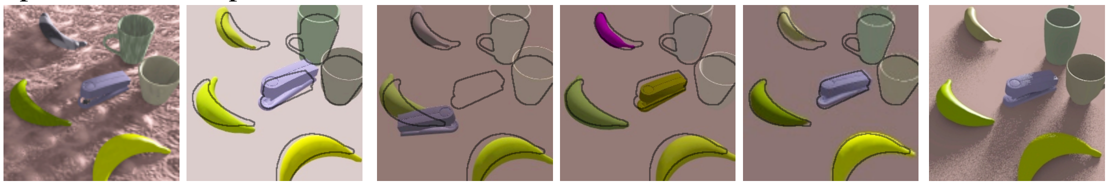
Poor initialization: GBO and Simplex converge to wrong configurations of object poses and colours, while LiDO is robust to initialization errors; note here the initialized object sizes are wrong and LiDO improves all the detected objects.



Typical input (1): There are 7 objects, 6 object converge properly for all the methods, the initialized position of the bottom banana in wrong, all the methods fail to fix it: GBO and Simplex corrupt the colour, LiDO maintains the yellow colour.



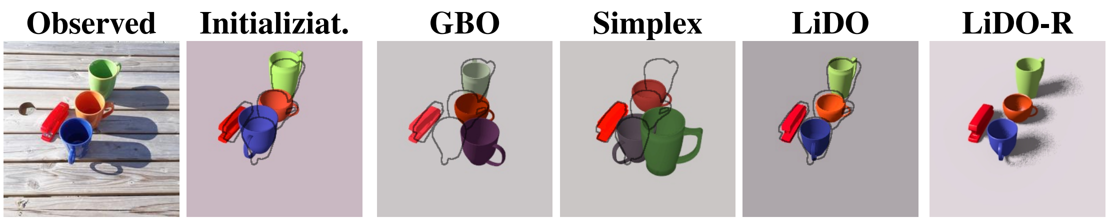
Typical input (2): All objects are initialized well and converge properly, except the green banana for which the azimuthal rotation is wrong, all the methods improve the pose. Note well predicted shadows in LiDO-R.



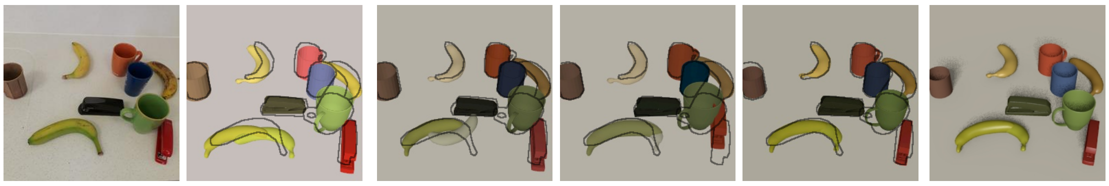
Strong textures and shadows: For GBO the blue stapler (middle) diverges, for Simplex it becomes brown, LiDO is robust to such distractors: stapler pose/size improves, both mugs become smaller with proper colours (also compare Observed and LiDO-R).

Figure 5.10: Example runs for Synthetic dataset, showing from left: the observed input image, the initialization (OpenGL), and images after refinement for GBO, Simplex and LiDO using OpenGL renderers, and LiDO using Blender renderer (LiDO-R). We overlay black contours of the ground truth object masks on top of each OpenGL image to ease the comparison of object poses.

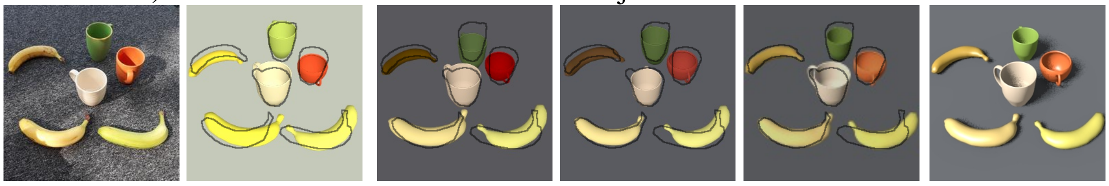
5.10.5 More Examples of Prediction for Real Dataset



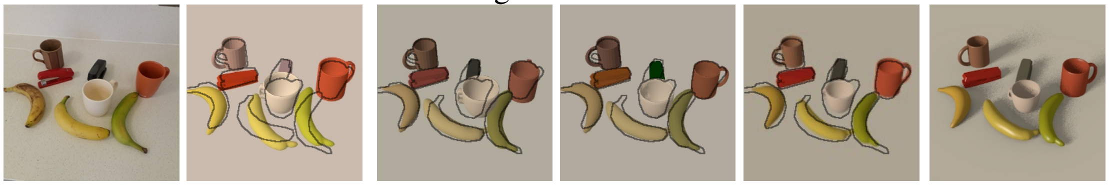
GBO and Simplex corrupt the initialization, LiDO improves the poses and understands the scene well.



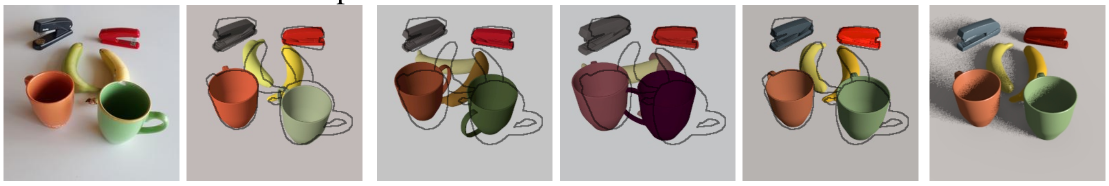
All methods improve the colours, note double detection of the front banana (for the Observed banana in the front, in the Initialization there are two bananas intersecting each other) and different behaviours for this object.



All the methods improve the ground plane colour. Only LiDO accurately fits the pose and the colour of the left-top banana; none of the methods perform well on the switched-orientation banana on the right.



All the methods improve the object poses and colours, note the refinement behaviour of the occluded black stapler.



GBO and Simplex make the Init worse: bananas are rotated, mugs have wrong colours, LiDO improves the colours of mugs. Only LiDO refines the handle of the orange mug, and none of the methods correctly identify the handle position of the green mug.

Figure 5.11: Example runs for Real dataset, the order is the same as in Figure 5.10. Also note how similar the Observed and LiDO-R images are.

5.11 Discussion

Above we have demonstrated LiDO, a full framework for the initialization and refinement of a 3D representation of the scene from a single image. The main features of LiDO are: the advantage of not requiring an error metric E to be defined in image space, rapid convergence, and robust refinement in the presence of noise and distractors. LiDO is generally robust to issues that are common for error-based methods: the updates can point in wrong direction when dealing with cluttered scenes and shadows in observed images; difficulties can arise from an inability to exactly match the target object with one of a different shape; and when predicted objects overlap the background or other objects. LiDO is generally robust to such problems as it directly learns to optimize in the latent space. Our method is not limited to rigid objects, one could use LiDO for e.g. multiple-human pose estimation, or hand-pose and appearance reconstruction.

One apparent limitation of LiDO is that we need to train an additional Prediction Network in advance for a particular dataset. Incorporating neural networks requires extra time for training, but allows for smarter LV updates as outlined above. Also, as shown in Figure 5.6, LiDO is much faster at test-time than the competitors. This is because each LiDO update requires only a single render of the scene, while standard optimizers search over the error landscape and usually need several renders (e.g. within a line search to choose the step-size) before accepting a new configuration.

Another potential limitation is a need for a synthetic training dataset. Although such a dataset is required to train the LiDO Prediction Network, for any method one also needs to use an initializer of the scene graph LVs. This means that: i) the initializer was trained already on such dataset, and ii) a 3D graphics representation of a scene graph LVs would be available. This representation and the dataset can be reused for LiDO, e.g. by computing the initialization errors, or by adding noise to the ground truth LVs. The advantages of LiDO mean that it could be a critical component in the development of future vision-as-inverse-graphics systems.

Chapter 6

Conclusion and Future Work

Throughout this thesis we have employed the Vision-as-Inverse-Graphics paradigm, which consists in inferring the 3D scene graph latent variables and then rendering these to reconstruct the input image. This analysis-by-synthesis approach provides a much more detailed explanation of the scene than standard bounding-boxes or pixel-level segmentation – it predicts the underlying 3D scene given only a single image.

The 3D scene graph reconstruction is an attractive representation. It is physical and interpretable, therefore enables interactions with the 3D scene, such as computation of possible paths so as not to collide with the objects present in the scene. The methods we have developed are designed to be applicable to a wide range of tasks and domains, and are not limited to rigid objects; one could employ the VIG framework for e.g. multiple-human pose estimation, or hand-pose and appearance reconstruction.

Our framework acts like an autoencoder, where the latent representation of the scene is interpretable and is of variable dimension to match the number of the detected objects. This representation is of a key importance as it can be useful if we wish to edit, refine, or interpret the scene. It could be helpful for e.g. an industrial robot interacting with a set of objects, and could facilitate subsequent reasoning methods such as the ones dealing with the task of Visual Question Answering (Antol et al., 2015).

A large number of methods for 3D reconstruction use additional input: multi-image input, video input, or an image with a depth channel. In contrast, throughout this thesis we have emphasized that we tackle the most challenging vision problem, i.e. of the rich scene explanation from a single image. Indeed, without a depth channel we needed to embed in our systems prior knowledge about the objects in the world so as to be able to infer a plausible 3D configuration. Another important aspect that we considered consists in combining evidence from multiple cues to allow for inference of a likely scene configuration.

The next section provides the summary of the contributions of this thesis, Section 6.2 provides self-critique of the work carried out in this thesis, while Section 6.3 discusses possibilities for future work.

6.1 Summary of Contributions

In Chapter 3 we investigated how the recognition models can be used to infer the scene graph given only a single RGB image. These models were trained using realistic synthetic images and corresponding ground truth scene graphs, obtained from a rich stochastic scene generator. In the proposed framework we incorporated auxiliary modules for object contact point and projection scale prediction, to allow the computation of object poses and sizes in 3D scene coordinates, given the camera parameters. We have shown how to successfully put all of the components together to create an interpretable scene-graph representation of a 3D scene from a single image. Importantly, the framework has been shown to work on both synthetic and real images.

In Chapter 4 we introduced the Probabilistic HoughNets framework for combining probabilistic votes to infer the camera parameters given the votes cast by the detected objects. Each detection provides one vote in the form of a noisy low-dimensional manifold in the Hough space, and by intersecting the votes probabilistically we reduce the uncertainty on the camera parameters. Importantly, the way in which the detections cause voting in the Hough space is learned from data. An outlier model allows for exclusion of corrupted predictions so as they do not interfere with the final result. Probabilistic HoughNets by definition can cast multimodal votes, and we have demonstrated that the composition of these leads to an overall improvement in performance. This is because the votes are combined by intersecting whole distributions, not by aggregating point estimates.

In Chapter 5 we developed a framework called ‘Learning Direct Optimization’ (LiDO) for optimization of the latent variables of a multi-object scene. Given an initialization of a scene graph, its refinement typically involves computationally expensive and inefficient search through the latent space. To overcome this issue, instead of minimizing an error metric that compares observed image and the render, LiDO optimization is driven by neural networks that make use of the auto-context in the form of a current scene graph and its render to predict the update of the latent variables. Our experiments showed that the LiDO method converges rapidly, as it does not need to perform a search on the error landscape, produces better solutions than error-based competitors, and is able to handle the mismatch between the data and the fitted scene model. We applied LiDO to a realistic synthetic dataset, and showed also that the method transfers to work well with real images. These advantages of LiDO mean that it could be a critical component in the development of future vision-as-inverse-graphics systems.

6.2 Critique

In this section we discuss the decisions taken and possible alternatives. We outline some weaknesses and limitations of the developed methods, and discuss how these might be overcome.

Detector The initialization networks rely heavily on the object detector. We developed our own object detector which is trained to activate at the object contact points to allow for accurate back-projection (see Section 3.3.1). The main reason for training our own detector was that as at the time of developing the initialization networks (2016) the accuracy of the existing detectors was low (approximately 30% overall), e.g. approximately 0.6 precision and 0.5 recall on Microsoft COCO dataset[1]. In addition, the detectors trained on real images are strongly biased towards the typical object appearances, and have difficulties in detecting objects at uncommon colour configurations. In real images, the objects are also typically taken at the height of a person and within a limited elevation range. In contrast, using SSG, we could train own detectors for specific classes using larger class-specific datasets. Note the objects in our dataset are observed at wide-ranging colour, viewpoint and illumination configurations, which allows for more robust detection. Nevertheless, nowadays the state-of-the-art detectors, such as Mask-RCNN (He et al., 2017), work well and these could be used to detect the object bounding-boxes. The main advantage of using off-the-shelf detectors is that these are already trained for a large number of classes, and do not require NMS post-processing, which allows better handling of cluttered scenes. Then, as we do not

[1]`http://cocodataset.org/#detection-leaderboard`

use bounding-boxes, the prediction of the projection scale and the contact point could be done within the LV initialization networks. Note the LV initialization networks may use smaller datasets than those required by the detectors.

Renderers Preparing existing CAD models for OpenGL rendering required a lot of manual work. The publicly available 3D datasets are typically used to generate voxels, but are not directly suited for OpenGL rendering. This is due to various issues with surface normals (reverted), and frequently some edge types are wrong – edges can be either shared by adjacent faces, or be separate (duplicated) – the correct case depends whether the surface of the object part should be smoothed (e.g. a sphere) or not (e.g. a cube). To compare our method with OpenDR we needed to use OpenGL, but it would be better to use advanced renderers (Blender etc.) that do not require such fixes. Note our framework does not need to use OpenGL and can use any renderer.

PHNs The main limitation of PHNs is the dimensionality of the latent space, for which we have considered 3 dimensions. With more dimensions the execution time would be scaling similarly to the standard GHT approach that uses a grid of bins. Since such scalability is exponential with the number of dimensions, this would limit PHNs applicability to approximately 5 to 6 dimensions. However, this issue may be overcome by either using a single Gaussian in some dimensions (that are known to be uni-modal), or allowing the Gaussian mixture model to "float", i.e. to predict the mean and the variance of each of the Gaussians, instead of the grid structure currently used. The advantage of PHNs is that the complexity scales linearly with the number of the Gaussian components, not exponentially with the space dimension. Therefore, if PHNs make use of a fixed number of "floating" components, the issue of the exponential scaling with the number of the dimensions can be overcome.

LiDO We observed that LiDO converges better and fixes larger errors than the competitors, yet still it may not fix very large errors, e.g. in object position or rotations. This happens because such large errors are very rare in the training set. Possible solution would be to introduce a "large-z" dataset, analogously to the small-z one that we have used for training. To obtain such a dataset, one could sample the initializations uniformly in the whole domain of interest (e.g. for the initialization of the azimuthal rotation sample 360 degrees uniformly).

6.3 Future Work

In this section we focus on the strands of further research which are most related to the frameworks developed in this thesis. We note, however, that several other directions can be considered given the breadth of topics in computer vision.

Object shape representation One area for possible extension is the representation of the shape of the objects. The contribution of our frameworks was to demonstrate strong performance for handling *multiple objects* in a complex scene, plus camera and illumination variables. However, since the predictions are made *per detected object and per object class*, one could extend the 1-out-of-K object shape encoding to use e.g. shape and texture morphable models like Blanz and Vetter (2003).

To investigate such a shape representation, in Appendix A we developed a parts-based deformable generative shape model that can cover a wide range of object class shape variability. This is done by specifying an object template and skeleton, which we demonstrated for the teapot object class. Importantly, the whole deformation is implemented to be automatically-differentiable. It is a standard practice to control a mesh, of either fixed or variable shape, using a skeleton controlled by rotation parameters of each bone, see for example the work by Loper et al. (2015). In contrast, our bones are controlled by joint locations, and we specified methods for automatic computation of bone rotation parameters. This allows for robust local gradient-based optimization where the parametrization of the child bone joint locations are independent of the parent. Otherwise, the optimization might not succeed if e.g. the remaining bones fit the observation well but not the parent one. In this case the fit w.r.t. the parent bone rotation might lead to a local optimum as any rotation would unnecessarily move away all the descendant bones. A related approach by Shi et al. (2007) also controls a mesh via joints, but not in a differentiable manner. In addition, our approach allows for variable shape via bone thickness and bone lengths, thus the whole deformation is controlled only by the skeleton. The derivatives are then used to fit the template to a collection of 3D mesh instances using a gradient-based optimizer. Importantly, such a skeleton-based representation is interpretable, and could be used in VIG approaches. One may use our model to explain an object by fitting its render to the observed image, or to sample new object shapes for the purpose of generation of a large amount of synthetic data.

On the other hand, voxel-based or volume-based approaches, such as the Occupancy Network (Mescheder et al., 2019) are the best performing methods for 3D shape reconstruction, but these representations do not have notion of object parts and are not morphable. The Occupancy Network predicts object boundaries as a function in the 3D space, then these can be materialized to a mesh with a chosen resolution. Extending this approach to be able to deform the object to fit image data, and with a notion of shape interpretability such as object parts, would be of a great interest.

Object appearance representation Regarding appearance, we used single colour objects, but multi-colour or textured objects would be more natural. Common approaches to appearance modelling consider multi-textured objects with textures being different per object part, see e.g. Wang et al. (2016). The texture model can be learned using collections of images (typical texture collections include patterns such as fabric, brick, marble, wood etc.), or be regressing a texture map. These approaches can potentially deal with a variety of appearances, but usually work only for a single deformable shape, such as a sphere, with a predefined mapping of the texture onto the mesh. The approach by Oechsle et al. (2019) overcomes this issue and makes use of a "texture field", which models a texture in a function space from 3D space to colour. This allows for the reconstruction of invisible areas for arbitrary shapes. The method uses VAE and GAN networks, and at the test time the texture can be materialized from multiple viewpoints (given a mesh). Another approach to texture modelling is to learn how to copy the pixels. Kanazawa et al. (2018) predict the "texture flow", which maps from object surface to the image pixels defining from where to copy the colour. Copying the pixels is a promising approach, as it can deal with arbitrary noise, such as text, logos, drawings or other common fine-grained details. Note that the methods predicting the textures by default do not incorporate the notion of the illumination. This means, for instance, that when one rotates the reconstructed object, it is illuminated in a fixed manner. One could possibly overcome this problem by decomposing a lit texture into illumination plus a canonical texture.

Learning object shape and appearance representation directly from real images A crucial issue is related to the scalability of the object shape and appearance models with the number of object classes. Usually the approaches for shape modelling require 3D CAD objects of a given class, and thus can be developed only for the common object classes, for which there are several instances covering a variety of possible shapes. The variability of the appearance of such CAD models is very limited, and often no appearance of the mesh is available. Ultimately, one would prefer to learn the shape and appearance model directly from real images of such objects, without CAD models. For example, a recent approach of Kanazawa et al. (2018) jointly learns the object shape and texture from images, object mask and key-point annotations. Their model is trained for the bird class so as the rendered object under the predicted camera is similar to the observed image, including shape masks and key-points consistency. The texture is modelled using a texture flow, so is based on copying the pixels from the image to the mesh. The shape is represented as a deformed sphere, so it would be of a great importance to extend this approach to learn a more complex shape representation from real images, capable of dealing with objects with holes, such as the mug class.

Refinement of the presence/absence of objects In our experiments we assumed a fixed set of predicted objects, as obtained from the initialization networks. This allowed us to conduct a proper comparison of the optimization methods and the analysis of the improvement in the latent variables for the same set of objects. For the initialization, over-detection could be investigated, so as not to miss any correct but weak detections. This means a much larger number of objects with a high ratio of false positives would be considered. One could perform verification of the presence or absence of these objects – superfluous objects that do not match the input could be removed from the initialization. Another approach would be to refine the set of instantiated objects iteratively. For each object at each iteration one could predict whether it should be rendered, so that the object could alternate between visible and invisible states.

More complex physical configurations There is a wide range of possible extensions of the scene graph representation that are beyond the scope of this thesis. For richer object poses, one can consider an object support prediction, such as in the Holistic 3D Parsing of Huang et al. (2018), where objects might lie on top of others. More complex object poses could be allowed, such as an object lying on its side or upside-down. Furthermore, one can add in-plane rotation to the camera configuration, or consider a richer illumination set-up such as colourful illumination environment or even additional light sources present "inside" the scene (such as light-bulbs).

Goodness of fit A separate topic for further research is how to assess the fit of the outcome of the prediction or optimization. Once the objects are explained, one could develop models to predict the errors that have been made. This can be done by comparison of the observed and predicted images. Note such networks can use a richer input, including the predicted object masks or depth channel. We call this an assessment of the *goodness of fit*. This can be considered (i) at the level of the presence or absence of objects, or (ii) for each latent variable associated with an object (or the global latent variables).

The level of presence/absence evaluation would aim to answer questions such as: *is the object a false positive? Are there any unexplained areas? Is the object class correct?* One could consider neural networks that, given the input image and renders with and without the object, assess which case is more likely. The goodness of fit could be also evaluated per latent variable (e.g. to predict whether the object size is correct). One could say that this is exactly what the LiDO Prediction Network is dealing with, however, this is only one possible solution for the task of goodness of fit assessment. In LiDO we have so far predicted the updates as a single value (except the rotation via bins). When LiDO predicts no update, this does not necessarily imply a proper convergence, but could also arise from the network being unsure whether to lower or increase a given value. Having these predictions in the form of a possibly multimodal posterior distribution for each latent variable, for example via PHNs, would make the predictions more explainable. One could also use the MAP or a sample from this distribution as the LiDO update. Moreover, for goodness of fit assessment one could develop methods that not only predict the errors in the latent variables, but also the value of the evaluation error measure, e.g. such as the Intersection over Union (shape mismatch) or the re-projection error (camera calibration).

Appendix A

Modelling Shape Using a Skeleton-Based 3D Deformable Mesh

A.1 Skeleton-Based 3D Deformable Mesh

One of the aspects of the VIG approach is a need to synthesise a range of possible object shapes, both to generate synthetic scenes, as well as to understand the objects present in the image and their shape and poses. The goal of this chapter is to obtain a shape model of an object of a given class that is deformable and can model a wide variety of object shapes. Our approach is based on a skeleton, like that which could control e.g. a human pose, but is not limited to shapes that actually have a skeleton. The deformation is automatically-differentiable and automatically computed wrt. joint positions, which serve as control points. This allows us to perform various tasks that make use of the gradient, such as optimization. Importantly, such a skeleton-based representation is interpretable, as we can obtain properties of object parts or their fragments from the skeleton pose, and can edit the pose or shape of a particular object part.

Usually a skeleton is controlled by rotation parameters of each bone, see for example the work on human pose by Loper et al. (2015). In contrast, our bones are controlled by joint locations, and we specified methods for automatic computation of bone rotation parameters. This allows for robust local gradient-based optimization where the parametrization of the child bone joint locations are independent of the parent. Otherwise, the optimization might not succeed if e.g. the remaining bones fit the observation well but not the parent one. In this case the fit w.r.t. the parent bone rotation might lead to a local optimum as any rotation would unnecessarily move away all the descendant bones.

Our work below considers modelling the shape of a teapot object. The teapot is composed of different components (parts) like the spout, body and the handle, and each of the components features a high variability of shape, size and pose. Therefore, a teapot is a challenging class. Our deformable model is able to model the variability in each of the components, while being controlled by only a small number of parameters. One important aspect of our deformations is that the shapes can cover a large variety of object part sizes as we allow for variable thickness of the parts. Such a model could be used in the VIG approach, the shape recognition task would then be to infer the parameters of the skeleton. To create a generative model of the objects, we need to obtain the prior distribution for the model parameters from a dataset of mesh instances. Therefore, our first goal is to fit a mesh of the deformable model, controlled by a skeleton, to different instances of the objects of the same class.

A.1.1 Introduction

To keep the number of parameters low, a mesh is deformed by a skeleton composed of a small number of bones arranged in a structure of multiple trees (a forest), as visualized in Figure A.1 (top). A skeleton has a small number of parameters compared to the number of mesh vertices; these are the bone parameters, which are often limited to a small set of allowable transformations (typically 3-5 parameters per bone). We aim to infer the parameters of the bones for each of the given set of instances.

The deformation of a mesh is defined by the properties of the bones. A change of pose of a bone, from its initial pose to the final one, defines an affine transformation. For instance if a bone is rotated, all the vertices assigned to that bone are also rotated in the same way, hence they follow the bone pose.

Figure A.1 presents a teapot in the undeformed pose which in graphics is called the "rest pose" (top) and a teapot pose where the handle joints are translated (bottom). The teapot is composed of three components: the body, the handle and the spout. Each component is a tube. In case of the body, the tube ends in the shape of a teapot-top. The start of the spout and handle tubes are connected to an internal part of the teapot.

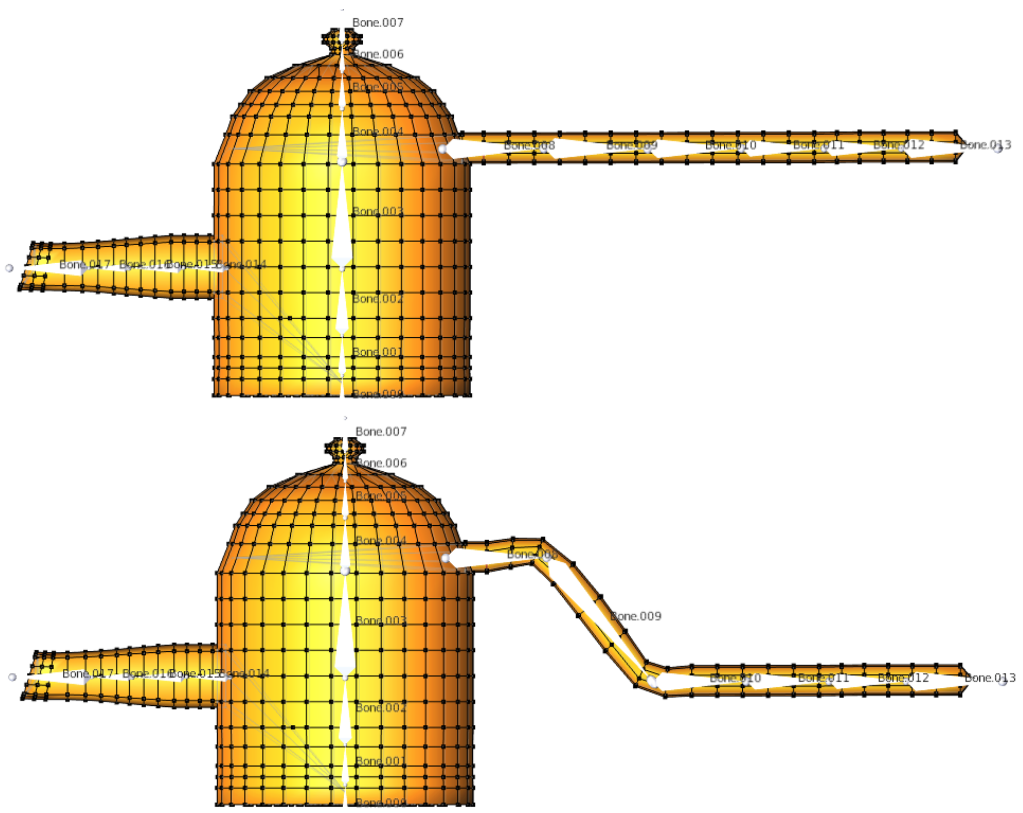


Figure A.1: A deformable teapot composed of 16 bones (8 in the body, 4 in the spout, 6 in the handle). Top: rest pose, bottom: deformed handle.

The straightforward skeleton parametrization that consists of bone rotation parameters wrt. parent allows for a simple sequential computation of the whole pose of the skeleton. However, it is undesirable when it comes to inference or manipulation, as the bones parameters are then a complex function of all their ancestors. When one would need to e.g. shift a particular fragment of object part downwards, this would require not only to rotate the preceding bone, but also to rotate the next bones so as the further parts do not shift as well. As shown in the bottom plot of Figure A.1, to keep the handle bones horizontal after rotating the second bone of the handle, one needs to rotate the third bone in an opposite direction. Placing the further bones at the initial locations would require further manipulations.

Therefore, our deformation is defined by the positions of the joints of the skeleton. Shifting a joint moves only the bones connected to this joint. In general, joints behave like control points – we can move each joint independently of the others. A difficulty that arises from this approach is that we need to define the specific functions to automatically compute the bone rotations for different degrees of rotational freedom. We keep the parameters and constraints relative – rotation is also always limited with respect to its parent, so we can easily evaluate the angles of deformation compared to the rest pose.

To allow smooth deformation of a mesh, each vertex position can be a result of a weighted combination of transformations (coming from a few bones). Figure A.2 presents an example of a teapot deformed using the skeleton and vertex weights.

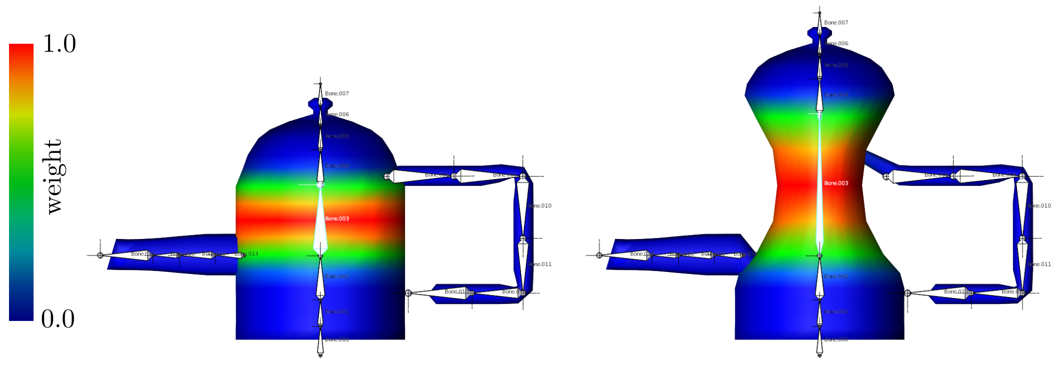


Figure A.2: The deformable teapot with a skeleton.

The colours denote the weight of vertices assigned to the bone in the middle of the body (red = 1.0, blue = 0.0). In the second picture the length of the middle bone was increased by a factor of 2, while the thickness was decreased by a factor of 2. Since the weights blend from one part to the others, the skeleton smoothly deforms the body.

The 3D mesh of the deformable object is modelled and prepared in Blender software. We build a skeleton using the Blender's skeleton object. One advantage of using Blender for building of the skeleton is that we can set the computed vertex weights and then visualize basic deformations by rotating the bones in Blender.

We then define our own computation of the deformation system for our specific purpose, then enable the skeleton to control the deformation wrt. joint positions. We implement the deformation in a special manner that makes the computation automatically-differentiable, hence allows for obtaining of the gradient of the fit and shape optimization.

It is worth mentioning that we do not deform a predefined shape of a fixed size as usually done in Computer Graphics. For example in Shi et al. (2007) a deformable mesh using bones is considered, also controlled by the joint positions, but not in a differentiable manner. However, their model maintains the properties of the initial shape (e.g. the length of arms and legs) by application of rules of rigidity, balance and skin properties to obtain the final pose, while in our deformable model we allow arbitrary deformations of the scale of each component (within constraints imposed on the skeleton).

A.1.2 Notation

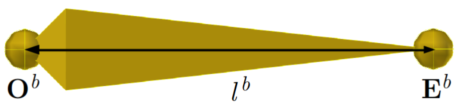


Figure A.3: A single bone.

The skeleton setting and possible transformations can be described by B bones and their degrees of freedom. The skeleton is parametrized by $\boldsymbol{\theta}$, which consists of positions of the skeleton joints plus the scaling parameters of the thickness of B bones (in two axes).

The initial values of the parameters are specified directly by the skeleton rest pose. The rest pose parameters are denoted by $\boldsymbol{\theta}_0$ and the deformable mesh vertices are denoted by $\mathbf{V}_0$. The deformed mesh vertices $\mathbf{V}$ are obtained by modifying the rest pose parameters $\boldsymbol{\theta}_0$ and then the rest pose vertices $\mathbf{V}_0$ using modified parameters $\boldsymbol{\theta}$, i.e. $\mathbf{V} = V(\boldsymbol{\theta})$. The deformation function V applies an affine transformation (details given in Equation A.1 and A.2), respectively to each of the vertices of the deformable mesh.

The b-th bone starts at a point $\mathbf{O}^b$ (O for origin) and points to the end point $\mathbf{E}^b$, with length $l^b = \left\|\mathbf{E}^b - \mathbf{O}^b\right\|$, as presented in Figure A.3. The rest pose parameters are denoted with 0 subscript, e.g. $\mathbf{E}_0^b$ is the initial position of the end of the b-th bone. If we know the bone identity from the context, we can omit the b superscript index.

The non-negative weight of the b-th bone on the m-th vertex $\mathbf{v}_m$ is denoted by $w_m^b,\ b \in \{1,\dots,B\}, m \in \{1,\dots,M\}$. The sum of the weights for a given vertex equals one:

$$\sum_b w_m^b = 1\ \forall m, \qquad w_m^b \geq 0\ \forall m \forall b.$$

Each bone defines an affine transformation. However, when vertices belong to multiple bones (with given weights), the location of the vertex given $\boldsymbol{\theta}$ is a weighted sum of a number of affine transformations, as described below.

Notation: if $\widetilde{\boldsymbol{A}}$ is an affine transformation matrix in homogeneous coordinates, then by $A(\boldsymbol{x})$ we denote a function that performs the given transformation, i.e. transforms $\boldsymbol{x}$ to homogeneous coordinates ($\widetilde{\boldsymbol{x}}$), and returns $\widetilde{\boldsymbol{A}}\widetilde{\boldsymbol{x}}$ back in the usual coordinates. We denote by $\widetilde{\boldsymbol{A}_b}$ the affine transformation of the b-th bone from its original to the final pose, and analogously the corresponding transformation function by $A_b(\boldsymbol{x})$.

The function for the affine transformation for vertex m parametrized by $\boldsymbol{\theta}$ is a linear combination of the affines according to the weights, and is given by:

$$A_m^{\boldsymbol{\theta}}(\cdot) = \sum_{b=1}^{B} w_m^b A_b^{\boldsymbol{\theta}}(\cdot). \tag{A.1}$$

Thus the final position of vertex m is given by applying the above transformation:

$$\mathbf{v}_m = A_m^{\boldsymbol{\theta}}(\mathbf{v}_{0m}), \tag{A.2}$$

where $\mathbf{v}_{0m}$ is the position of the m-th vertex before any deformation happens.

The computation of an affine matrix of a given bone is described in Section A.1.5.

A.1.3 Bone Coordinate Frames

We first define the bone axes: the scaling along the y-axis changes the bone length, while changes along the x-axis and z-axis affect the thickness through the remaining two directions (we follow the bone axes order used in Blender).

A pose of each bone is given within its "parent (coordinate) frame", a new pose produces a new frame, called the "local (coordinate) frame". Figure A.4 presents the details of the coordinate frames. The scaling is performed along the chosen axes of the local frame. The pose of the whole skeleton is computed sequentially from each root bone to the leaves.

A.1.4 Details of the Skeleton Deformation Properties

A.1.4.1 Scaling

Scaling is performed directly using the scaling parameters s_x and s_z for the x-axis and z-axis (controlling thickness of a bone). Scaling in the y-axis (the length of a bone) is calculated from the distance of the bone joints.

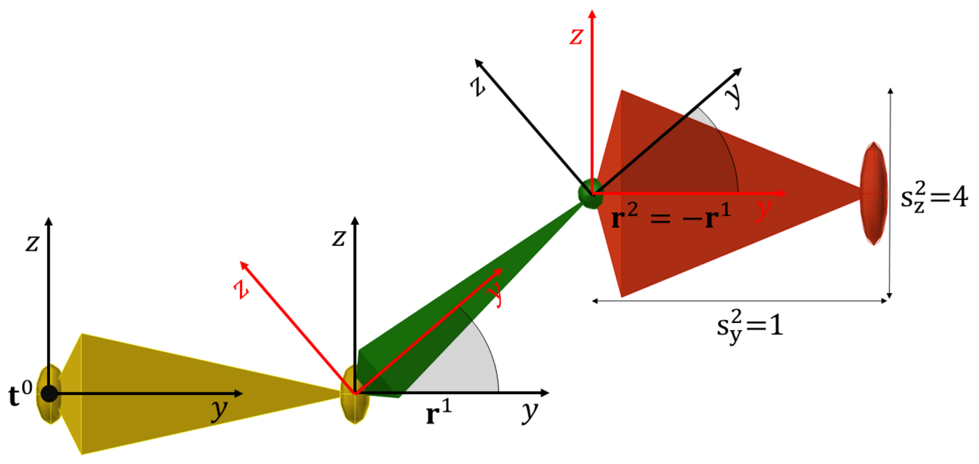


Figure A.4: Bone frames.

The yellow root bone is not rotated, so its parent and local frames overlap. The green bone has the same "parent frame" (in black), the origin is at the bone's starting point. The green bone, rotated by $\mathbf{r}^1$, produces a new frame, its "local frame" (in red colour). This local frame is used as a parent frame of the red bone: again, the parent frame axes are in black, local frame axes in red. Here the red bone rotation is $\mathbf{r}^2 = -\mathbf{r}^1$. Each bone has a scale parameters: the green bone is of the default scale, while the red and yellow bones are wider.

Scale parameters can be constrained to vary together in different manners. For instance, if we want to allow scaling of both thickness parameters together, then $s_x^j = s_z^j$. This can be the case e.g. for a car wheel, which should stay a cylinder, here s_y^j would control the wheels thickness. Another example is to constrain a set of bones (for instance a chain of bones controlling a tube) to have the same scale for a set of bones, so as the tube parts would be scaled together consistently.

A.1.4.2 0D and 1D Rotation

The simplest case of rotation is when the rotations are not allowed, e.g. for a teapot body. Note one could model the rotation of the whole object to explain the object pose. 1D rotation happens in a usual way, i.e. the joints are moved within a 2D plane, for example teapot spout or handle parts are parametrized using 1D rotations. The angle between the bones directly maps to the bone rotation.

A.1.4.3 2D and 3D Rotation

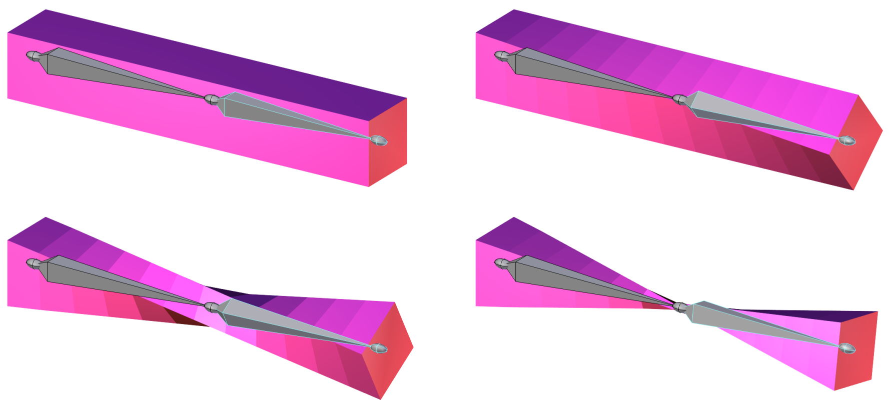


Figure A.5: Twisting problem. A simple object and the deformation by twisting the child bone (rotating around the y-axis) by 40, 120 and 170 degrees, respectively.

For some classes, e.g. a human arm, we would need more degrees of freedom (DOFs). Usually, if a bone allows rotations in more directions, it can be rotated at least around its x and z axes in the local frame. A sequence of non-zero rotations around these two axes results in a rotation around the third one, i.e the y-axis, which we call a twist of a bone. The twist is the roll in the roll-pitch-yaw angles. In such a case a part of the mesh collapses, as presented in Figure A.5.

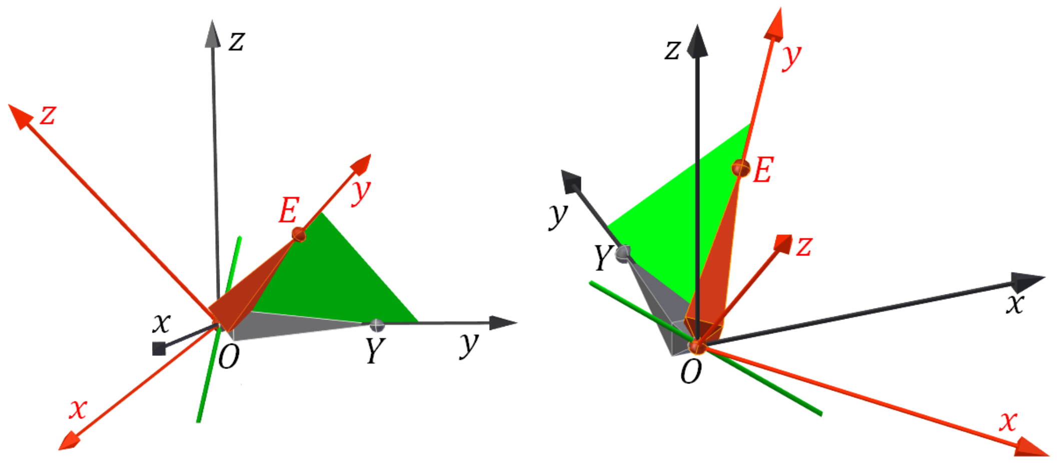


Figure A.6: Untwisted 2D bone rotation in 3D (from two different views for clarity, black/red axes refer to the parent/local frames, in the analogous way as in Figure A.4). The grey bone is rotated to its final pose, represented as a red bone. The axes in black represent the parent frame, the axes in red are a new local coordinate frame. The bone is rotated around the green axis (lying within the green rotation plane $\boldsymbol{EOY}$).

Hence, for more degrees of freedom, we consider the following two cases.

- 2D: given the desired joint positions, the rotation is always kept untwisted. An untwisted rotation to the end point $\boldsymbol{E}$ is a rotation around the axis perpendicular to both the $\boldsymbol{OE}$ axis and the parent $\boldsymbol{OY}$ axis, passing through $\boldsymbol{O}$. The example of a green rotation axis and green $\boldsymbol{EOY}$ plane is presented in Figure A.6.
- 3D: twisting is allowed, but it has to be limited, although one may use several bones to overcome the twisting problem for arbitrary rotations. An example can be a human head: twisting of a neck might be allowed to around ± 90 degrees, relative to the shoulders. In this case the twist becomes an additional parameter r_{twist} and is stored as the additional bone rotation parameter.

A.1.5 Determining the Affine Matrix of a Given Bone

Given all the skeleton parameters, we can find the transformation matrix of a given bone. For a root bone we know the parent frame and current parameters, so we can obtain the local frame and the transformation matrix. We compose the parent frame rotation with a local one to obtain the parent frame for the next bone. In this recursive approach we can compute the final rotations of all the bones in the global coordinates.

The bone frame rotations are stored using quaternions, which avoid the gimbal lock problem (Kuipers, 1999). A unit quaternion is a unit length vector $\boldsymbol{q} = (q_0, q_1, q_2, q_3) = (q_0, \boldsymbol{q}_{1:3})$, where q_0 serves as a value to ensure a unit length, where the length of the sub-vector $\boldsymbol{q}_{1:3} = \boldsymbol{a}$ is between 0 and 1. Antipodal quaternions, $\boldsymbol{q}$ and $-\boldsymbol{q}$, represent the same rotation. It is simple to compose rotations by quaternion multiplication, and the rotations composition of rotations is essential to compute skeletal deformation.

The computation of the affine of a given bone uses:

- The bone parameters: $[\boldsymbol{O}, \boldsymbol{E}, \boldsymbol{O}_0, \boldsymbol{E}_0, s_x, s_z]$ – the bone joint start and end positions, initial joint start and end positions, size on the x-axis and z-axis.
- The parent bone parameters: $[\boldsymbol{O}^p, \boldsymbol{E}^p, \boldsymbol{q}^p]$ – parent bone joints and rotation caused by the parent (i.e. parent frame). For roots, which do not have parents, we set $\boldsymbol{O}^p = \boldsymbol{O}_0, \boldsymbol{E}^p = \boldsymbol{E}_0, \boldsymbol{q}^p = (1,0,0,0)$, i.e. the bone rest pose parameters are treated themselves as its parent, hence there is no inherited rotation for roots.

Translation caused by a given bone (all transformations are given in homogeneous coordinates) is represented as:

$$\widetilde{\boldsymbol{T}} = \begin{bmatrix} 1 & 0 & 0 & O_x - O_{0x} \\ 0 & 1 & 0 & O_y - O_{0y} \\ 0 & 0 & 1 & O_z - O_{0z} \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Initial translation from the global origin, $\widetilde{\boldsymbol{T}_0}$, and its inverse $\widetilde{\boldsymbol{T}_0}^{-1}$:

$$\widetilde{\boldsymbol{T}_0} = \begin{bmatrix} 1 & 0 & 0 & O_{0x} \\ 0 & 1 & 0 & O_{0y} \\ 0 & 0 & 1 & O_{0z} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \widetilde{\boldsymbol{T}_0}^{-1} = \begin{bmatrix} 1 & 0 & 0 & -O_{0x} \\ 0 & 1 & 0 & -O_{0y} \\ 0 & 0 & 1 & -O_{0z} \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Scaling matrix:

$$\widetilde{\boldsymbol{S}} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & \frac{\|\boldsymbol{E}-\boldsymbol{O}\|}{\|\boldsymbol{E}_0-\boldsymbol{O}_0\|} & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For rotation matrices we use analogous notation to the one used for translation matrices: $\boldsymbol{R}_0$ is an auxiliary rotation representing an initial bone pose, so using $\boldsymbol{R}_0$ and $\boldsymbol{R}_0{}^{-1}$ we can align it with the global frame to perform scaling. The rotation matrix $\boldsymbol{R}$ denotes actual rotation caused by the given bone.

Below we describe the procedure for calculation of rotation matrix $\boldsymbol{R}$. Let $\boldsymbol{u}$ and $\boldsymbol{v}$ be the normalized vectors of the bone and its parent:

$$\boldsymbol{u} = \frac{\boldsymbol{E} - \boldsymbol{O}}{\|\boldsymbol{E} - \boldsymbol{O}\|},$$

$$\boldsymbol{v} = \frac{\boldsymbol{E}^p - \boldsymbol{O}^p}{\|\boldsymbol{E}^p - \boldsymbol{O}^p\|},$$

The formulas for obtaining and composing quaternions make use of vector operations, for which we use standard notation, where "·" is the inner product, "×" is the cross product. The unnormalized quaternion $\boldsymbol{q}^{lu}$ of the local rotation from $\boldsymbol{u}$ to $\boldsymbol{v}$ within the plane defined by $\boldsymbol{u}, \boldsymbol{v}$ (2D rotation) is obtained using $\boldsymbol{u}, \boldsymbol{v}$ in the following way:

$$\boldsymbol{q}^{lu} = (1 + \boldsymbol{u} \cdot \boldsymbol{v}; \boldsymbol{u} \times \boldsymbol{v}),$$

thus

$$\boldsymbol{q}^{lu} = (1 + u_x v_x + u_y v_y + u_z v_z, \quad u_y v_z - u_z v_y, \quad u_x v_z - u_z v_x, \quad u_x v_y - u_y v_x).$$

We work with normalized quaternions, thus normalized local rotation quaternion $\boldsymbol{q}^l$ is given by:

$$\boldsymbol{q}^l = \frac{\boldsymbol{q}^{lu}}{\|\boldsymbol{q}^{lu}\|}.$$

For 3D rotation, one would need to further rotate the local rotation by the quaternion representing the twist angle r_{twist}, defined as follows:

$$\boldsymbol{q}_{twist} = (\cos(r_{twist}), \ 0, \ \sin(r_{twist}), \ 0).$$

Composition of the local and the parent rotations: the parent frame rotations are computed recursively from the root bones, hence the transformation of a given bone depends on the parameters of its ancestors. The formula for the composition (product) of quaternions is as follows:

$$\begin{aligned} \boldsymbol{q} = \boldsymbol{q}^l \boldsymbol{q}^p &= (q_0^l, \boldsymbol{q}_{1:3}^l)(q_0^p, \boldsymbol{q}_{1:3}^p) = (q_0^l, \boldsymbol{a}^l)(q_0^p, \boldsymbol{a}^p) \\ &= (q_0^l q_0^p - \boldsymbol{a}^l \cdot \boldsymbol{a}^p, \quad q_0^l \boldsymbol{a}^p + q_0^p \boldsymbol{a}^l + \boldsymbol{a}^l \times \boldsymbol{a}^p). \end{aligned}$$

The rotation matrix corresponding to a quaternion $\boldsymbol{q}$ is given by:

$$\widetilde{\boldsymbol{R}}(\boldsymbol{q}) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) & 0 \\ 2(q_1 q_2 + q_0 q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2 q_3 - q_0 q_1) & 0 \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

To compute the whole affine transformation we first need to rotate the bone to align with global axes and translate back to the global origin since the scaling happens along the axes in a bone local coordinate frame. We use the above-mentioned inverse of the initial rest pose rotation/translation, $\widetilde{\boldsymbol{R}_0}^{-1}, \widetilde{\boldsymbol{T}_0}^{-1}$. After performing the scaling, we apply $\widetilde{\boldsymbol{T}_0}, \widetilde{\boldsymbol{R}_0}$ back. Note the sequence of transformations is applied from the last to the first, e.g. $\widetilde{\boldsymbol{A}_2}\widetilde{\boldsymbol{A}_1}\widetilde{\boldsymbol{x}}$ applies $\widetilde{\boldsymbol{A}_1}$ then $\widetilde{\boldsymbol{A}_2}$ to $\widetilde{\boldsymbol{x}}$, since $\widetilde{\boldsymbol{A}_2}\widetilde{\boldsymbol{A}_1}\widetilde{\boldsymbol{x}} = \widetilde{\boldsymbol{A}_2}(\widetilde{\boldsymbol{A}_1}\widetilde{\boldsymbol{x}})$. Thus the scaling transformation is given by: $\widetilde{\boldsymbol{T}_0}\widetilde{\boldsymbol{R}_0}\widetilde{\boldsymbol{S}}\widetilde{\boldsymbol{R}_0}^{-1}\widetilde{\boldsymbol{T}_0}^{-1}$.

The above formula is a transformation that only performs the desired scaling transformation along the local axes. We then need to extend it by applying the current rotation and current translation. This is done after applying each of the initial rotation and translation transformations. Summing up, we perform inverse of translation and rotation to align with the global axes and the origin, current scaling, initial rotation, current rotation, initial translation back to the previous position and finally current translation, as given by:

$$\widetilde{\boldsymbol{A}} = \widetilde{\boldsymbol{T}}\widetilde{\boldsymbol{T}_0}\widetilde{\boldsymbol{R}}\widetilde{\boldsymbol{R}_0}\widetilde{\boldsymbol{S}}\widetilde{\boldsymbol{R}_0}^{-1}\widetilde{\boldsymbol{T}_0}^{-1}.$$

A.1.6 Calculating the Vertex Weights

We assign a set of vertices to each bone which should be rigid wrt. translation and rotation of a given bone, i.e. strictly move with the bone. Note such vertices can still be stretched as these can be scaled in all the three axes. The larger the distance between rigid parts, the smoother the shape deformations are. Some parts might be more flexible than others, for example a human arm would be rigid along the bones, while the teapot parts such as the handle should be flexible. Figure A.7 shows how the specification of the rigid parts influences the deformations.

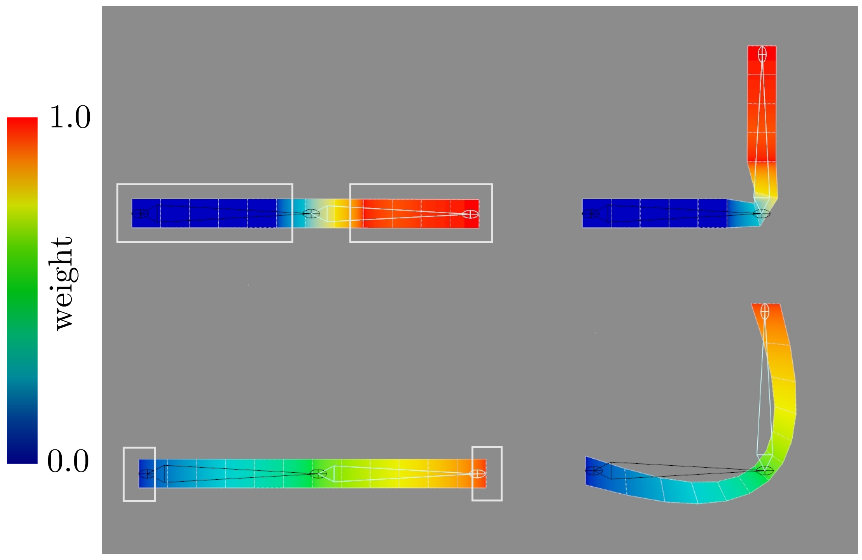


Figure A.7: Two cases of the same object and the skeleton with different rigid parts. The white frames denote rigid vertices belonging to the two identical bones. On the right is shown the same shape when the right bone is rotated by 90 degrees. The mesh colours denote the influence of this bone on the given vertices (red = 1.0, blue = 0.0). The deformation is interpolated between the rigid parts, so the top stick bends only in the middle, while the bottom one bends through all its length.

Rigid vertices have a weight 1.0 assigned to a given bone. The input for the algorithm that calculates the weights is a mesh with the skeleton and the assignment of rigid vertices to each bone. The output are weights of all vertices. Vertices between rigid parts are automatically assigned weights changing quadratically in a distance. The distance is a distance through edges (the shortest path between a vertex to a rigid vertex of a given bone), so as the vertices which are not connected but close to each other do not move together.

The weights blend from one rigid part to the other one, allowing smooth transformations, where the weight is *proportional to the inverse of the distance squared, and normalized*. For example consider a vertex which is between the rigid vertices of two bones. If the distances to the rigid vertices are 1 and 2, the weights are 0.8 and 0.2 respectively (as $\frac{1}{1^2} : \frac{1}{2^2} = 0.8 : 0.2$). If the distance is the same, both weights are 0.5.

A vertex can be assigned (with a non-zero weight) to several bones if the rigid vertices of these bones can be reached through a path consisting only of vertices not assigned to any rigid part. The distance of the m-th vertex to the b-th bone is denoted by $dist(m,b)$, the set of bones reachable from the m-th vertex through edges that do not contain any rigid vertices is denoted by $R(m)$. The weight of the b-th bone on the vertex m-th is given by:

$$w_m^b = \frac{\frac{1}{dist(m,b)^2}}{\sum_{b \in R(m)} \frac{1}{dist(m,b)^2}}.$$

A.1.7 Preparation of the Deformable Mesh

The mesh instances are kept in the `.obj` format, which is a standard format for triangulated 3D meshes, while our own deformable mesh format called (`.dm`) is composed of the single deformable mesh (also as `.obj`), skeleton and the weights. The preparation process is as follows:

1. Prepare in Blender a 3D generic class mesh and its skeleton.
2. Select which vertices are rigid for each bone. Recalculate weights through as per the algorithm in Section A.1.6. The main advantage of applying the weights to the vertices in Blender interface is that we can use it to visualize the deformations, i.e. see what happens when a bone is rotated using the recalculated weights and ensure that the shape is deformed correctly.

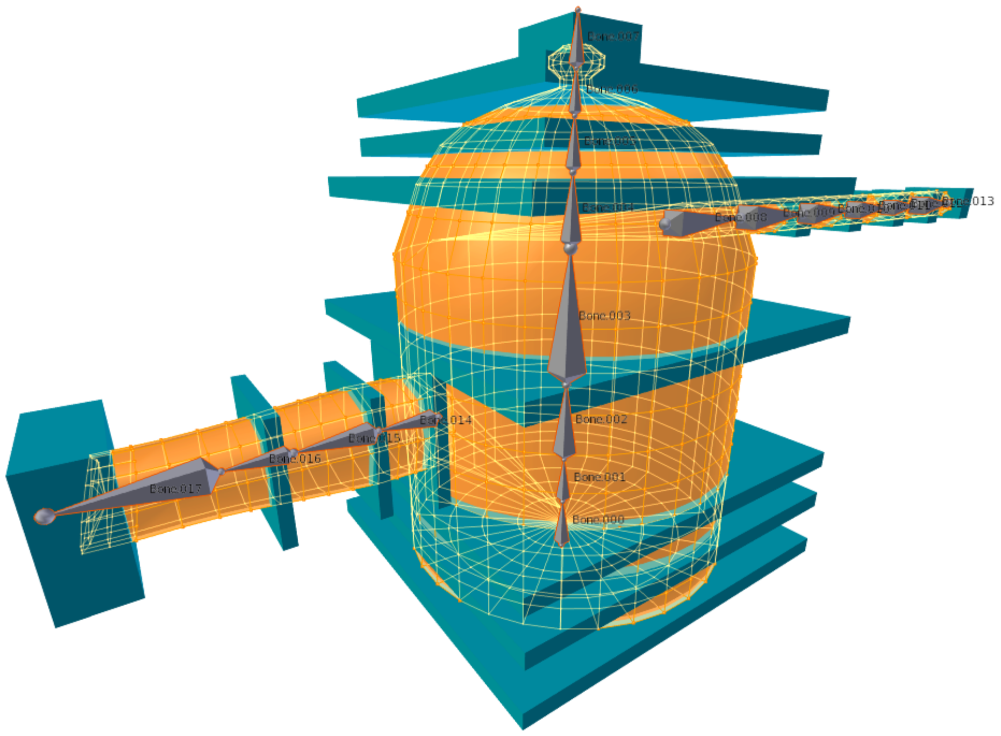


Figure A.8: The teapot generic mesh (orange) with the bones (grey) and their respective rigid vertices located inside the boxes (blue).

3. Export the deformable model to the `.dm`.

Our deformable mesh creation process is simple and allows us to efficiently perform any modifications. The rigid vertices are defined to be these which lie inside a blue box of a given bone, as shown in Figure A.8. The correspondences bone-to-box are obtained by setting matching object names, for example a bone `Bone.005` has rigid vertices in the box named `Cube.005`. This can be straightforwardly extended to a set of boxes, e.g. a bone can influence two symmetric parts of plane wings. If we want to change which vertices should be rigid for a given bone, we only have to move a single box and the weights are recalculated.

A.2 Deformable Mesh Fitting

A.2.1 Fitting a Shape Model to Data

The goal of the fitting is to obtain a mesh similar to a given instance. We then can model the distribution of the skeleton parameters to sample new instances. As outlined earlier, we perform the fitting by moving each joint independently, and also fit the thickness of the bones, while their length is obtained from the joint positions.

For the teapot template mesh the number of vertices M equals 1095. The meshes of the instances are often of a very high quality designed for realistic rendering, composed of tens of thousands of vertices. Since the computation time scales with the product of vertices number in both shapes, we aimed to keep it similar to M, yet slightly more detailed so as not to lose important details such as thickness of a thin handle when reducing the number of vertices. The vertex reduction is done by iteratively merging pairs of nearby vertices with increasing minimal distance threshold. We found out that approximately two thousand vertices provide a good shape approximation, and automatically decreased the number of vertices in small steps until it is below 2200.

Notation: deformable mesh vertices are denoted by $\mathbf{V} = [\mathbf{v}_1, \dots, \mathbf{v}_M] := V(\boldsymbol{\theta})$, the instance vertices (datapoints) are denoted by $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_D]$. The model used for the fitting is a 3-dimensional Gaussian Mixture Model (GMM), which is a weighted sum of M components of Gaussian densities:

$$p(\mathbf{x}_i | \{\mathbf{w}, \boldsymbol{\mu}, \boldsymbol{\Sigma}\}) = \sum_{i=1}^{M} w_m \mathcal{N}(\mathbf{x}_i | \boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m). \tag{A.3}$$

The component means are located at the vertices of the deformable mesh, so $\boldsymbol{\mu} = \mathbf{V}$, while instance vertices $\mathbf{X}$ are the datapoints. All component weights $\mathbf{w}$ are equal to $\frac{1}{M}$. The covariance matrix is spherical, and is common for all the components. Hence there is only one parameter σ (a radius equal to the Gaussian standard deviation) to be chosen ($\boldsymbol{\Sigma} = \sigma^2 \boldsymbol{I}_3$).

The value of σ is set to be approximately a half of the distance between vertices. It has to be sufficiently small to match correctly the surface of the components (e.g. the spout tube sides), but not too small to avoid getting stuck in local minima (the density on the whole surface should be high, not only in the vertices). Another difficulty is that the distances between vertices vary even more when it is deformed. In the experiments we set the object bounding-box volume to 30 units, which leads to approximately 3 to 4 units per object dimension, for which we used $\sigma = 0.05$. Hence σ is for example approximately between $\frac{1}{80}$ and $\frac{1}{60}$ of the teapot height.

Thus, the density at the point $\mathbf{x}_i$ is:

$$p(\mathbf{x}_i|\boldsymbol{\theta}) = \sum_{m=1}^{M} \frac{1}{M} \mathcal{N}(\mathbf{x}_i|\mathbf{v}_m, \sigma^2 \boldsymbol{I}_3). \tag{A.4}$$

We assume that the datapoints are generated i.i.d. by the GMM:

$$p(\mathbf{X}|\boldsymbol{\theta}) = \prod_{d=1}^{D} p(\mathbf{x}_d|\boldsymbol{\theta}) = \prod_{d=1}^{D} \sum_{m=1}^{M} \frac{1}{M} \mathcal{N}(\mathbf{x}_d|\mathbf{v}_m, \sigma^2 \boldsymbol{I}_3). \tag{A.5}$$

Maximizing the likelihood of the given model would produce a density that covers the datapoints, however, there could be also areas of density far away from the datapoints, since the lack of the datapoints in not treated as evidence. The fitting should be symmetric – the instance should cover the deformable mesh, and the deformable mesh cover the instance, see Revow et al. (1996). Hence, we fit both the deformable mesh to the instance and the instance to the deformable mesh. The second model is also a GMM with the same covariance matrix where the component means are now located at the vertices of the instance. This model is analogous to that above, i.e.:

$$p(V(\boldsymbol{\theta})|\mathbf{X}) = \prod_{m=1}^{M} p(\mathbf{v}_m|\mathbf{X}) = \prod_{m=1}^{M} \sum_{d=1}^{D} \frac{1}{D} \mathcal{N}(\mathbf{v}_m|\mathbf{x}_d, \sigma^2 \boldsymbol{I}_3). \tag{A.6}$$

We again assume that both sets of vertices were generated independently. The log-likelihood of both models are as follows:

$$\ell(\boldsymbol{\theta}) = \log p(\mathbf{X}|\boldsymbol{\theta}) = \sum_{d=1}^{D} \log \sum_{m=1}^{M} \frac{1}{M} \mathcal{N}(\mathbf{x}_d|\mathbf{v}_m, \sigma^2 \boldsymbol{I}_3). \tag{A.7}$$

$$\ell(\mathbf{X}) = \log p(V(\boldsymbol{\theta})|\mathbf{X}) = \sum_{m=1}^{M} \log \sum_{d=1}^{D} \frac{1}{D} \mathcal{N}(\mathbf{v}_m|\mathbf{x}_d, \sigma^2 \boldsymbol{I}_3). \tag{A.8}$$

The datapoints $\mathbf{X}$ are an approximation of the instance shape and their number can be chosen. We let the model scale the influence of both likelihood terms of the likelihood independently from the number of vertices chosen. The number of datapoints of the instance, D, can vary, since we can run the algorithm with different levels of detail (also each instance has a similar but different number of vertices after the pre-processing). Therefore, we introduce a weighting factor λ which includes the ratio $\frac{M}{D}$ of the number of the datapoints, making the likelihood of each model to have the same influence, irrespective of the chosen D (M is constant for a given class).

For $M = D$ we obtain $\frac{M}{D} = 1$, so the model for the equal numbers of vertices in both meshes is symmetric. In addition, since the vertices cannot match exactly, we weight the preference for having the whole instance covered with outlying vertices of the deformable mesh than not covered at all, denoted by ω. We set $\omega = 2.0$, which is a preference for covering the whole instance, even with some outlying vertices, so:

$$\lambda = \omega \cdot \frac{M}{D}. \tag{A.9}$$

We perform the MAP estimation by maximizing the joint log-posterior of vertices being generated by both GMMs. The above model incorporates parameters needed for fitting in general, and for a given class. For general parameters, we use only two parameters: (σ, ω). Additional parameters are used within the log-prior probability of the skeleton parameters, $\log p(\boldsymbol{\theta})$, which is given in Equation A.11.

Finally, the log-posterior incorporating λ is:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \left(\lambda \log p(\mathbf{X}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) + \log p(V(\boldsymbol{\theta})|\mathbf{X})\right). \tag{A.10}$$

Figure A.9 presents the results of the fitting with partial and full model, with overlaid instance mask (transparent magenta). In the left plot we can notice that all the instance vertices were explained, but often by significant stretching of the parts. For example the spout intersects the whole teapot body so a part of the spout is visible near the handle. Moreover, the end of the handle is outlying, while the top of the teapot is reversed. The mesh in the right plot is deformed to match the instance properly while maintaining the usual teapot shape.

A.2.2 Skeleton Parameters Priors

A.2.2.1 Class-General Part

To perform the fitting, we define how the mesh can be deformed by setting the degrees of freedom of each bone: we choose a subset of the skeleton parameters that can be changed. The rest of the parameters are constant (values defined by the skeleton). We also define constraints on how the parameters can be changed, and a global translation parameter which moves the whole deformable mesh.

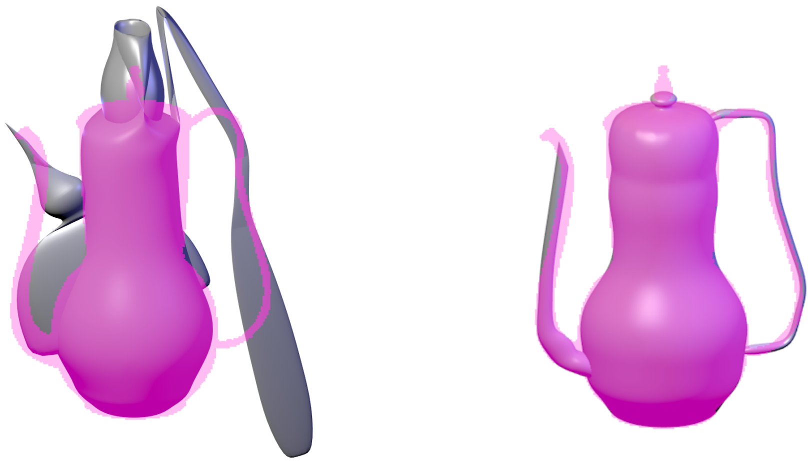


Figure A.9: The deformable mesh fitted to the instance (transparent magenta mask) without the second-way term and without the skeleton prior (left, maximizing Equation A.7) and using the whole model (right, maximizing Equation A.10).

We aim to keep the bones rotation-free relative to their parent, penalising deformations from the rest pose. Rotation above 180 degrees is forbidden, but for the teapot class the rotations above 90 degrees are also considered as strange and unlikely, as we use several bones to approximate each object part. The rotation prior distribution is common for all the bones, which is set to be $Beta(\alpha_r, \beta_r) = Beta(20, 20)$, scaled to the range [-180, 180] degrees. The mode is for 0 degrees, the pdf for 60 degrees is 10 times lower than for 0 degrees, for 90 degrees it is 200 times lower. The scaling factor priors: $Beta(\alpha_s, \beta_s) = Beta(1.5, 5.5)$ scaled to the range $[0, 10]$. The mode for scaling equals 1. These priors are applied independently to each of the three dimensions.

Note that to set the hyper-parameters of the *Beta* distributions above we had to consider only one degree of freedom, since the mode is fixed. The mode is at the configuration that is at the rest pose, so rotation-free and without extra scaling: 0 degrees for rotation, scale 1 for scaling factors. We keep the above prior distributions common for all the bones, while we model the actual strength of the prior using separate class-specific parameters, called bone stiffness: λ^r for rotations, and λ^s for scaling, which we describe in the next section.

A.2.2.2 Class-Specific Part: Teapot

The skeleton lies within a plane, so the joints are moved in 2D. Thus in our case we allow for variable y-axis and z-axis positions of the joints and all of the scaling parameters. In addition the teapot body joints lie on a vertical line – the body cannot rotate.

The assumption of the data being generated independently by a large number of Gaussian generators requires setting the weights of the priors. The stiffness values are denoted by λ_b^r for rotation of the b-th bone by the angle α^b relative to the parent, and λ_s^b for the scaling factor of a given bone in each of the local axes (i.e. s_x^b, s_y^b, s_z^b). We assume that all the priors for parameters of each bone are independent, hence the full log-prior probability is as follows:

$$\log p(\boldsymbol{\theta}) = \sum_{b=1}^{B} \left[\lambda_r^b \log p(\alpha^b) + \lambda_s^b \sum_{d \in \{x,y,z\}} \log p(s_d^b) \right]. \tag{A.11}$$

Table A.1 presents the values of the stiffness parameters chosen for the teapot bones. The stiffness values were chosen manually by investigating the convergence outcomes to produce reasonable fits, note that this was done for each object part independently, so we could find the right parameters quickly. The stiffness of the body is large, since the body bones cover a large number of vertices (the body cannot rotate, hence there is no rotation prior). The stiffness of the spout is large as well, since the spout should be quite straight. On the other hand, the handle has to be very flexible, able to easily model a C-shaped handle requiring significant rotation of bones relative to the parent. Therefore, only the handle root bone is stiff (so it does not rotate to align with the teapot body), all the remaining bones are very flexible.

Object part (# of bones)	λ_r	λ_s
Body (8)	-	400
Handle (6)	20 (root) | 2 (other)	40
Spout (4)	80	100

Table A.1: Teapot priors

The teapot class bone stiffness hyper-parameters.

A.2.3 Fitting Experiments

Fitting is performed by maximizing log-posterior (Equation A.10) using the `scipy.minimize` L-BFGS-B method. The algorithm employs Automatic Differentiation to evaluate exact gradients during optimization, which is superior to both symbolic differentiation in terms of speed and capabilities of differentiating complex algorithm executions, and numerical differentiation in terms of the accuracy.

Our dataset consists of 36 teapot examples that feature a high variability of object shape, collected from free on-line 3D repositories such as 3DWarehouse[1] and 3Dmdb[2]. The orientation of teapot meshes was prepared to be the same. We use two different initializations as we treat two types of handles as separate teapot classes, Type 1 is a C-shaped handle, and Type 2 is a handle located over the top of the teapot. See Figure A.11 for comparison of the two types. To perform the fitting, in general we would require a dataset annotated with which object parts are present in the given instance so as we know which parts we need to instantiate and fit to.

Initialization close to the actual shape is important, since the fitting will otherwise end up in some local maximum where deformable teapot parts would not match the same parts of the instance. The spout and the handle are initialized to their usual position. Both the spout and the handle are shifted farther away so as they do not merge with the teapot body of wider teapots at the beginning of the fitting process, see the first column (initialization) in Figure A.10. The spout top is set to point upwards. The handle is initialized to the half of its thickness so it can easier match detailed thin handles.

Fitting process:

1. Load the deformable model file and a given instance; set the equal size of both meshes: the equal bounding boxes volumes. The centres of both bounding-boxes are set to be the same.
2. Decrease the number of vertices in the instance.
3. Fit the deformable model to the 3D instance, stop when the gradient norm is below a given threshold (typically after 50-100 iterations).
4. Export the skeleton parameters.

[1]https://3dwarehouse.sketchup.com
[2]https://3dmdb.com/

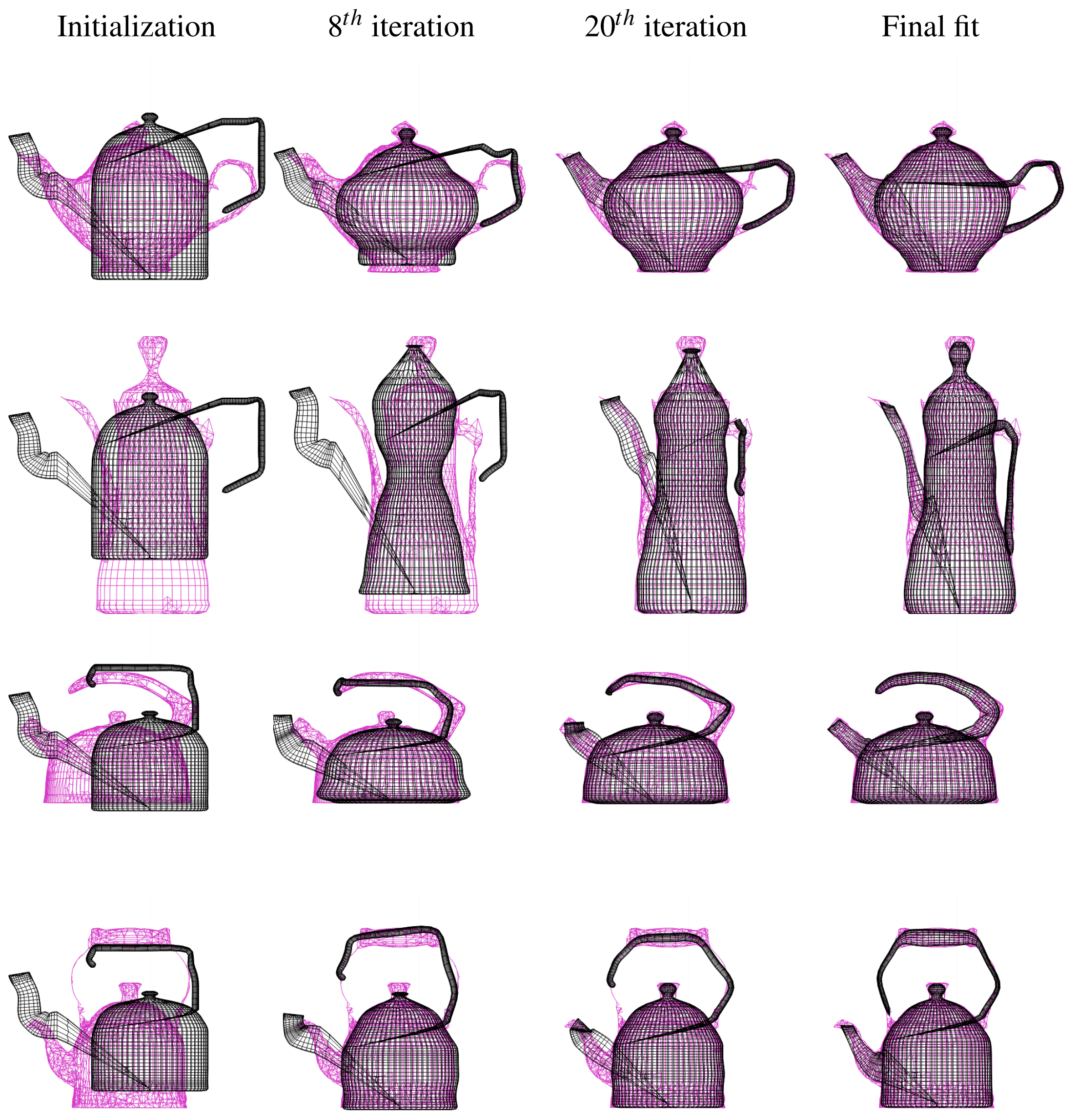


Figure A.10: Fitting iterations: the first frame is the initialization, after setting equal volumes of the bounding-boxes. The next frames are at the 8, 20, and the final iteration. Each row presents fitting of the deformable mesh (a black wireframe) to a given instance (in purple). The first two rows present fitting of the teapot with C-shaped handle (Type 1), while the next two rows present the initialization where the teapot handle is located at the top (Type 2).

Figure A.10 presents iterations of the fitting process for four different instances. The deformable mesh is able to properly explain all the teapot parts of variable shapes and poses. In the third row the initialized black model is not aligned well with the given purple instance, but is able to first move to align the centres, then the bent spout stretches properly, and the handle thickness increases to accurately fit to the given shape.

More examples of the fitting procedure are given in the video at:
`https://youtu.be/dQzWcx0lMFo`

A.2.4 Implementation

We use Python for implementation, which is also in-built in Blender and the Blender API is in Python as well. This means that one can write instructions directly in Blender interface which is designed to perform any kind of operations that are available in Blender using Python scripts. Such scripts are widely used to automate the work, some examples are: a method that stochastically generates a grass mesh on an arbitrary surface according to the parametrization of the grass blade shapes and poses, or the method that generates a human mesh where one can vary parameters responsible for different properties of the human body.

We use the `Autograd` package to perform automatic differentiation of the log-posterior. Automatic differentiation can compute a derivative of the whole algorithm execution. Firstly the value of a function in a given point is computed. Then, having known the actual computation flow, including the chosen if-statements paths, number of loop executions etc., the computation of the derivative of the value returned by the function with respect to the function parameters can be performed. The gradient can be then used to perform optimization.

Automatic differentiation can easily handle complex executions, as bone poses depend on the parent frames, hence the whole deformation and differentiation is recursive from the root bones to the leaves. Moreover, a given vertex position is a weighted average of the affines of several bones, while the affines are obtained by calling recursively several sub-functions. Note we have re-implemented several standard functions for transformations and operations on quaternions to comply with the `Autograd` package requirements.

One of the main difficulties to overcome was the issue that each fitting iteration consists of computing approximately 5 million distance-pairs ($2 \times 2.2\text{k} \times 1.1\text{k}$), which would not be an issue by itself, but the computation of the derivatives of these 5 million distances wrt. the bone parameters in each iteration had a large impact on the speed.

Therefore, the optimization of the execution time was crucial, it was possible to implement it efficiently in pure Python by using the `Numpy` package broadcasting feature (which executes internally the code in C) within the `Autograd` package methods. This allowed to perform one fitting iteration in approximately 3 seconds on a standard CPU, and the whole fitting to one instance in approximately 4 minutes.

A.2.5 Results

Figure A.11 below presents the results of the fitting of the deformable mesh for four instances. For each pair, the left render is the instance, on the right the result of the fitting. The skeleton of the deformable model is able to properly fit to very different shapes, for example for bottom left pair it can properly model the much thicker middle part of the handle, while for the bottom right pair the mesh properly explains the thin and wide handle.

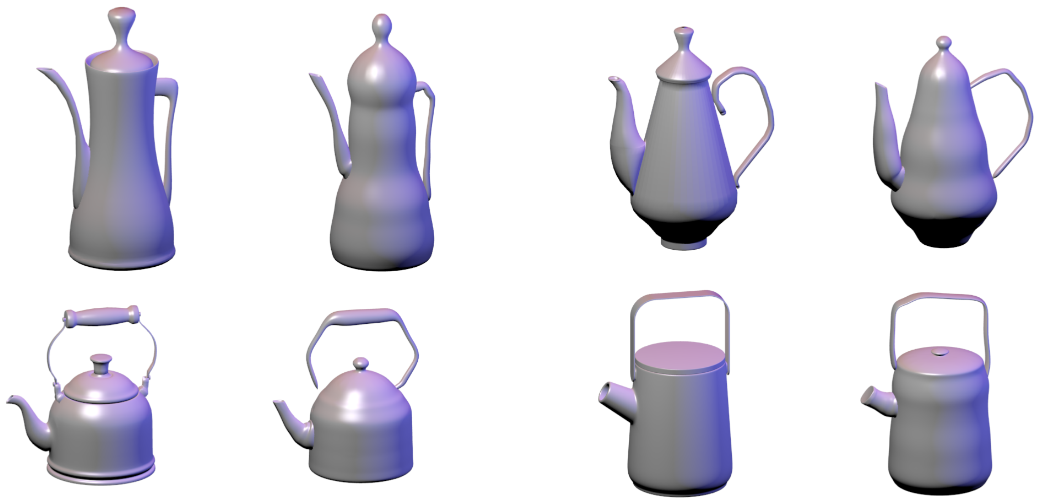


Figure A.11: Results of the fitting for four teapots, for each pair the left mesh is the instance, and the right is the fitted deformable mesh. Top row: Type 1, bottom row: Type 2.

A.3 Generative Model of Skeleton Parameters

A.3.1 Shape Model

Skeleton parameters are kept in a T-dimensional vector denoted by $\boldsymbol{\theta}$, where $\boldsymbol{\theta}^P$ are the joint parameters (joint positions in a given axis) and $\boldsymbol{\theta}^S$ are the scaling parameters (positive scaling factors of bones):

$$\boldsymbol{\theta} = (\boldsymbol{\theta}^P, \boldsymbol{\theta}^S), \tag{A.12}$$

The training set of skeleton parameters is obtained from the fitting process described in Section A.2. These are unconstrained parameters, i.e. the ones that are not kept constant. For a given object class from the unconstrained parameters we can recover the full skeleton parameter representation, so the deformable mesh vertices are given by:

$$\mathbf{V} = V(\boldsymbol{\theta}). \tag{A.13}$$

The scaling factors are transformed to their logarithms using a function denoted by F, this is a function that takes $\boldsymbol{\theta}$ as input and applies the logarithm only to the scaling factors $\boldsymbol{\theta}^S$ entries and the identity to $\boldsymbol{\theta}^P$ entries. The logarithm is used to make the domain of the scaling factors unbounded, this in particular allows to add Gaussian noise to the scaling factors.

Afterwards we perform standardisation: for each entry the mean is subtracted and divided by the standard deviation estimated using the dataset obtained during the fitting process. The standardisation is obtained by application of a function denoted by S, i.e:

$$\widetilde{\boldsymbol{\theta}} = S(F(\boldsymbol{\theta})), \quad \widetilde{\boldsymbol{\theta}} \in \mathbb{R}^T, \tag{A.14}$$

thus the deformable mesh vertices are given by:

$$\mathbf{V} = V(F^{-1}(S^{-1}(\widetilde{\boldsymbol{\theta}}))). \tag{A.15}$$

For the generative model of $\widetilde{\boldsymbol{\theta}}$ we use Probabilistic Principal Component Analysis (PPCA) by Tipping and Bishop (1999). The transformed parameters $\widetilde{\boldsymbol{\theta}}$ are modelled by K-component PPCA parametrized by $(\mathbf{W} \in \mathbb{R}^{T \times K}, \boldsymbol{\mu} \in \mathbb{R}^T, \sigma)$:

$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_K), \tag{A.16}$$

$$\widetilde{\boldsymbol{\theta}} \sim \mathcal{N}(\mathbf{W}\mathbf{z} + \boldsymbol{\mu}, \sigma^2 \mathbf{I}_T). \tag{A.17}$$

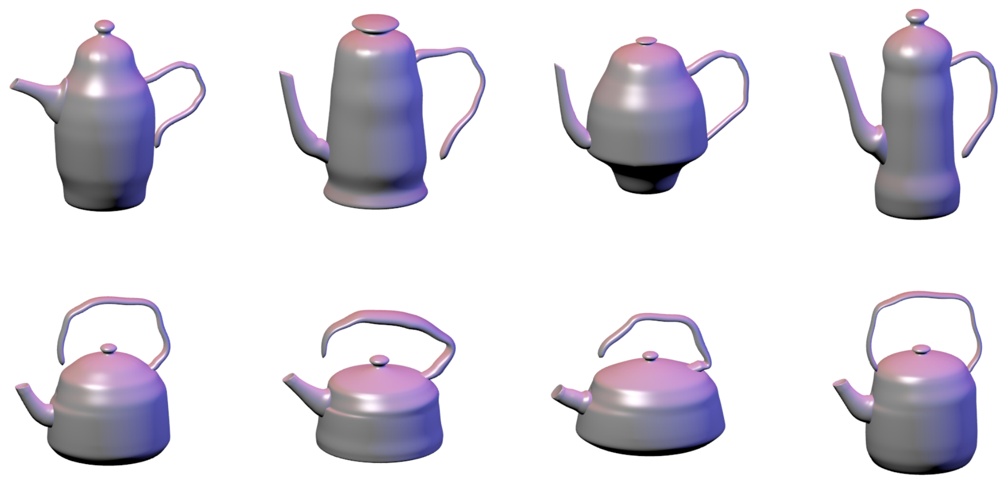


Figure A.12: Sampled teapots using PPCA that reveal a high shape variability, for Type 1 (top row) and Type 2 (bottom row).

We treat teapots with different types of handles as two distinct classes, and so use a separate PPCA model for each class. The resulting data has $T = 69$ dimensions, while the number of examples of two teapot types is 26 and 10. The number of components is chosen so that PPCA explains at least 95% of the variance ($K = 15$ and $K = 7$, respectively).

Sampling novel meshes consists of sampling $\widetilde{\boldsymbol{\theta}}$ directly from the PPCA model. Figure A.12 presents examples of the generated teapots. We can notice that the generated teapot parts are of different shapes, and the model generalizes the shape well since the sampled teapots are significantly different.

A.3.2 Object Completion

In the object completion experiments we are given a portion of a shape: the vertices $\mathbf{X}$, and must generate the most likely shape that fits to the given vertices and completes parts that are not constrained.

In the previous section we described the generative model of the skeleton parameters. Since we fit also to arbitrary shapes, we allow the model to move slightly so it can fit properly even when the centre of the most likely teapot model is not located at the global origin.

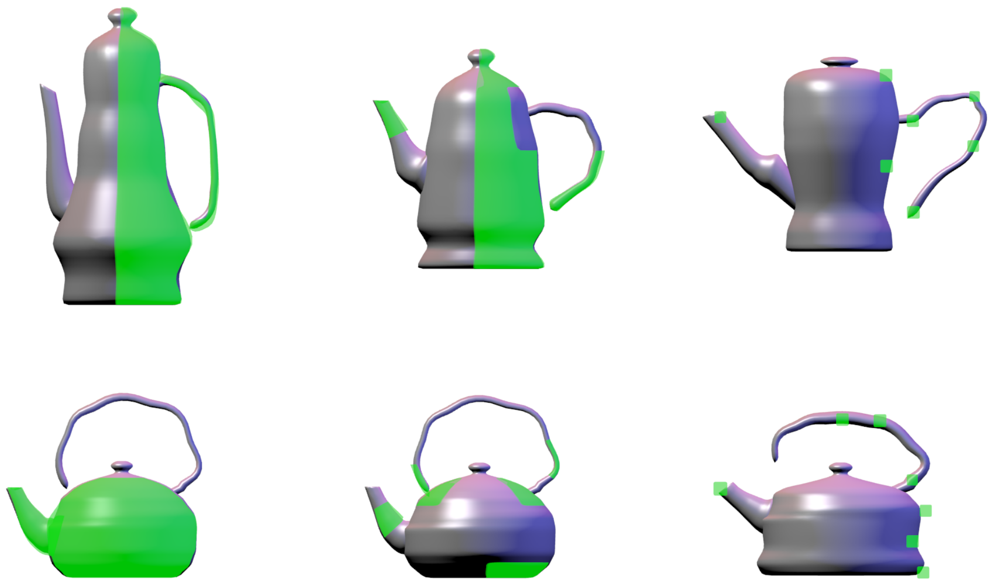


Figure A.13: Completion results (best fit – MAP). Green: a given partial shape, grey: deformable mesh completion. In the first two columns the given shapes are fragments of the sampled teapots; in the third column the shapes are arbitrary cubes.

We also allow the skeleton parameters to contain some level of an additional noise, to better generalize the given object class, i.e. to fit to examples not seen before. Therefore, a Gaussian global translation $\mathbf{t}$ and a Gaussian noise $\boldsymbol{\varepsilon}$ are introduced:

$$\mathbf{t} \sim \mathcal{N}(\mathbf{0}, \tau^2 \mathbf{I}_3), \tag{A.18}$$

$$\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_\varepsilon), \tag{A.19}$$

where $\tau = 1$ and $\boldsymbol{\Sigma}_\varepsilon$ is a diagonal matrix with entries equal to extra hyper-parameters σ_P^2 for joint positions parameters and σ_S^2 for log-scaling parameters, $\sigma_P = 0.04$, $\sigma_S = 0.2$. The m-th deformable mesh vertex including the translation and the noise are given by:

$$\mathbf{V}_m = V(F^{-1}(S^{-1}(\widetilde{\boldsymbol{\theta}}) + \boldsymbol{\varepsilon}))_m + \mathbf{t}. \tag{A.20}$$

The fit is the probability of the data $\mathbf{X}$ given deformable mesh vertices $\mathbf{V}$ using parameters $\widetilde{\boldsymbol{\theta}}$, analogous as in the fitting process:

$$\log p(\mathbf{X}|\widetilde{\boldsymbol{\theta}}) = \sum_{d=1}^{D} \log \sum_{m=1}^{M} \frac{1}{M} \mathcal{N}(\mathbf{x}_d | \mathbf{v}_m, \sigma^2 \boldsymbol{I}_3). \tag{A.21}$$

The completion task log-posterior consists of the fit (weighted using parameter $\lambda = \frac{20.0}{D}$) and probability of $\widetilde{\boldsymbol{\theta}}$, $\boldsymbol{\varepsilon}$, $\mathbf{t}$:

$$\lambda p(\mathbf{X}|\widetilde{\boldsymbol{\theta}}) + \log p(\widetilde{\boldsymbol{\theta}}) + \log p(\boldsymbol{\varepsilon}) + \log p(\mathbf{t}). \tag{A.22}$$

The completion is obtained by MAP optimization of the above equation using L-BFGS-B, facilitated by the gradient obtained from the Automatic Differentiation. We initialize the procedure at the mean teapot of the PPCA model. The log-posterior is jointly optimized w.r.t. $(z, \boldsymbol{\varepsilon}, \mathbf{t})$.

Some completion examples are shown in Figure A.13. We can notice that the model very accurately recovers the original teapots given the fragments of the sampled teapots. For observations consisting of arbitrary cubes (see the rightmost column in Figure A.13) the meshes of the generated teapots pass properly through all the constraints, e.g. in the top right example the teapot handle deforms to pass through the four cubes.

A.3.3 Constrained Sampling

We also perform constrained sampling, which consists in sampling from the completion log-posterior, Equation A.22. To this end we employ the Hamiltonian Monte Carlo algorithm (HMC, Neal 2010), which uses the gradient of the log-posterior density to explore space more effectively. We initialize at the MAP completion result, so the probability at the point of initialization is significantly higher than in the case of the mean teapot. This omits a difficult process of searching for a region of high density by HMC.

Generated constrained samples are presented in Figure A.14. We can notice that all the teapot parts reveal some shape variability, especially for the unconstrained bottom of the teapot base in the top row, while for the remaining parts the object surface still is nearby the imposed constraints.

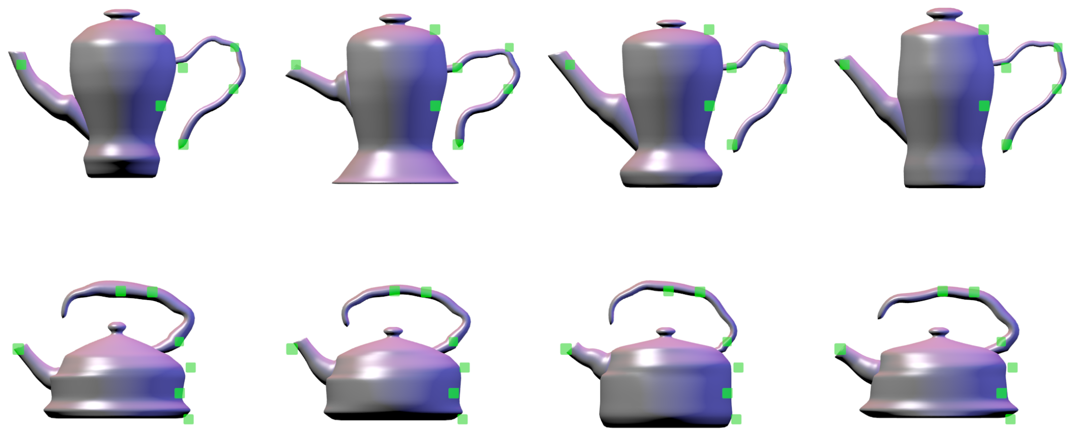


Figure A.14: Constrained sampling results using HMC, for two teapot types. Green: a given shape, grey: deformable mesh.

A.4 Discussion

We have demonstrated a shape model that uses a skeleton representation, which we have thoroughly investigated in various experiments for the teapot class object. Importantly, such a skeleton-based representation is interpretable, as we can obtain properties of object parts or their fragments from the the skeleton pose, and we can edit it, e.g. when one would model a human pose then one could change pose of a particular hand.

The deformation is differentiable, and hence allows to perform the model fitting and other gradient-based procedures. This allowed us to obtain a generative model of the shapes, as well as to perform other tasks such as object completion and shape sampling. Such representation could be used in the VIG problems, where one may want to explain an object by fitting its render to the observed image, by optimization of the deformable mesh parameters, either original parameters or via PPCA components. Such skeleton-based model could be used for example for the purpose of 3D human hand pose reconstruction. One could also sample new object shapes as demonstrated using PPCA, together with random pose and size for the purpose of generation of a large amount of synthetic data.

Bibliography

Ahuja, N. and Todorovic, S. (2008). Connected Segmentation Tree - A Joint Representation of Region Layout and Hierarchy. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8.

Allan, M. and Williams, C. K. I. (2009). Object Localization using the Generative Template of Features. *Computer Vision and Image Understanding*, 113:824–838.

Angel, E. (2003). *Interactive Computer Graphics*. Addison Wesley, third edition.

Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Lawrence Zitnick, C., and Parikh, D. (2015). VQA: Visual Question Answering. In *Proceedings of International Conference on Computer Vision (ICCV)*, pages 2425–2433.

Badrinarayanan, V., Kendall, A., and Cipolla, R. (2017). SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 39(12):2481–2495.

Ballard, D. H. (1981). Generalizing the Hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122.

Bansal, A., Russell, B., and Gupta, A. (2016). Marr Revisited: 2D-3D Alignment via Surface Normal Prediction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5965–5974.

Barinova, O., Lempitsky, V., and Kohli, P. (2012). On detection of multiple object instances using Hough transforms. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 34(9):1773–1784.

Bazin, J. C., Seo, Y., Demonceaux, C., Vasseur, P., Ikeuchi, K., Kweon, I., and Pollefeys, M. (2012). Globally optimal line clustering and vanishing point estimation in

Manhattan world. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 638–645.

Blanz, V. and Vetter, T. (2003). Face Recognition Based on Fitting a 3D Morphable Model. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 25(9):1063–1074.

Brock, A., Donahue, J., and Simonyan, K. (2019). Large Scale GAN Training for High Fidelity Natural Image Synthesis. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

Brooks, R. A., Creiner, R., and Binford, T. O. (1979). The ACRONYM Model-based Vision System. In *Proceedings of the 6th International Joint Conference on Artificial Intelligence - Volume 1*, pages 105–113.

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. (2016). InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. In *Advances in Neural Information Processing Systems 29*, pages 2172–2180.

Choi, W., Chao, Y.-W., Pantofaru, C., and Savarese, S. (2015). Indoor Scene Understanding with Geometric and Semantic Contexts. *International Journal of Computer Vision (IJCV)*, 112(2):204–220.

Cootes, T. F., Edwards, G. J., and Taylor, C. J. (2001). Active Appearance Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 23(6):681–685.

Cootes, T. F., Taylor, C. J., Cooper, D. H., and Graham, J. (1995). Active Shape Models - Their Training and Application. *Computer Vision and Image Understanding*, 61(1):38–59.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1:886–893.

Davies, R. H., Twining, C. J., Cootes, T. F., Waterton, J. C., and Taylor, C. J. (2002). 3D Statistical Shape Models Using Direct Optimisation of Description Length. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–20. Springer.

Deserno, M. (2004). How to generate equidistributed points on the surface of a sphere. *Technical Report, Max-Planck-Institut fur Polymerforschung*.

Dickinson, S. (2009). The Evolution of Object Categorization and the Challenge of Image Abstraction. *Object Categorization: Computer and Human Vision Perspectives*.

Esalmi, S. M. A., Heess, N., Weber, T., Tassa, Y., Kavukcuoglu, K., and Hinton, G. E. (2016). Attend, Infer, Repeat: Fast Scene Understanding with Generative Models. In *Advances in Neural Information Processing Systems 29*, pages 3225–3233.

Eslami, S. A., Rezende, D. J., Besse, F., Viola, F., Morcos, A. S., Garnelo, M., Ruderman, A., Rusu, A. A., Danihelka, I., Gregor, K., et al. (2018). Neural Scene Representation and Rendering. *Science*, 360(6394):1204–1210.

Eslami, S. M. A., Heess, N., Weber, T., Tassa, Y., Kavukcuoglu, K., and Hinton, G. E. (2016). Attend, Infer, Repeat: Fast Scene Understanding with Generative Models. In *Advances in Neural Information Processing Systems 29*.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2010). The PASCAL Visual Object Classes (VOC) challenge. *International Journal of Computer Vision (IJCV)*, 88(2):303–338.

Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D. (2010). Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 32(9):1627–1645.

Fergus, R., Perona, P., and Zisserman, A. (2003). Object class recognition by unsupervised scale-invariant learning. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2:264–271.

Gall, J., Yao, A., Razavi, N., Van Gool, L., and Lempitsky, V. (2011). Hough Forests for Object Detection, Tracking, and Action Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 33(11):2188–2202.

Genova, K., Cole, F., Maschinot, A., Sarna, A., Vlasic, D., and Freeman, W. T. (2018). Unsupervised Training for 3D Morphable Model Regression. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8377–8386.

Girshick, R. (2015). Fast R-CNN. In *Proceedings of the 2015 IEEE International Conference on Computer Vision*, pages 1440–1448.

Grenander, U. (1976). *Lectures in Pattern Theory: Vol. 1 Pattern Synthesis*. Springer-Verlag.

Grenander, U. (1978). *Lectures in Pattern Theory: Vol. 2 Pattern Analysis*. Springer-Verlag.

Grimson, W. E. L. and Lozano-Perez, T. (1984). Model-based recognition and localization from sparse range or tactile data. *The International Journal of Robotics Research*, 3(3):3–35.

Hariharan, B., Arbelaez, P., Girshick, R., and Malik, J. (2016). Object Instance Segmentation and Fine-Grained Localization Using Hypercolumns. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 39(4):627–639.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask R-CNN. In *The IEEE International Conference on Computer Vision (ICCV)*, pages 2961–2969.

He, X., Zemel, R. S., and Carreira-Perpiñán, M. Á. (2004). Multiscale conditional random fields for image labeling. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2:695–703.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. (2017). beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Hoiem, D., Efros, A. A., and Hebert, M. (2008). Putting objects in perspective. *International Journal of Computer Vision (IJCV)*, 80(1):3–15.

Hoiem, D., Stein, A. N., Efros, A. A., and Hebert, M. (2007). Recovering Occlusion Boundaries from an Image. In *Proceedings of the 2007 IEEE International Conference on Computer Vision*, pages 1–8.

Hosang, J., Benenson, R., and Schiele, B. (2016). A convnet for non-maximum suppression. In *German Conference on Pattern Recognition (GCPR), 2016*.

Hough, P. V. C. (1962). Method and means for recognizing complex patterns. *U.S. Patent 3069654*.

Hu, G., Yan, F., Kittler, J., Christmas, W., Chan, C. H., Feng, Z., and Huber, P. (2017). Efficient 3D Morphable Face Model Fitting. *Pattern Recognition*, 67:366–379.

Huang, G. B., Mattar, M., Berg, T., and Learned-Miller, E. (2008). Labeled Faces in the Wild: A Database for StudyingFace Recognition in Unconstrained Environments.

Huang, S., Qi, S., Zhu, Y., Xiao, Y., Xu, Y., and Zhu, S.-C. (2018). Holistic 3D Scene Parsing and Reconstruction from a Single RGB Image. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 187–203.

Izadinia, H., Shan, Q., and Seitz, S. M. (2017). IM2CAD. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5134–5143.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive Mixtures of Local Experts. *Neural Computation*, 3:79–87.

Jampani, V., Nowozin, S., Loper, M., and Gehler, P. V. (2015). The Informed Sampler: A Discriminative Approach to Bayesian Inference in Generative Computer Vision Models. *Computer Vision and Image Understanding*, 136:32–44.

Kalogerakis, E., Averkiou, M., Maji, S., and Chaudhuri, S. (2017). 3d shape segmentation with projective convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Kanazawa, A., Tulsiani, S., Efros, A. A., and Malik, J. (2018). Learning Category-Specific Mesh Reconstruction from Image Collections. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 371–386.

Kar, A., Tulsiani, S., Carreira, J., and Malik, J. (2015). Amodal completion and size constancy in natural scenes. In *Proceedings of the 2015 IEEE International Conference on Computer Vision*, pages 127–135.

Kendall, A., Grimes, M., and Cipolla, R. (2015). PoseNet: A convolutional network for real-time 6-DOF camera relocalization. In *Proceedings of the 2015 IEEE International Conference on Computer Vision*, pages 2938–2946.

Kim, H. and Mnih, A. (2018). Disentangling by Factorising. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 2654–2663.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. In *International Conference on Learning Representations (ICLR).*

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105.

Kuipers, J. B. (1999). *Quaternions and rotation sequences*, volume 66. Princeton Univ. Press.

Kulkarni, T. D., Kohli, P., Tenenbaum, J. B., and Mansinghka, V. (2015a). Picture: A probabilistic programming language for scene perception. In *Proc CVPR*, pages 4390–4399.

Kulkarni, T. D., Whitney, W. F., Kohli, P., and Tenenbaum, J. B. (2015b). Deep convolutional inverse graphics network. In *Advances in Neural Information Processing Systems 28*, pages 2539–2547.

Kulkarni, T. D., Whitney, W. F., Kohli, P., and Tenenbaum, J. B. (2015c). Picture: A probabilistic programming language for scene perception. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4390–4399.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. (1990). Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems 2*, pages 396–404.

Lee, J.-K. and Yoon, K.-J. (2015). Real-time joint estimation of camera orientation and vanishing points. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1866–1874.

Leibe, B., Leonardis, A., and Schiele, B. (2004). Combined Object Categorization and Segmentation With An Implicit Shape Model. In *ECCV workshop on statistical learning in computer vision*, pages 17–32.

Li, B., Dai, Y., and He, M. (2018a). Monocular Depth Estimation with Hierarchical Fusion of Dilated CNNs and Soft-Weighted-Sum Inference. *Pattern Recognition*, 83:328–339.

Li, Y., Wang, G., Ji, X., Xiang, Y., and Fox, D. (2018b). DeepIM: Deep Iterative Matching for 6D Pose Estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 683–698.

Liu, Z., Luo, P., Wang, X., and Tang, X. (2015). Deep Learning Face Attributes in the Wild. In *Proceedings of International Conference on Computer Vision (ICCV)*.

Loper, M., Mahmood, N., Romero, J., Pons-Moll, G., and Black, M. J. (2015). SMPL: A Skinned Multi-Person Linear Model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, 34(6):248:1–248:16.

Loper, M. M. and Black, M. J. (2014). OpenDR: An Approximate Differentiable Renderer. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 154–169. Springer.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. *International Conference on Computer Vision, 1999*, 2:1150–1157.

Manhardt, F., Kehl, W., Navab, N., and Tombari, F. (2018). Deep Model-Based 6D Pose Refinement in RGB. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 800–815.

Marr, D. and Nishihara, H. K. (1978). Representation and recognition of the spatial organization of three-dimensional images. *Proceedings of the Royal Society of London, Series B*, 200:269–294.

Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., and Geiger, A. (2019). Occupancy Networks: Learning 3D Reconstruction in Function Space. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4460–4470.

Moreno, P., Williams, C. K. I., Nash, C., and Kohli, P. (2016). Overcoming Occlusion with Inverse Graphics. In *ECCV 2016 Workshops Proceedings Part III*, pages 170–185. Springer. LNCS 9915.

Nam, H. and Han, B. (2016). Learning Multi-Domain Convolutional Neural Networks for Visual Tracking. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4293–4302.

Neal, R. M. (2010). MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 54:113–162.

Neubeck, A. and Van Gool, L. (2006). Efficient non-maximum suppression. In *Proceedings of the 2006 IEEE Conference on Pattern Recognition*, pages 850–855. IEEE.

Oechsle, M., Mescheder, L., Niemeyer, M., Strauss, T., and Geiger, A. (2019). Texture

Fields: Learning Texture Representations in Function Space. In *The IEEE International Conference on Computer Vision (ICCV)*.

Ramamoorthi, R. (2006). Modeling Illumination Variation with Spherical Harmonics. In *Face Processing: Advanced Modeling Methods*, pages 385–424. ACM.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Redmon, J. and Farhadi, A. (2017). YOLO9000: Better, Faster, Stronger. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7263–7271.

Revow, M., Williams, C. K., and Hinton, G. E. (1996). Using Generative Models for Handwritten Digit Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(6):592–606.

Romaszko, L., Williams, C. K. I., Moreno, P., and Kohli, P. (2017). Vision-as-Inverse-Graphics: Obtaining a Rich 3D Explanation of a Scene from a Single Image. In *ICCV 2017 Geometry Meets Deep Learning Workshop*, pages 851–859.

Romaszko, L., Williams, C. K. I., and Winn, J. (2020). Learning Direct Optimization for Scene Understanding. *Pattern Recognition*.

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

Russell, B. C., Torralba, A., Murphy, K. P., and Freeman, W. T. (2008). Labelme: A database and web-based tool for image annotation. *International Journal of Computer Vision (IJCV)*, 77(1-3):157–173.

Satkin, S., Rashid, M., Lin, J., and Hebert, M. (2015). 3DNN: 3D Nearest Neighbor. *International Journal of Computer Vision (IJCV)*, 111(1):69–97.

Schönborn, S., Egger, B., Morel-Forster, A., and Vetter, T. (2017). Markov Chain Monte Carlo for Automated Face Image Analysis. *International Journal of Computer Vision (IJCV)*, 123(2):160–183.

Sharp, T., Keskin, C., Robertson, D., Taylor, J., Shotton, J., Kim, D., Rhemann, C., Leichter, I., Vinnikov, A., Wei, Y., et al. (2015). Accurate, robust, and flexible real-time hand tracking. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 3633–3642. ACM.

Shi, X., Zhou, K., Tong, Y., Desbrun, M., Bao, H., and Guo, B. (2007). Mesh Puppetry: Cascading Optimization of Mesh Deformation with Inverse Kinematics. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, New York, NY, USA. ACM.

Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., and Blake, A. (2011). Real-Time Human Pose Recognition in Parts from a Single Depth Image. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

Silberman, N., Hoiem, D., Kohli, P., and Fergus, R. (2012). Indoor Segmentation and Support Inference from RGBD Images. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 746–760.

Simonyan, K. and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. In *International Conference on Learning Representations (ICLR).*

Song, S. and Xiao, J. (2014). Sliding shapes for 3d object detection in depth images. In *Proceedings of the 2014 European Conference on Computer Vision*, pages 634–651. Springer.

Stephens, R. S. (1990). A probabilistic approach to the Hough Transform. In *Proceedings of the British Machine Vision Conference, (BMVC)*, pages 1–6.

Stevens, M. R. and Beveridge, J. R. (2001). *Integrating Graphics and Vision for Object Recognition*. Kluwer Academic Publishers, Boston.

Su, H., Qi, C. R., Li, Y., and Guibas, L. J. (2015). Render for CNN: Viewpoint Estimation in Images Using CNNs Trained With Rendered 3D Model Views. In *Proceedings of the 2015 IEEE International Conference on Computer Vision.*

Sun, X., Wu, J., Zhang, X., Zhang, Z., Zhang, C., Xue, T., Tenenbaum, J. B., and Freeman, W. T. (2018). Pix3D: Dataset and Methods for Single-Image 3D Shape Modeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2974–2983.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9.

Szeliski, R. (2010). *Computer Vision: Algorithms and Applications*. Springer-Verlag New York, Inc., 1st edition.

Tan, T.-N., Sullivan, G. D., and Baker, K. D. (1998). Model-based localisation and recognition of road vehicles. *International Journal of Computer Vision (IJCV)*, 27(1):5–25.

Tipping, M. E. and Bishop, C. M. (1999). Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622.

Torralba, A. (2003). Contextual priming for object detection. *International Journal of Computer Vision (IJCV)*, 53(2):169–191.

Tran, L. and Liu, X. (2018). Nonlinear 3D Face Morphable Model. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7346–7355.

Tu, Z. and Bai, X. (2009). Auto-Context and Its Application to High-Level Vision Tasks and 3D Brain Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 32(10):1744–1757.

Tulsiani, S., Kar, A., Carreira, J., and Malik, J. (2016). Learning Category-Specific Deformable 3D Models for Object Reconstruction. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*.

Tulsiani, S., Zhou, T., Efros, A. A., and Malik, J. (2017). Multi-view Supervision for Single-view Reconstruction via Differentiable Ray Consistency. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2626–2634.

Uijlings, J. R., Van De Sande, K. E., Gevers, T., and Smeulders, A. W. (2013). Selective Search for Object Recognition. *International Journal of Computer Vision (IJCV)*, 104(2):154–171.

Van den Oord, A., Kalchbrenner, N., Espeholt, L., Vinyals, O., Graves, A., et al. (2016). Conditional Image Generation with PixelCNN Decoders. In *Advances in Neural Information Processing Systems 29*, pages 4790–4798.

Wang, T. Y., Su, H., Huang, Q., Huang, J., Guibas, L., and Mitra, N. J. (2016). Unsupervised Texture Transfer from Images to Model Collections. *ACM Trans. Graph.*, 35(6):177:1–177:13.

Williams, C. K. I., Revow, M., and Hinton, G. E. (1997a). Instantiating deformable models with a neural net. *Computer Vision and Image Understanding*, 68(1):120–126.

Williams, C. K. I., Revow, M., and Hinton, G. E. (1997b). Instantiating deformable models with a neural net. *Computer Vision and Image Understanding*, 68(1):120–126.

Wu, J., Tenenbaum, J. B., and Kohli, P. (2017a). Neural Scene De-rendering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 699–707.

Wu, J., Wang, Y., Xue, T., Sun, X., Freeman, B., and Tenenbaum, J. (2017b). MarrNet: 3D Shape Reconstruction via 2.5D Sketches. In *Advances in Neural Information Processing Systems 30*, pages 540–550.

Yao, S., Hsu, T.-M. H., Zhu, J.-Y., Wu, J., Torralba, A., Freeman, W. T., and Tenenbaum, J. B. (2018). 3D-Aware Scene Manipulation via Inverse Graphics. In *Advances in Neural Information Processing Systems 31*.

Yildirim, I., Kulkarni, T. D., Freiwald, W. A., and Tenenbaum, J. B. (2015). Efficient analysis-by-synthesis in vision: A computational framework, behavioral tests, and comparison with neural representations. In *Thirty-Seventh Annual Conference of the Cognitive Science Society*.

Yuille, A. and Kersten, D. (2006). Vision as Bayesian inference: analysis by synthesis? *Trends in Cognitive Science*, 10(7):301–308.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and understanding convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 818–833. Springer.

Zou, C., Guo, R., Li, Z., and Hoiem, D. (2019). Complete 3D scene parsing from an RGBD image. *International Journal of Computer Vision (IJCV)*, 127(2):143–162.