Learning to Generate 3D Training Data

by

Dawei Yang

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2020

Doctoral Committee:

Professor Jia Deng, Co-Chair, Princeton University
Assistant Professor David F. Fouhey, Co-Chair
Professor Vineet R. Kamat
Professor Benjamin Kuipers

Dawei Yang

ydawei@umich.edu

ORCID iD: 0000-0003-0799-5234

To all who struggle during this dynamic time

ACKNOWLEDGMENTS

First of all, I would like to thank my parents, Yang Zhengwang and Weng Donghua, who have been supporting me unconditionally since I started kindergarten. Further, I would like to thank my advisor Prof. Jia Deng, for his enthusiasm for the projects, for his continuous support and guidance during this research. I also wish to express my appreciation for Prof. Shi-Min Hu and Prof. Kun Xu, who advised me during my undergraduate years. Finally, special thanks to all members of the dissertation committee, Prof. Jia Deng, Prof. David Fouhey, Prof. Vineet Kamat and Prof. Benjamin Kuipers, for thoughtful comments and recommendations on this dissertation.

TABLE OF CONTENTS

Chapter

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Human-level visual 3D perception ability has long been pursued by researchers in computer vision, computer graphics, and robotics. Recent years have seen an emerging line of works using synthetic images to train deep networks for single image 3D perception. Synthetic images rendered by graphics engines are a promising source for training deep neural networks because it comes with perfect 3D ground truth for free. However, the 3D shapes and scenes to be rendered are largely made manual. Besides, it is challenging to ensure that synthetic images collected this way can help train a deep network to perform well on real images. This is because graphics generation pipelines require numerous design decisions such as the selection of 3D shapes and the placement of the camera.

In this dissertation, we propose automatic generation pipelines of synthetic data that aim to improve the task performance of a trained network. We explore both supervised and unsupervised directions for automatic optimization of 3D decisions. For supervised learning, we demonstrate how to optimize 3D parameters such that a trained network can generalize well to real images. We first show that we can construct a pure synthetic 3D shape to achieve state-of-the-art performance on a shape-from-shading benchmark. We further parameterize the decisions as a vector and propose a hybrid gradient approach to efficiently optimize the vector towards usefulness. Our hybrid gradient is able to outperform classic black-box approaches on a wide selection of 3D perception tasks. For unsupervised learning, we propose a novelty metric for 3D parameter evolution based on deep autoregressive models. We show that without any extrinsic motivation, the novelty computed from autoregressive models alone is helpful. Our novelty metric can consistently encourage a random synthetic generator to produce more useful training data for downstream 3D perception tasks.

CHAPTER 1

Introduction

1.1 Background

1.1.1 Single Image 3D Perception with Deep Neural Networks

We capture our 3D world into digital images, store them on our digital devices such as cameras, phones, and computers. We print them out, view them on our display, or share them online with others. Though these images encode the 3D of the scene, they are represented in the form of 2D matrices of color values, or pixels. For humans, we are naturally good at understanding the 3D structure and recovering the 3D content inside, regardless of their 2D form. However, for computers, human-level visual 3D perception ability is not for free; such ability has long been pursued by researchers in a wide variety of fields, including computer vision, computer graphics, and robotics.

In prior research, single image 3D perception generally relies on monocular cues in the image, such as shading, texture or edges. Early studies focus on those cues for 3D surface reconstruction, also known as shape-from-X. For example, one can estimate the surface normal direction from how texture patterns vary across the 2D image [32]. These studies of monocular cues base the assumptions on our knowledge of the real world, such as assumptions of smooth surfaces, uniform painting, and natural illumination [8].

Recent years have seen an emerging line of literature using data-driven approaches, specifically deep neural networks, with significant progress on the task of single image 3D recovery. Deep neural networks do away from manual specification of priors. Instead, they automatically learn the prior knowledge from a dataset of images with 3D ground truth. This largely avoids the manual design of optimization objectives and algorithms, and instead take advantage of large-scale datasets. Deep networks have shown great promise in recovering 3D from images, with increasing state-of-the-art performance on standard benchmarks [38, 74, 82, 70, 53, 44, 81, 113, 24, 71, 116, 176, 59].

1.1.2 3D Representations in Deep Learning

For single image 3D perception, the 3D representation can take many different forms. Especially for deep learning, researchers typically cater to their needs. Here we summarize common 3D representations used in this dissertation.

3D meshes Meshes are compact surface representation describing the surface geometry using polygon faces. In deep learning, 3D polygon meshes are prevalently used for rendering synthetic images because of the easy integration with graphics rendering pipelines. It is also a standard representation in 3D modeling. Therefore, 3D mesh datasets [17, 179, 27, 62, 147] are collected for synthesizing images to train or evaluate 3D perception tasks for deep neural networks.

Depth, normal, reflectance and shading Maps For deep neural networks, 2D maps of depth and normal are common options for representing 3D structures of scenes. They are obtained by mapping the normal vector and the depth of corresponding 3D surface points at each pixel to a 2D matrix. This structured 2D representation is similar to a 2D image, so it is easier to design neural network architectures suited for pixelwise prediction. A large body of work uses this representation for designing deep neural networks for 3D tasks [38, 82, 70, 53, 44, 113, 24, 116, 59]. Compared to recovering the full 3D surface from a single image, the depth and normal maps only include the surface points that appear in the 2D image, and thus avoids the need for hallucinating the occluded parts.

Another related representation is intrinsic images, which are the reflectance and the shading components of a single image. The reflectance map contains the material color of the objects at each pixel mapped into a 2D image. Similarly, the shading map is the accumulation of lighting on the surface at each pixel. In simplified settings, the reflectance only depends on the intrinsic material properties of the objects and does not depend on external illumination; the shading is uniquely determined by the surface geometry and external illumination. Though reflectance and shading do not directly represent a 3D surface, a 3D surface can be further extracted from those intrinsic components. Therefore, they also share the common goal of 3D perception, and frequently appear in related literature [137, 177, 47, 9, 61, 88, 135, 79, 173, 134, 78].

Implicit functions For 3D surface representation, an implicit function $F(x, y, z)$ takes a 3D coordinate (x, y, z) as input and outputs a real value. The surface is represented by the zero level set of the function $F(x, y, z)$. The shape surface in this form is not directly available for rendering, unless special rendering techniques are developed [55], or the surface is explicitly extracted using algorithms such as marching cubes [85]. Recently in deep learning, implicit functions are represented using a deep neural network, sometimes conditioned on the input image for single image 3D recovery [83, 168, 96, 26, 105, 50].

Constructive solid geometry Constructive solid geometry (CSG) represents a shape using a symbolic tree of primitive shapes [43]. The leaf nodes represent primitive shapes such as cylinders, cubes, spheres, *etc*. The non-leaf nodes specify the boolean operations such as union, intersection and difference on their children. The final shape is computed by recursively applying the boolean operations from children to parents. Under the assumption that the objects can be composed by a set of simple primitives, researchers have also studied parsing CSG for single shape recovery [182, 136, 36, 149].

1.1.3 Training Datasets for Single Image 3D

Large-scale datasets such as ImageNet [35], Microsoft COCO [80] are essential for the success of deep neural networks in classification and object recognition tasks. The class label and object bounding boxes are relatively easy to annotate. However, for single image 3D, the task requires dense prediction of 3D, so sufficiently annotated 3D ground truth is often required for supervised learning of deep networks. The main approaches for acquiring 3D ground truth approximately fall into several categories.

1.1.3.1 Manual Capturing of Real World

Depth scanned by sensors is a valid source for collecting dense 3D ground truth. A number of datasets follow this line and capture RGB-D images using Kinect or LiDAR [129, 138, 48, 62, 139, 27]. In RGB-D images, the depth map is recorded using the range sensor and then aligned to the corresponding RGB image. For example, the NYU Depth V2 dataset [138] has 407,024 RGB-D frames of indoor rooms collected using a mounted Kinect. SUN RGB-D [139] has 10,000 indoor RGB-D images along with 58,657 human-annotated bounding boxes of 3D objects. For autonomous driving scenarios, The KITTI Vision Benchmark Suite [49] includes RGB-D videos of road scenes along with many other 3D ground truths of stereo matching, optical flow, odometry and semantic segmentation.

While these large-scale datasets have shown their effectiveness in training a deep neural network to perform 3D perception tasks, capturing using sensors may be limited to sensor types and may not result in diverse data. The datasets typically target to specific scenarios such as autonomous driving [49] or indoor scenes and objects [138, 139].

1.1.3.2 Manual Annotation of Internet Images

Other than sensor capturing, there is also a line of work asking human workers to annotate 3D from the internet images [22, 9, 21]. For example, Chen et al. [21] exploits human's ability of judging relative depth to collect sparse depth pairs in an image. Their followup work [22] designs an efficient UI for collecting sparsely-annotated normals.

Due to limited throughput for human-computer interaction, the annotations are expected to be sparse. In addition, human annotation may be prone to errors so quality assurance is typically needed to ensure the datasets will be relevant for training a deep neural network.

1.1.3.3 Synthetic Rendering using Graphics Engines

Graphics engines are physically accurate simulation of photo-taking in the real world. In the simulation, 3D ground truth such as depth and normal maps, 3D surfaces are readily available without much additional cost. As mentioned earlier, single image 3D perception with deep neural networks rely on provided 3D ground truth for supervision. Synthetic training images rendered by graphics engines perfectly suit this scenario: they come with high-quality ground truth annotations for free, such as pixel-perfect depth, normal and intrinsic images, complete lighting condition and camera parameters. Therefore, synthetic images generated by computer graphics have been extensively used for training deep networks for numerous tasks, including single image 3D reconstruction [139, 58, 95, 62, 171, 17], optical flow estimation [94, 15, 46], human pose estimation [155, 23], action recognition [121], visual question answering [64], and many others [114, 91, 164, 151, 119, 120, 163]. The success of these works has demonstrated the effectiveness of synthetic images.

Compared to manual collection of real images, graphics rendering of synthetic images gives full flexibility and control over the scene to be rendered. This includes the selection of objects in the scene, their poses, the illumination configuration, camera intrinsics and extrinsics. The flexibility in a graphics pipeline allows free designing and modification of every piece of a scene to fit specific needs. In the context of training deep neural networks, the target is to make synthetic images better training data. This means the synthetic images need to be large-scale and relevant to 3D perception tasks.

Most of today's existing synthetic datasets collect 3D assets with manual labor. The idea behind is similar to capturing real images: designs of 3D scenes and collection of 3D shapes can be made manual. Scenes and shapes that are designed by humans are similar to those that exist in the real world, so they are expected to be useful training data for deep neural networks. Take a popular dataset as an example, ShapeNet [17] includes 50,000+ shapes in 55 real-world categories. As for indoor scenes, SUNCG [140] contains 45,622 realistic indoor designs that are crowdsourced from an online platform. Users design artistic floor plans in a house, arrange textured furniture and other objects in the scene. Zhang et al. [175] and Li and Snavely [77] further render them for 3D tasks and show the large-scale datasets can help a network to perform well on real indoor scenes.

Similar datasets have been collected in this way, including SceneNet [95], Falling Things [152] and SceneNN [58].

To increase the number of samples and reduce human cost, we have seen a few attempts on automation of generating synthetic datasets. Manual heuristics can be seen in constructing synthetic datasets, such as [175]. They randomly generate camera locations and angles, and then filter out camera views that produce renderings with minimal pixel variety of object instances. Another example is SceneNet [95], where they use gravity to randomize object arrangements. The randomization increases the variety of the data and may potentially increase the task performance of the trained network.

1.2 Motivation

As we have seen, synthetic images rendered by graphics engines are a great source for training deep neural networks on various 3D perception tasks. However, today's synthetic datasets still involve large manual effort. Same as manual capturing of real images, human collection involves intensive labor and considerable time for large-scale datasets. Besides, the application of synthetic training data is hindered by the reality gap: it is not clear how to make sure that the generated data will be useful for real-world tasks. Since there are only limited instances in the datasets collected this way, it is also not clear how much can be considered enough for training deep neural networks to perform well.

In this dissertation, we consider synthetic rendering pipelines aiming to solve the above problems. Specifically, we are motivated by the flexibility of synthetic scenes: since we have full control of every part of 3D content, we should make full use of this advantage. Our strategy is full automation of the decisions including shape selection, lighting environment, object arrangement and camera placement *etc*., while aiming at improving the quality of the dataset for training deep neural networks.

1.2.1 Automating the Graphics Pipeline

Automating the generation of synthetic data can relieve manual effort of collecting data. In this dissertation, we propose methods that generate synthetic data in a fully automatic fashion, with minimal manual design involved. For each task scenario, we first design a minimal template for it. For example, for single shape from shading, we assume there is one single object and a global illumination model in the scene. For human face reconstruction, we assume a face model is used with directional lighting. For indoor scenes, there can be multiple shape instances with surrounding walls, ceilings and floors. These templates are rather flexible and can be adjusted for different scopes.

After the template is defined, the content of the scene is filled by a synthetic generator. The synthetic generator has a set of parameters that control its behavior. For example, in Chapter 2, the synthetic generator produces a 3D shape for shape-from-shading. In Chapter 3 and 4, the synthetic generator can produce parameters for human face models, object instances, texture material, and indoor layout. The generator can be parameterized as a discrete population of instances (Chapter 2 and 4) or a continuous distribution in a probabilistic model (Chapter 3).

This formulation is powerful: it can theoretically generate endless 3D configurations that are different from one another, thus providing unlimited training samples for a deep neural network. However, they are not all suited for training deep neural networks because it can easily produce poor compositions, such as completely empty images due to incorrect lighting of a scene or bad placement of a camera. Therefore, such images may not be useful for training a deep network as little knowledge or information is included in the training sample.

To solve this problem, we first take a look at a number of domain-specific automatic generation pipelines [112, 172, 63, 156]. These approaches aim at learning to generate synthetic scenes that resemble real-world [63, 86], that are indistinguishable from real images [156], or that deal with human-centric relations among furniture [112].

We also tweak a synthetic data generator, but we reiterate our goal: we hope to generate unlimited training data that will be useful for training deep networks for 3D perception. Towards this goal, we explore two directions in this dissertation: supervised learning and unsupervised learning of a synthetic data generator.

1.2.2 Supervised Learning: Optimization towards Usefulness

We optimize the synthetic generator directly towards usefulness. The usefulness is defined as the generalization performance of a trained deep network, on an external dataset that represents partial observation of the real world. This dataset can be a set of real images that are manually collected. Note that this validation set is not for training the deep neural networks, but for evaluating a trained one, so it does not have to be large-scale.

The overall idea is to tweak the parameters of a synthetic generator, obtain a set of training samples, train a deep neural network on a task and then evaluate it on the external validation set. Once we obtain the generalization performance, it serves as a supervision signal to learn the synthetic generator parameters. Several concurrent works [65, 123] share a similar high-level concept, while the task design and methodology are vastly different. In our work, the supervision signal is passed back as fitness scores to guide the evolution (Chapter 2), or as an optimization loss for gradient-based optimization (Chapter 3).

Based on this idea, in Chapter 2 we first propose an evolutionary method for automatically generating random synthetic 3D shapes. In Chapter 3, we then generalize the formulation to be a probabilistic 3D synthetic content generator controlled by a parameter vector, and accelerate the optimization towards usefulness using what we call “hybrid gradient”.

1.2.3 Unsupervised Learning: Optimization towards Novelty

Supervised learning that involves training and evaluation of a deep network can be costly. Besides, it depends on the task and the validation dataset. In Chapter 4, we hope to build a task-agnostic unsupervised approach that optimizes the synthetic generator using intrinsic motivation. We examine what serves a good training dataset based on the nature of training deep neural networks. A deep neural network is able to perform well when it encounters a test sample it has already seen in the training set. This means a good training dataset should be diverse, covering a wide range of training samples that may exist in the real world.

To this end, we draw inspiration from Novelty Search [73, 161]. In evolutionary robotics, the diversity-rewarding algorithms can prevent the population from converging to homogenous distributions under performance optimization. We bring it to the context of generating synthetic training data: we design a novelty metric that encourages the synthetic generator to produce synthetic images that are different from each other. The diversity constraint constantly drives the synthetic generator to be creative and move away from existing samples, therefore producing a wide range of rendered images as well as 3D configurations.

1.3 Contributions

1.3.1 Learning to Generate 3D Synthetic Shapes through Shape Evolution (Chapter 2)

In this chapter, we address shape-from-shading by training deep networks on synthetic images. We consider constructing pure synthetic shapes that are contrary to manually curated shapes such as ShapeNet [17]. We automate the synthetic shape generation with an evolutionary algorithm that jointly generates 3D shapes and trains a shape-from-shading deep network. We evolve complex shapes entirely from simple primitives such as spheres and cubes. The evolution process is supervised by a usefulness metric, based on generalization performance of a trained deep network.

We demonstrate that a network trained in this way can achieve better performance than previous algorithms, without using any external dataset of 3D shapes.

This chapter is based on a published work:

- D. Yang and J. Deng. Shape from shading through shape evolution. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018

1.3.2 Learning to Generate 3D Synthetic Data through Hybrid Gradient (Chapter 3)

In this chapter, we parameterize the synthetic generation pipeline as a probabilistic 3D content generator. We then optimize the parameter vector of the generator towards the usefulness target. We propose a new method for such optimization, based on what we call "hybrid gradient". The basic idea is to make use of the analytical gradient where they are available, and combine them with black-box optimization for the rest of the function.

We evaluate our approach on the task of estimating surface normal, depth and intrinsic components from a single image. Experiments on standard benchmarks and controlled settings show that our approach can outperform prior methods on optimizing the generation of 3D training data, particularly in terms of computational efficiency.

This chapter is based on a published work:

- D. Yang and J. Deng. Learning to generate synthetic 3d training data through hybrid gradient. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020

1.3.3 Learning to Generate 3D Synthetic Data with a Novelty Metric (Chapter 4)

We consider an unsupervised approach to learning a synthetic data generator. We focus on the intrinsic motivation of a random synthetic data generator—novelty. It means a synthetic generator should always generate a new data sample that looks different from previous ones. Once a novel sample is seen, it is no longer viewed as novel. The novelty motivation constantly pressures the generator to be creative, so that it is able to explore a broader range of synthetic samples.

To define the novelty metric, we make use of an existing autoregressive model, PixelCNN [154, 128], to model the probabilistic distributions of the generated images. The novelty model is updated to include new synthetic images; in turn, the generator is pressured to generate images that are new to the novelty model.

We experiment in two zero-shot scenarios: shape-from-shading of a single object and intrinsic image decomposition of a scene containing multiple objects. In both scenarios, we define two templates for the task, with zero knowledge of the test distribution. We demonstrate that the novelty metric is able to help a random synthetic data generator to produce more useful training data. The network trained on the novelty-guided synthetic data is able to generalize better on unseen images in different test datasets.

This chapter is based on a draft submission for publication:

- D. Yang and J. Deng. Novelty-guided evolution for generating synthetic 3d training data. Manuscript submitted for publication, 2020

CHAPTER 2

Learning to Generate 3D Synthetic Shapes through Shape Evolution[1]

2.1 Introduction

Shape from Shading (SFS) is a classic computer vision problem at the core of single-image 3D reconstruction [174]. Shading cues play an important role in recovering geometry and are especially critical for textureless surfaces.

Traditionally, Shape from Shading has been approached as an optimization problem where the task is to solve for a plausible shape that can generate the pixels under a Lambertian shading model [167, 37, 8, 7, 6]. The key challenge is to design an appropriate optimization objective to sufficiently constrain the solution space, and to design an optimization algorithm to find a good solution efficiently.

In this chapter, we address Shape from Shading by training deep networks on synthetic images. This follows an emerging line of work on single-image 3D reconstruction that combines synthetic imagery and deep learning [144, 95, 92, 175, 119, 148, 28, 166, 15]. Such an approach does away with the manual design of optimization objectives and algorithms, and instead trains a deep network to directly estimate shape. This approach can take advantage of a large amount of training data, and has shown great promise on tasks such as view point estimation [144], 3D object reconstruction and recognition [148, 28, 166], and normal estimation in indoor scenes [175].

One limitation of this data-driven approach, however, is availability of 3D shapes needed for rendering synthetic images. Existing approaches have relied on manually constructed [17, 179, 2]

[1] This chapter is based on a joint work with Jia Deng [171].

or scanned shapes [27]. But such datasets can be expensive to build. Furthermore, while synthetic datasets can be augmented with varying viewpoints and lighting, they are still constrained by the number of distinct shapes, which may limit the ability of trained models to generalize to real images.

An intriguing question is whether it would be possible to do away with manually curated 3D shapes while still being able to use synthetic images to train deep networks. Our key hypothesis is that shapes are compositional and we should be able to compose complex shapes from simple primitives. The challenge is how to enable automatic composition and how to ensure that the composed shapes are useful for training deep networks.

We propose an evolutionary algorithm that jointly generates 3D shapes and trains a shape-from-shading deep network. We evolve complex shapes entirely from simple primitives such as spheres and cubes, and do so in tandem with the training of a deep network to perform shape from shading. The evolution of shapes and the training of a deep network collaborate—the former generates shapes needed by the latter, and the latter provides feedback to guide the former. Our approach is significantly novel compared to prior works that use synthetic images to train deep networks, because they have all relied on manually curated shape datasets [144, 92, 175, 148].

In this algorithm, we represent each shape using an implicit function [117]. Each function is composed of simple primitives, and the composition is encoded as a computation graph. Starting from simple primitives such as spheres and cubes, we evolve a population of shapes through transformations and compositions defined over graphs. We render synthetic images from each shape in the population and use the synthetic images to train a shape-from-shading network. The performance of the network on a validation set of real images is then used to define the fitness score of each shape. In each round of the evolution, fitter shapes have better chance of survival whereas less fit shapes tend to be eliminated. The end result is a population of surviving shapes, along with a shape-from-shading network trained with them. Figure 2.1 illustrates the overall pipeline.

The shape-from-shading network is incrementally trained in a way that is tightly integrated with shape evolution. In each round of evolution, the network is fine-tuned *separately* with each shape in the population, spawning one new network instance per shape. Then the best network instance

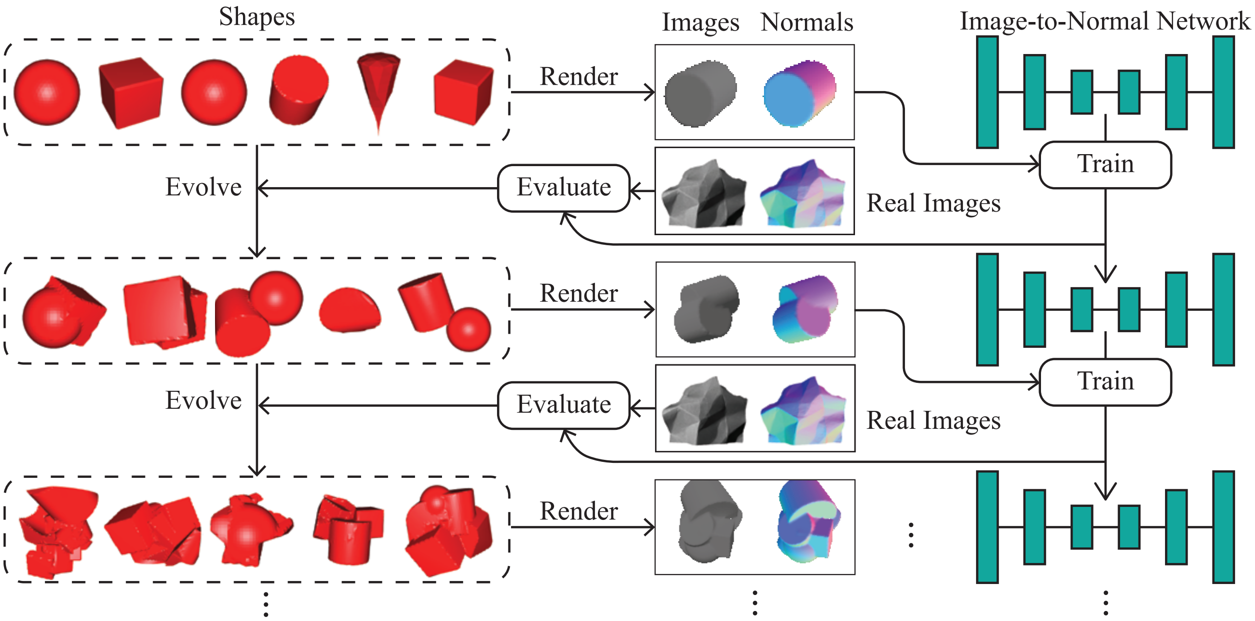


Figure 2.1: The overview of our approach. Starting from simple primitives such as spheres and cubes, we evolve a population of complex shapes. We render synthetic images from the shapes to incrementally train a shape-from-shading network. The performance of the network on a validation set of real images is then used to guide the shape evolution.

advances to the next round while the rest are discarded. In other words, the network tries updating its weights using each newly evolved shape, and the best weights are kept to the next round.

We evaluate our approach using the MIT-Berkeley Intrinsic Images dataset [8]. Experiments demonstrate that we can train a deep network to achieve state-of-the-art performance on real images using synthetic images rendered entirely from evolved shapes, without the help of any manually constructed or scanned shapes. In addition, we present ablation studies which support the design of our evolutionary algorithm.

Our results are significant in that we demonstrate that it is potentially possible to completely automate the generation of synthetic images used to train deep networks. We also show that the generation procedure can be effectively adapted, through evolution, to the training of a deep network. This opens up the possibility of training 3D reconstruction networks with a large number of shapes beyond the reach of manually curated shape collections.

To summarize, our contributions are twofold: (1) we propose an evolutionary algorithm to jointly evolve 3D shapes and train deep networks, which, to the best of our knowledge, is the first time this has been done; (2) we demonstrate that a network trained this way can achieve state-of-the-art performance on a real-world shape-from-shading benchmark, without using any external dataset of 3D shapes.

2.2 Related Work

Recovering 3D properties from a single image is one of the most fundamental problems of computer vision. Early works mostly focused on developing analytical solutions and optimization techniques, with zero or minimal learning [32, 174, 8, 7, 6]. Recent successes in this direction include the SIRFS algorithm by Barron and Malik [8], the local shape from shading method by Xiong *et al*. [167], and "polynomial SFS" algorithm by Ecker and Jepson [37]. All these methods have interpretable, "glass box" models with elegant insights, but in order to maintain analytical tractability, they have to make substantial assumptions that may not hold in unconstrained settings. For example, SIRFS [8] assumes a known object boundary, which is often unavailable in practice. The method by Xiong *et al*. assumes quadratically parameterized surfaces, which has difficulties approximating sharp edges or depth discontinuities.

Learning-based methods are less interpretable but more flexible. Seminal works include an MRF-based method proposed by Hoiem *et al*. [56] and the Make3D [129] system by Saxena *et al*. Cole *et al*. [30] proposed a data-driven method for 3D shape interpretation by retrieving similar image patches from a training set and stitching the local shapes together. Richter and Roth [118] used a discriminative learning approach to recover shape from shading in unknown illumination. Some recent works have used deep neural networks for predicting surface normals [159, 4] or depth [38, 157, 16] and have shown state-of-the-art results.

Learning-based methods cannot succeed without high-quality training data. Recent years have seen many efforts to acquire 3D ground truth from the real world, including ScanNet [33], NYU Depth [138], the KITTI Vision Benchmark Suite [49], SUN RGB-D [139], B3DO [62], and Make3D [129], all of which offer RGB-D images captured by depth sensors. The MIT-Berkeley Intrinsic Images dataset [8] provides real world images with ground truth on shading, reflectance, normals in addition to depth.

In addition to real world data, synthetic imagery has also been explored as a source of supervision. Promising results have been demonstrated on diverse 3D tasks such as pose estimation [144, 92, 2], optical flow [15], object reconstruction [148, 28], and surface normal estimation [175]. Such advances have been made possible by concomitant efforts to collect 3D content needed for rendering. In particular, the 3D shapes have come from a variety of sources, including online CAD model repositories [17, 179], interior design sites [175], video games [119, 120], and movies [15].

The collection of 3D shapes, from either the real world or a virtual world, involves substantial manual effort—the former requires depth sensors to be carried around whereas the latter requires human artists to compose the 3D models. Our work explores a new direction that automatically generates 3D shapes to serve an end task, bypassing real world acquisition or human creation.

Our work draws inspiration from the work of Clune & Lipson [29], which evolves 3D shapes as Compositional Pattern Producing Networks [142]. Our work differs from theirs in two important aspects. First, Clune & Lipson perform only shape generation, particularly the generation of interesting shapes, where interestingness is defined by humans in the loop. In contrast, we *jointly* generate shapes and train deep networks, which, to the best of our knowledge, is the first this has been done. Second, we use a significantly different evolution procedure. Clune & Lipson adopt the NEAT algorithm [143], which uses generic graph operations such as insertion and crossover at random nodes, whereas our evolution operations represent common shape “edits” such as translation, rotation, intersection, and union, which are chosen to optimize the efficiency of evolving 3D shapes.

2.3 Shape Evolution

Our shape evolution follows the setup of a standard genetic algorithm [57]. We start with an initial population of shapes. Each shape in the population receives a fitness score from an external evaluator. Then the shapes are sampled according to their fitness scores, and undergo random geometric operations to form a new population. This process then repeats for successive iterations.

2.3.1 Shape Representation

We represent shapes using implicit surfaces [117]. An implicit surface is defined by a function $F : \mathbb{R}^3 \rightarrow \mathbb{R}$. that maps a 3D point to a scalar. The surface consists of points (x, y, z) that satisfy the equation:

$$F(x, y, z) = 0.$$

And if we define the points $F(x, y, z) < 0$ as the interior, then a solid shape is constructed from this function F. Note that the shape is not guaranteed to be closed, *i.e.* may have points at infinity. A simple workaround is to always confine the points within a cube [29].

Our initial shape population consists of four common shapes— sphere, cylinder, cube, and cone, which can be represented by the functions below:

$$\begin{aligned}
&\text{Sphere :} && F(x,y,z) = x^2 + y^2 + z^2 - R^2 \\
&\text{Cylinder :} && F(x,y,z) = \max\left(\tfrac{x^2+y^2}{R^2}, \tfrac{|z|}{H}\right) - 1 \\
&\text{Cube :} && F(x,y,z) = \max(|x|, |y|, |z|) - \tfrac{L}{2} \\
&\text{Cone :} && F(x,y,z) = \max(\tfrac{x^2+y^2}{R^2} - \tfrac{z^2}{H^2}, -z, z - H)
\end{aligned} \tag{2.1}$$

An advantage of implicit surfaces is that the composition of shapes can be easily expressed as the composition of functions, and a composite function can be represented by a (directed acyclic) computation graph, in the same way a neural network is represented as a computation graph.

Suppose a computation graph $G = (V, E)$. It includes a set of nodes $V = \{x, y, z\} \cup \{v_1, v_2, \cdots\} \cup \{t\}$, which includes three input nodes $\{x, y, z\}$, a variable number of internal nodes $\{v_1, v_2, \cdots\}$, and a single output node t. Each node $v \in V$ (excluding input nodes) is associated with a scalar bias b_v, a reduction function r_v that maps a variable number of real values to a single scalar, and an activation function ϕ_v that maps a real value to a new value. In addition, each edge $e \in E$ is associated with a weight w_e.

It is worth noting that different from a standard neural network or a Compositional Pattern Producing Network (CPPN) that only uses `sum` as the reduction function, our reduction function

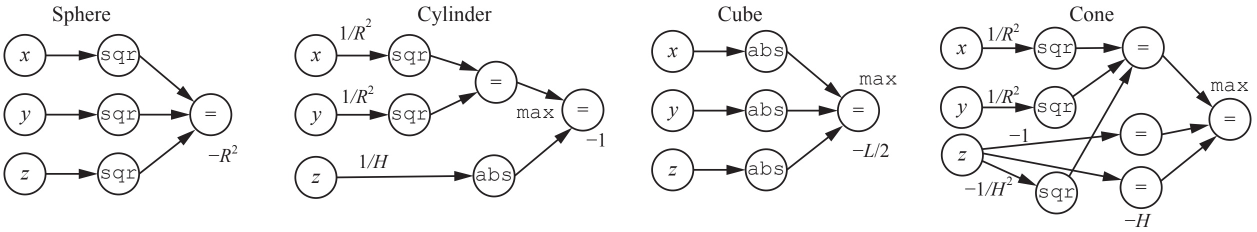


Figure 2.2: The computation graphs of four primitive shapes defined in Equation 2.1. The unlabeled edge weight and node bias are 0, and the unlabeled reduction function is `sum`.

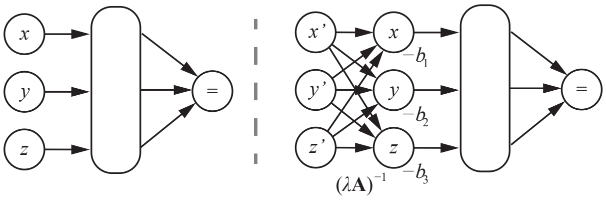


Figure 2.3: Shape transformation represented by graph operation. Left: the graph of the shape before transformation. Right: the graph of the shape after transformation.

can be `sum`, `max` or `min`. As will become clear, this is to allow straightforward composition of shapes.

To evaluate the computation graph, each node takes the weighted activations of its predecessors and applies the reduction function, followed by the activation function plus the bias. Figure 2.2 illustrates the graphs of the functions defined in Equation 2.1.

Shape transformation To evolve shapes, we define graph operations to generate new shapes from existing ones. We first show how to transform an individual shape. Given an existing shape represented by $F(x, y, z)$, let $F(T(x, y, z))$ represent a transformed shape, where $T : \mathbb{R}^3 \to \mathbb{R}^3$ is a 3D-to-3D map. It is easy to verify that $F(T(x, y, z))$ represents the transformed shape under translation, rotation, and scaling if we define T as

$$T(x, y, z) = (\lambda \mathbf{A})^{-1}[x, y, z]^{\mathsf{T}} - \mathbf{b}, \tag{2.2}$$

where $\mathbf{A}$ is a rotation matrix, λ is the scalar, and the $\mathbf{b}$ is the translation vector. Note that for simplicity our definitions have only included a single global scalar, but more flexibility can be easily introduced by allowing different scalars along different axes or an arbitrary invertible matrix $\mathbf{A}$.

This shape transformation can also be expressed in terms of a graph transformation, as illustrated

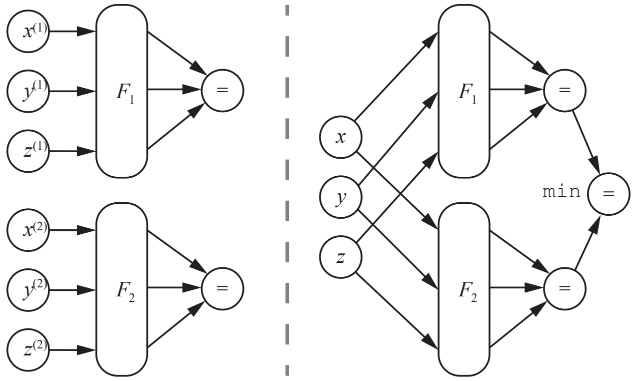


Figure 2.4: The union of two shapes represented by graph merging. Left: the respective graphs of the two shapes to be unioned. Right: the graph of the unioned shape.

in Figure 2.3. Given the original graph of the shape, we insert 3 new input nodes x', y', z' before the original input nodes, connect new nodes to the original nodes with weights corresponding to the elements of the matrix $(\lambda \mathbf{A})^{-1}$, and set the biases of the original nodes to the vector $-\mathbf{b}$.

Shape composition In addition to transforming individual shapes, we also define binary operations over two shapes. This allows complex shapes to emerge from the composition of simple ones. Suppose we have two shapes with the implicit representations $F_1(x, y, z)$ and $F_2(x, y, z)$. As a basic fact [117], the union, intersection, and difference of the two can be represented as follows:

$$\begin{aligned} F_{\text{union}}(x, y, z) &= \min(F_1(x, y, z), F_2(x, y, z)) \\ F_{\text{intersection}}(x, y, z) &= \max(F_1(x, y, z), F_2(x, y, z)) \\ F_{\text{difference}(1,2)}(x, y, z) &= \max(F_1(x, y, z), -F_2(x, y, z)). \end{aligned} \tag{2.3}$$

In terms of graph operations, composing two shapes together corresponds to merging two graphs. As illustrated by Figure 2.4, we merge the input nodes of the two graphs and add a new output node that is connected to the two original output nodes. We set the reduction function (`max`, `min`, or `sum`) and the weights of the incoming edges to the new output node according to the specific composition chosen.

2.3.2 Evolution Algorithm

Our evolution process follows a standard setup. It starts with an initial population of n shapes: $\{s_1, s_2, \cdots, s_n\}$, all of which are primitive shapes described in Equation 2.1. Next, m new shapes ($\{s'_1, s'_2, \cdots, s'_m\}$) are created from two randomly sampled existing shapes (*i.e.* two parent shapes). Specifically, the two parent shapes each undergo a random rotation, a random scaling and a random translation, and are then combined by a random operation chosen from union, intersection and difference to generate a new child shape. Now, the population consists of a total of $n + m$ (n parent shapes and m child shapes). Each shape is then evaluated and given a fitness score, based on which n shapes are selected to form the next population. This process is then repeated to evolve more complex shapes.

Having outlined the overall algorithm, we now discuss several specific designs we introduce to make our evolution more efficient and effective.

Fitness propagation Simply evaluating fitness as a function of individual shape is suboptimal in our case. Our shapes are evolved based on composition, and to generate a new shape requires combining existing shapes. If we define fitness strictly on an individual basis, simple shape primitives, which may be useful in producing more complex shapes, can be eliminated during the early rounds of evolution. For example, suppose our goal is to evolve an implicit representation of a target shape. As the population nears the target shape, smaller and simpler cuts and additions are needed to further refine the population. However, if small, simple shapes, which poorly represent the target shape, have been eliminated, such refinement cannot take place.

We introduce fitness propagation to combat this problem. We propagate fitness scores from a child shape to its parents to account for the fact that a parent shape may not have a high fitness in itself, but nonetheless should remain in the population because it can be combined with others to yield good shapes. Suppose in one round of evolution, we evaluate each of the n existing shapes and m newly composed shapes and obtain $n + m$ fitness scores $\{f_1, \cdots, f_n, f'_1, \cdots, f'_m\}$. But instead of directly assigning the scores, we propagate the m fitness scores of the child shapes back to the parent shapes. A parent shape f_i is assigned the best fitness score obtained by its children and itself:

$$f_i \leftarrow \max\left(\{f'_j : s_i \in \pi(s'_j)\} \cup \{f_i\}\right), \tag{2.4}$$

where $\pi(s'_j)$ is the parents of shape s'_j.

Computational resource constraint Because shapes evolve through composition, in the course of evolution the shapes will naturally become more complex and have larger computation graphs. It is easy to verify that the size of the computational graph of a composed shape will at least double in the subsequent population. Thus without any constraint, the average computational cost of a shape will grow exponentially in the number of iterations as the population evolves, quickly depleting available computing resources before useful shapes emerge. To overcome this issue, we impose a resource constraint by capping the growth of the graphs to be linear in the number of rounds of evolution. If the number of nodes of a computation graph exceeds βt, where β is a hyperparameter and t is time, the graph will be removed from the population and will not be used to construct the next generation of shapes.

Discarding trivial compositions A random composition of two shapes can often result in trivial combinations. For instance, the intersection of shape A and shape B may be empty, and the union of two shapes can be the same as one of the parent. We detect and eliminate such cases to prevent them from slowing down the evolution.

Promoting diversity Diversity of the population is important because it prevents the evolution process from overcommitting to a narrow range of directions. If the externally given fitness score is the only criterion for selection, shapes deemed less fit at the moment tend to go extinct quickly, and evolution can get stuck due to a homogenized population. Therefore, we incorporate a diversity constraint into our algorithm: a fixed proportion of the shapes in the population are sampled not based on fitness, but based on the size of their computation graph, with bigger shapes sampled proportionally less often.

2.4 Joint Training of Deep Network

The shapes are evolved in conjunction with training a deep network to perform shape-from-shading. The network takes a rendered image as input, and predicts the surface normal at each pixel. To train this network, we render synthetic images and obtain the ground truth normals using the evolved shapes.

The network is trained incrementally with a training set that consists of evolved shapes. Let D_i be the training set after the ith iteration of the evolution, and let N_i be the network at the same time. The training set is initialized to empty before the evolution starts, *i.e.* $D_0 = \emptyset$, and the network is initialized with random weights.

In the ith evolution iteration, to compute the fitness score of a shape d in the population, we *fine-tune* the current network N_{i-1} with $D_{i-1} \cup \{d\}$—the current training set plus the shape in consideration—to produce a fine-tuned network N^d_{i-1}, which is evaluated on a validation set of real images to produce an error metric that is then used to define the fitness score of shape d. After we have evaluated the fitness of every shape in the population, we update the training set with the fittest shape d^*_i,

$$D_i = D_{i-1} \cup \{d^*_i\}, \tag{2.5}$$

and set the $N^{d^*}_{i-1}$ as the current network,

$$N_i = N^{d^*}_{i-1}. \tag{2.6}$$

In other words, we maintain a growing training set for the network. In each evolution iteration, for each shape in the population we evaluate what would happen if we add the shape to the training set and continue to train the network with the new training set. This is done for each shape in the population separately, resulting in as many new network instances as there are shapes in the current population. The best shape is then officially added to the training set, and the corresponding

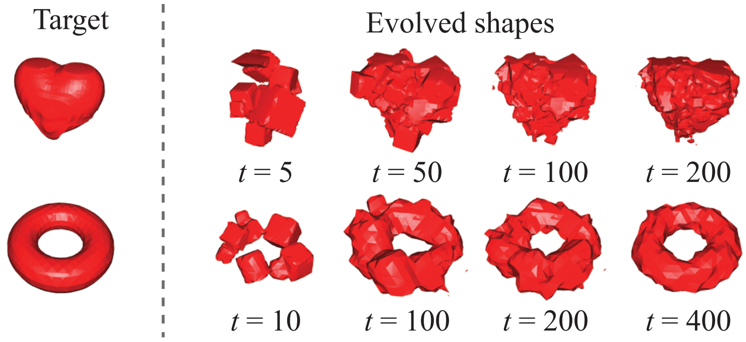


Figure 2.5: Evolution towards a target shape. Left: targets. Right: The fittest shapes in the population as the evolution progresses at different iterations.

fine-tuned network is also kept while the other network instances are discarded.

2.5 Experiments

2.5.1 Standalone Evolution

We first experiment with shape evolution as a standalone module and study the role of several design choices. Similar to [29], we evaluate whether our evolution process is capable of generating shapes close to a given target shape. We define the fitness score of an evolved shape as its intersection over union (IoU) of volume with the target shape.

Implementation details To select the shapes during evolution, half of the population are sampled based on the rank r of their fitness score (from high to low), with the selection probability set to 0.2^r. The other half of the population are sampled based on the rank s of their computation graph size (from small to large), with the relative selection probability set to 0.2^s, in order to maintain diversity. To compute the volume, we voxelize the shapes to $32 \times 32 \times 32$ grids. The population size n is 1000 and the number of child shapes $m = 1000$.

Results We use two target shapes, a heart and a torus. Figure 2.5 shows the two target shapes along with the fittest shape in the population as the evolution progresses. We can see that the evolution is able to produce shapes very close to the targets. Quantitatively, after around 600 iterations, the best IoU of the evolved shapes reaches 94.9% for the heart and 93.5% for the torus.

We also study the effect of the design choices described in Section 2.3.2, including fitness

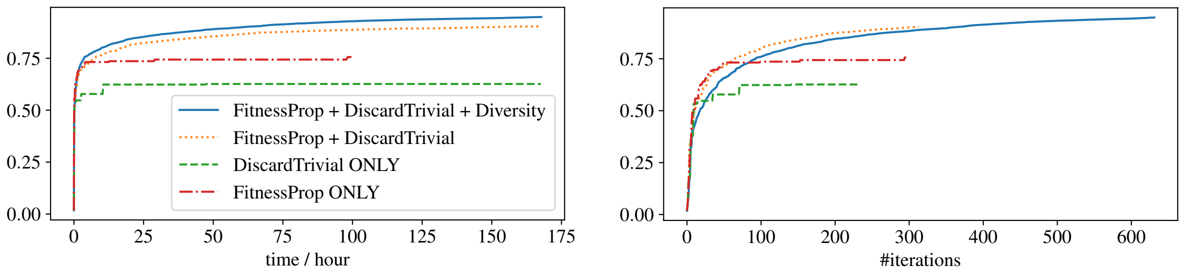


Figure 2.6: The best IoU with the target shape (heart) versus evolution time (top) and the number of iterations (bottom) for different combinations of design choices.

propagation, discarding trivial compositions, and promoting diversity. Figure 2.6 plots, for different combinations of these choices, the best IoU with the target shape (heart) versus evolution time, in terms of both wall time and the number of iterations. We can see that each of them is beneficial and enabling all three achieves fastest evolution in terms of wall time. Note that, the diversity constraint slows down evolution initially in terms of the number of iterations, but it prevents early saturation and is faster in terms of wall time because of lower computational cost in each iteration.

2.5.2 Joint Evolution and Training

We now evaluate our full algorithm that jointly evolves shapes and trains a deep network. We first describe in detail the setup of our individual components.

Setup of network training We use a stacked hourglass network [102] as our shape-from-shading network. The network consists of a stack of 4 hourglasses, with 16 feature channels for each hourglass and 32 feature channels for the initial layers before the hourglasses. In each round of evolution, we fine-tune the network for $\tau = 100$ iterations using RMSprop [150], a batch size of 4, and the mean angle error as the loss function. Before fine-tuning on the new dataset, we re-initialize the RMSprop optimizer.

Rendering synthetic images To render shapes into synthetic images, we use the Mitsuba renderer [60], a physically based photorealistic renderer. We run the marching cubes algorithm [85] on the implicit function of a shape with a resolution of $64 \times 64 \times 64$ to generate the triangle mesh for rendering. We use a randomly placed orthographic camera, and a directional light with a random direction within 60° of the viewing direction to ensure a sufficiently lit shape. All shapes are rendered with diffuse textureless surfaces, along with self occlusion and shadows. In addition to the images, we also generate ground truth surface normals.

Real images with ground truth For both training and testing, we need a set of real-world images with ground truth of surface normals. For training, we need a validation set of real images to evaluate the fitness of shapes, which is defined as how well they help the performance of a shape-from-shading network on real images. For testing, we need a test set of real images to evaluate the performance of the final network.

We use the MIT-Berkeley Intrinsic Image dataset [8, 51] as the source of real images. It includes images of 20 objects captured in a lab setting; each object has two images, one with texture and the other textureless. We use the textureless version of the dataset because our method only evolves shape but not texture. We adopt the official 50-50 train-test split, using the 10 training images as the validation set for fitness evaluation and the 10 test images to evaluate the performance of the final network.

Setup of shape evolution In each iteration of shape evolution, the population size is maintained at $n = 100$, and $m = 100$ new shapes are composed. To select the shapes, 90% of the population are sampled by a roulette wheel where the probability of each shape being chosen is proportional to its fitness score. The fitness score is the reciprocal of the mean angle error on the validation set. The remaining 10% are sampled using the diversity promoting strategy, where the shapes are sampled also based on the rank s of their computation graph size (from small to large), with the relative selection probability set to 0.5^s.

Evaluation protocol To evaluate the shape-from-shading performance of the final network, we use standard metrics proposed by prior work [159, 8]. We measure N-MAE and N-MSE, *i.e.* the mean angle distance (in radians) between the predicted normals and ground-truth normals, and the mean squared errors of the normal vectors. We also measure the fraction of the pixels whose normals are within 11.25, 22.5, 30 degrees angle distance of the ground-truth normals.

Since our network only accepts 128×128 input size but the images in the MIT-Berkeley dataset have different sizes, we pad the images and scale them to 128×128 to feed into the network, and then scale them back and crop to the original sizes for evaluation.

2.5.2.1 Baselines approaches

We compare with a number of baseline approaches including ablated versions of our algorithm. We describe them in detail below.

SIRFS SIRFS [8] is an algorithm with state-of-the-art performance on shape from shading. It is primarily based on optimization and manually designed priors, with a small number of learned parameters. Our method only evolves shapes but not texture, so we compare with SIRFS using the textureless images. Because the published results [8] only textured objects from the MIT-Berkeley Intrinsic Image dataset, we obtained the results on textureless objects using their open source code.

Training with ShapeNet We also compare a baseline approach that trains the shape-from-shading network using synthetic images rendered from an external shape dataset. We use a version of ShapeNet [17], a large dataset of 3D CAD models that consists of approximately 51,300 shapes. We evaluate two variants of this approach.

- *ShapeNet-vanilla* We train a single deep network on the synthetic images rendered using shapes in ShapeNet. Both the network structure and the rendering setting are the same as in the evolutionary algorithm. For every τ RMSprop iterations (the number of iterations used to fine-tune a network in the evolution algorithm), we record the validation performance and save the snapshot of the network. When testing, the snapshot with the best validation performance is used.

- *ShapeNet-incremental* Same as the first *ShapeNet-incremental*, except that we restart the

RMSprop training every τ iterations, initializing from the latest weights. This is because in our evolution algorithm only the network weights are reloaded for incremental training, while the RMSprop training starts from scratch. We include this baseline to eliminate any advantage the restarts might bring in our evolution algorithm.

Ablated versions of our algorithm We consider three ablated versions of our algorithm:

- *Ours-no-feedback* The fitness score is replaced by a random value, while all other parts of the algorithm remain unchanged. The shapes are still being evolved, and the networks are still being trained, but there is no feedback on how good the shapes are.

- *Ours-no-evolution* The evolution is disabled, which means the population remains to be the initial set of primitive shapes throughout the whole process. This ablated version is equivalent to training a set of networks on a fixed dataset and picking the one from $n + m$ networks that has the best performance on the validation set every τ training iterations.

- *Ours-no-evolution-plus-ShapeNet* The evolution is disabled, and maintain a population of $n + m$ network instances being trained simultaneously. For each τ iterations, the network with the best validation performance is selected and copied to replace the entire population. It is equivalent to *Ours-no-evolution* except that the primitive shapes are replaced by shapes randomly sampled from ShapeNet each time we render an image. This ablation is to evaluate whether our evolved shapes are better than ShapeNet shapes, controlling for any advantage our training algorithm might have even without any evolution taking place.

2.5.2.2 Results and analysis

Table 2.1 compares the baselines with our approach. We first see that the deep network trained through shape evolution outperforms the state-of-the-art SIRFS algorithm, without using any external dataset except for the 10 training images in the MIT-Berkeley dataset that are also used by SIRFS.

Table 2.1: The results of baselines and our approach on the test images. *Measured by uniformly randomly outputting unit vectors on the $+z$ hemisphere.

	Summary Stats ↑			Errors ↓	
	$\leq 11.25°$	$\leq 22.5°$	$\leq 30°$	MAE	MSE
Random*	1.9%	7.5%	13.1%	1.1627	1.3071
SIRFS [8]	20.4%	53.3%	70.9%	0.4575	0.2964
ShapeNet-vanilla	12.7%	42.4%	62.8%	0.4831	0.2901
ShapeNet-incremental	15.2%	48.4%	66.4%	0.4597	0.2717
Ours-no-evolution-plus-ShapeNet	14.2%	53.0%	72.1%	0.4232	0.2233
Ours-no-evolution	17.3%	50.2%	66.1%	0.4673	0.2903
Ours-no-feedback	19.1%	49.5%	66.3%	0.4477	0.2624
Ours	**21.6%**	**55.5%**	**73.5%**	**0.4064**	**0.2204**

We also see that our algorithm outperforms all baselines trained on ShapeNet as well as all ablated versions. This shows that our approach can do away with an external shape dataset and generate useful shapes from simple primitives, and the evolved shapes are as useful as shapes from ShapeNet for this shape-from-shading task. Figure 2.8 shows example shapes at different stages of the evolution, as well as shapes from ShapeNet, and Figure 2.7 shows qualitative results of our method and SIRFS on the test data.

More specifically, the *Ours-no-evolution-plus-ShapeNet* ablation shows that our evolved shapes are actually more useful than ShapeNet for the task, although this is not surprising given that the evolution is biased toward being useful. Also it shows that the advantage our method has over using ShapeNet is due to evolution, not idiosyncrasies of our training procedure.

The *Ours-no-evolution* ablation shows that our good performance is not a result of well chosen primitive shapes, and evolution actually generates better shapes. The *Ours-no-feedback* ablation shows that the joint evolution and training is also important—random evolution can produce complex shapes, but without guidance from network training, the shapes are only slightly more useful than the primitives.

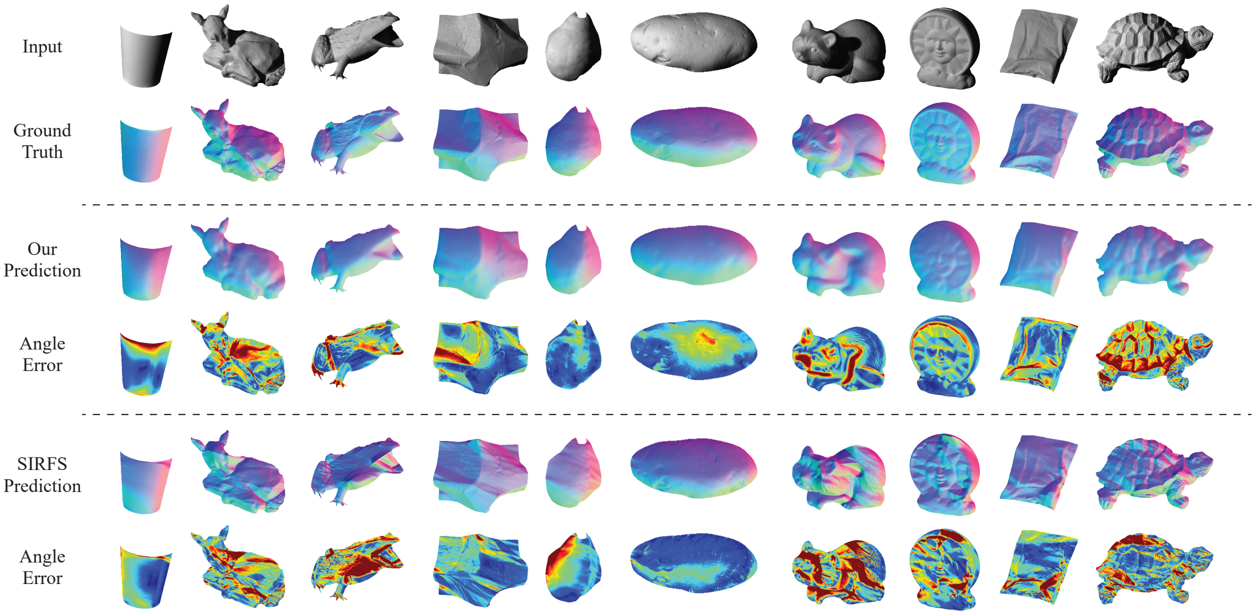


Figure 2.7: The qualitative results of our method and SIRFS [8] on the test data.

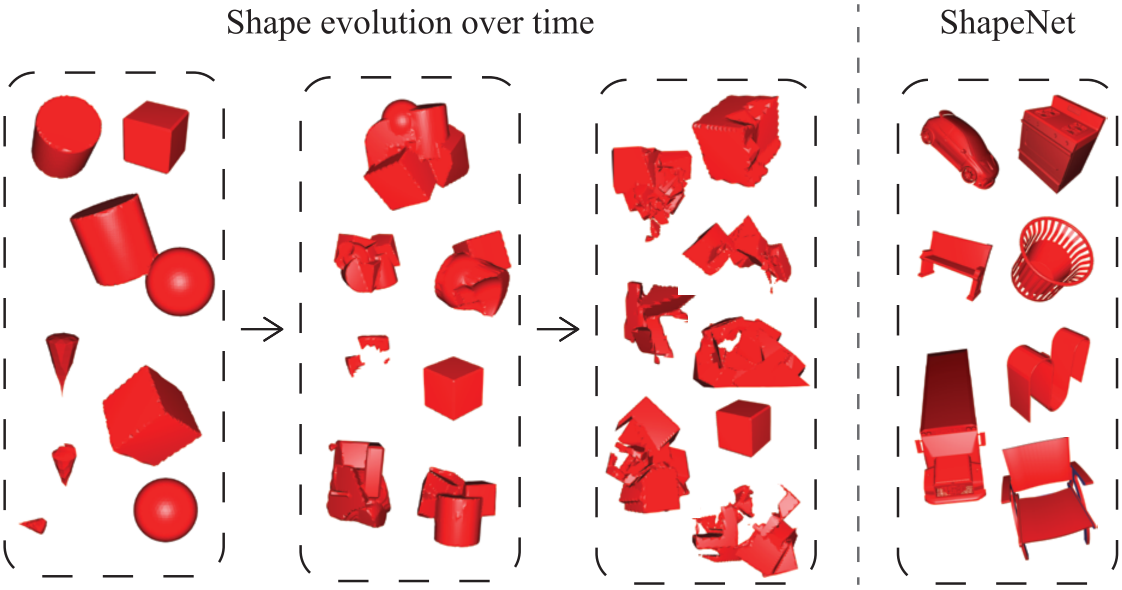


Figure 2.8: Example shapes at different stages of the evolution and shapes from ShapeNet.

CHAPTER 3

Learning to Generate 3D Synthetic Data through Hybrid Gradient[1]

3.1 Introduction

Synthetic images rendered by graphics engines have emerged as a promising source of training data for deep networks, especially for vision and robotics tasks that involve perceiving 3D structures from RGB pixels [15, 172, 155, 122, 95, 164, 18, 68, 140, 120, 119, 175, 77]. A major appeal of generating training images from computer graphics is that they have a virtually unlimited supply and come with high-quality 3D ground truth for free.

Despite its great promise, however, using synthetic training images from graphics poses its own challenges. One of them is ensuring that the synthetic training images are useful for real-world tasks, in the sense that they help train a network to perform well on real images. Ensuring this is challenging because a graphics-based generation pipeline requires numerous design decisions, including the selection of 3D shapes, the composition of scene layout, the application of texture, the configuration of lighting, and the placement of the camera. These design decisions can profoundly impact the usefulness of the generated training data, but have largely been made manually by researchers in prior work, potentially leading to suboptimal results.

In this paper, we address the problem of automatically optimizing a generation pipeline of synthetic 3D training data, with the explicit objective of improving the generalization performance of a trained deep network on real images.

One idea is black-box optimization: we try a particular configuration of the pipeline, use the pipeline to generate training images, train a deep network on these images, and evaluate the network

[1] This chapter is based on a join work with Jia Deng [169].

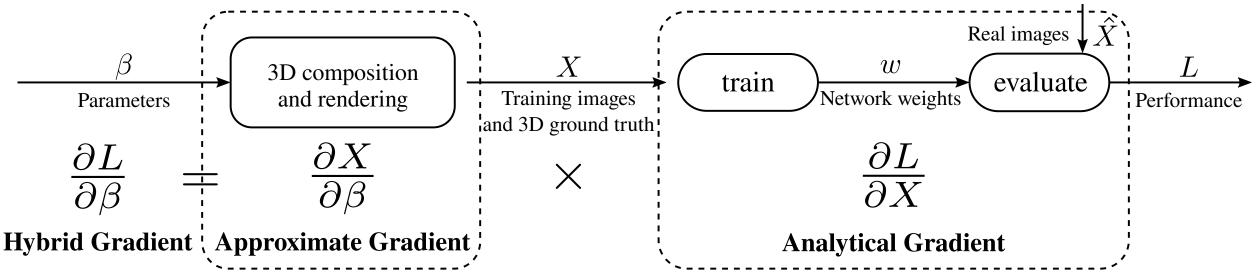

β
Parameters
3D composition and rendering
X
Training images and 3D ground truth
train
w
Network weights
evaluate
Real images
$\hat{X}$
L
Performance
$\frac{\partial L}{\partial \beta}$
$=$
$\frac{\partial X}{\partial \beta}$
$\times$
$\frac{\partial L}{\partial X}$
Hybrid Gradient
Approximate Gradient
Analytical Gradient

Figure 3.1: Our hybrid gradient method. We parametrize the design decisions as a real vector β and optimize the function of performance L with respect to β. From β to the generated training images and ground truth, we compute the approximate gradient by averaging finite difference approximations. From training samples X to L, we compute the analytical gradient through backpropagation with unrolled training steps.

on a validation set of real images. We can treat the performance of the trained network as a black-box function of the configuration of the generation pipeline, and apply black-box optimization techniques. Recent works [171, 123] have explored this exact direction. In Chapter 2, we use genetic algorithms to optimize the 3D shapes used in the generation pipeline. In particular, we start with a collection of simple primitive shapes such as cubes and spheres, and evolve them through mutation and combination into complex shapes, whose fitness is determined by the generalization performance of a trained network. We have shown that the 3D shapes evolved from scratch can provide more useful training data than manually created 3D CAD models. Meanwhile, Ruiz et al. [123] use black box reinforcement learning algorithms to optimize the parameters of a simulator, and shows that their approaches converge to the optimal solution in controlled experiments and can indeed discover good sets of parameters.

The advantage of black-box optimization is that it assumes nothing about the function being optimized as long as it can be evaluated. As a result, it can be applied to any existing function, including advanced photorealistic renderers. On the other hand, black-box optimization is computationally expensive—knowing nothing else about the function, it needs many trials to find a reasonable update to the current solution. In contrast, gradient-based optimization can be much more efficient by assuming the availability of the analytical gradient, which can be efficiently computed and directly correspond to good updates to the current solution, but the downside is that the analytical gradient is often unavailable, especially for many advanced photorealistic renderers.

In this work, we propose a new method that optimizes the generation of 3D training data based on what we call “hybrid gradient”. The basic idea is to make use of the analytical gradient where they are available, and combine them with black-box optimization for the rest of the function. We hypothesize that hybrid gradient will lead to more efficient optimization than black-box methods because it makes use of the partially available analytical gradient.

Concretely, if we parametrize the design decisions as a real vector β, the function mapping β to the network performance L can decompose into two parts: (1) from the design parameters β to the generated training images X, and (2) from the training images X to the network performance L. The first part often does not have analytical gradient, due to the use of advanced photorealistic renderers. We instead compute the approximate gradient by averaging finite difference approximations along random directions [90]. For the second part, we compute the analytical gradient through backpropagation—with SGD training unrolled, the performance of the network is a differentiable function of the training images. Then we combine the approximate gradient and the analytical gradient to obtain the hybrid gradient of the network performance L with respect to the parameters β, as illustrated in Figure 3.1.

A key ingredient of our approach is representing design decisions as real vectors of fixed dimensions, including the selection and composition of shapes. In Chapter 2, we have represented 3D shapes as a finite set of graphs, one for each shape. This representation is suitable for a genetic algorithm but is incompatible with our method in this chapter. Instead, we propose to represent 3D shapes as random samples generated by a Probabilistic Context-Free Grammar (PCFG) [54]. To sample a 3D shape, we start with an initial shape, and repeatedly sample a production rule in the grammar to modify it. The (conditional) probabilities of applying the production rules are parametrized as a real vector of a fixed dimension.

Our approach is novel in multiple aspects. First, to the best of our knowledge, we are the first to propose the idea of hybrid gradient, *i.e*. combining approximate gradient and analytical gradient, especially in the context of optimizing the generation of 3D training data. Second, we propose a novel integration of PCFG-based shape generation and our hybrid gradient approach.

We evaluate our approach on the task of estimating surface normal, depth and intrinsic components from a single image. Experiments on standard benchmarks and controlled settings show that our approach can outperform the prior state of the art on optimizing the generation of 3D training data, particularly in terms of computational efficiency.

3.2 Related Work

Generating 3D training data Synthetic images generated by computer graphics have been extensively used for training deep networks for numerous tasks, including single image 3D reconstruction [139, 58, 95, 62, 171, 17], optical flow estimation [94, 15, 46], human pose estimation [155, 23], action recognition [121], visual question answering [64], and many others [114, 91, 164, 151, 119, 120, 163]. The success of these works has demonstrated the effectiveness of synthetic images.

To ensure the relevance of the generated training data to real-world tasks, a large amount of manual effort has been necessary, particularly in acquiring 3D assets such as shapes and scenes [17, 62, 27, 166, 58, 95, 140]. To reduce manual labor, some heuristics have been proposed to generate 3D configurations automatically. For example, Zhang et al. [175] design an approach to use the entropy of object masks and color distribution of the rendered images to select sampled camera poses. McCormac et al. [95] simulate gravity for physically plausible object configurations inside a room.

Apart from simple heuristics, prior work has also performed automatic optimization of 3D configurations towards an explicit objective. For example, Yeh et al. [172] synthesize layouts with the target of satisfying constraints such as non-overlapping and occupation. Jiang et al. [63] learn a probabilistic grammar model for indoor scene generation, with parameters learned using maximum likelihood estimation on the existing 3D configurations in SUNCG [140]. Similarly, Veeravasarapu et al. [156] tune the parameters for stochastic scene generation using generative adversarial networks, targeting at making synthetic images indistinguishable from real images. Qi et al. [112] synthesize 3D room layouts based on human-centric relations among furniture, to achieve visual realism, functionality and naturalness of the scenes. However, these optimization objectives are different from ours, which is the generalization performance of a trained network on real images.

In terms of generating 3D training data, the closest prior works to ours are those of [171, 65, 123]. Specifically, in Chapter 2, we use a genetic algorithm to optimize the 3D shapes used for rendering synthetic training images. In this chapter our optimization objective is the same, except that the optimization method is different: we leverage gradient information as opposed to using evolution-based approach. Similarly, Meta-Sim [65] also tries to optimize 3D parameters with REINFORCE towards better task generalization performance, and Ruiz et al. [123] learn a policy for simulator parameters also using REINFORCE. However, they do not backpropagate analytical gradient from the meta-objective, so their algorithms can be considered as black-box estimation by multiple trials, with an improved efficient sampling strategy (REINFORCE). In our experiments, we compared to an algorithm that has been shown competitive to REINFORCE in training deep policy networks [90, 127, 141].

Unrolling and backpropagating through network training One component of our approach is unrolling and backpropagating through the training iterations of a deep network. This is a technique that has often been used by existing work in other contexts, including hyperparameter optimization [89], meta-learning [1, 52, 99, 75, 41] and others [178, 25]. Our work is different in that we apply this technique in a novel context: it is used to optimize the generation of 3D training data, and the gradient with respect to the input images is integrated with approximate gradient to form hybrid gradient.

Hyperparameter optimization Our method is connected to hyperparameter optimization in the sense that we can treat the design decisions of the 3D generation pipeline as hyperparameters of the training procedure.

Hyperparameter optimization of deep networks is typically approached as black-box optimization [12, 11, 69, 14]. While Klatzer and Pock [67] propose a bi-level gradient-based approach for continuous hyperparameter optimization of Support Vector Machines, but it has not been applied to deep networks and 3D generation. Since black-box optimization does not make assumption of the function being optimized, it requires repeated evaluation of the function, which is expensive in this case because it contains the process of training and evaluating a deep network. In contrast, we combine the analytical gradient from backpropagation and the approximate gradient from generalized finite difference for more efficient optimization.

Domain adaptation Researchers have also applied domain adaptation techniques to transfer the knowledge learned from synthetic data to real data. Like domain adaptation, our method involves data from two domains: synthetic and real. However, our setting is different: in domain adaptation, the distribution of training data is fixed; in our setting, we are concerned about *generating and changing* the distribution of training data in the source domain.

Differentiable rendering Researchers have also explored differentiable rendering engines to obtain the gradient with respect to the input 3D content such as mesh vertices, lighting intensity *etc*. [84, 66, 162, 76, 19]. Generally, they obtain the gradient through backpropagation [66, 84] or sampling [162, 76, 19]. The differentiable renderers often assume simple surface reflectance and illumination model, and they are typically developed for a specific 3D input format (such as triangle meshes and directional lighting) or a specific rendering algorithm (such as path tracing). In fact, we are not aware of any photorealistic differentiable renderer that is differentiable over a shape parametrization that allows not only continuous deformation but also topology change. In our method, we assume nothing about the rendering engine and obtain the gradient with respect to the decision vector by approximation, bypassing the surface and illumination model or any rendering algorithms. So our method is flexible and not limited by choices of graphics engines of any kind.

3.3 Problem Setup

Suppose we have a probabilistic generative pipeline. We use a deterministic function, $f(\beta, r)$ to represent the sampling operation. This function f takes the real vector β and the random seed r as input. An image and its 3D ground truth are computed by evaluating the function $f(\beta, r)$. By choosing n different random seeds r, we obtain a dataset of size n for training:

$$X = (f(\beta, r^{(1)}), f(\beta, r^{(2)}), \cdots, f(\beta, r^{(n)})) \tag{3.1}$$

Then, a deep neural network with initialized weights w_0 is trained on the training data X, with the function $\mathrm{train}(w_0, X)$ representing the optimization process and generating the weights of the trained network.

The network is then evaluated on real data $\hat{X}$ with a validation loss l_{eval} to obtain a generalization performance L:

$$L = l_{\text{eval}}(\mathrm{train}(w_0, X), \hat{X}) \tag{3.2}$$

Combining the above two functions, L is a function of β, and the task is to optimize this value L with respect to the parameters β.

As we mentioned in the previous section, black-box algorithms typically need repetitive evaluations of this function, which is expensive.

3.4 Approach

3.4.1 Generative Modeling of Synthetic Training Data

We decompose the function $f(\beta, r)$ into two parts: 3D composition and rendering.

3D composition Context-free grammars have been used in scene generation [63, 112] and in the parsing of the Constructive Solid Geometry (CSG) shapes [136] because they can represent shapes and scenes in a flexible and composable manner. Here, we design a probabilistic context-free grammar (PCFG) [54] to control the random generation of unlimited shapes [43].

In a PCFG, a tree is randomly sampled given a set of probabilities. Starting from a root node, the nodes are expanded by randomly sampling probabilistic rules repeatedly until all the leaf nodes cannot expand. Since multiple rules may apply, the parameters in a PCFG define the probability distribution of applying different rules.

In our PCFG, a shape is constructed by composing two other shapes through union and difference; this construction is recursively applied until all leaf nodes are a predefined set of concrete primitive shapes (terminals). The parameters include the parameters of primitive shapes as well as the probability of either expanding the node or replacing it with a terminal.

Given our PCFG model with the probability parameters β_S, a 3D shape S can be composed by computing a deterministic function f_S given β_S and a random string r_S as the input:

$$S = f_{\mathcal{S}}(\beta_S, r_S) \tag{3.3}$$

Rendering training images we use a graphics renderer R to render the composed shape S. The rendering configurations P (*e.g.* camera poses), are also sampled from a distribution controlled by a set of parameters β_R (with a random string r_R):

$$P = f_{\mathcal{R}}(\beta_R, r_R) \tag{3.4}$$

Now that we have Equation 3.3 and 3.4, The full function for training data generation can be represented as follows:

$$f(\beta, r) = R(S, P) = R(f_{\mathcal{S}}(\beta_S, r_S), f_{\mathcal{R}}(\beta_R, r_R)) \tag{3.5}$$

where $\beta = (\beta_R, \beta_S)$ and $r = (r_R, r_S)$.

By sampling different random strings r, we obtain a set of training images and their 3D ground truth X.

3.4.2 Hybrid Gradient

After training deep network on synthetic training data X, the network is evaluated on a set of validation images $\hat{X}$ to obtain the generalization loss L.

Recall that to compute the hybrid gradient $\frac{\partial L}{\partial \beta}$ to optimize β, we multiply two types of gradient:

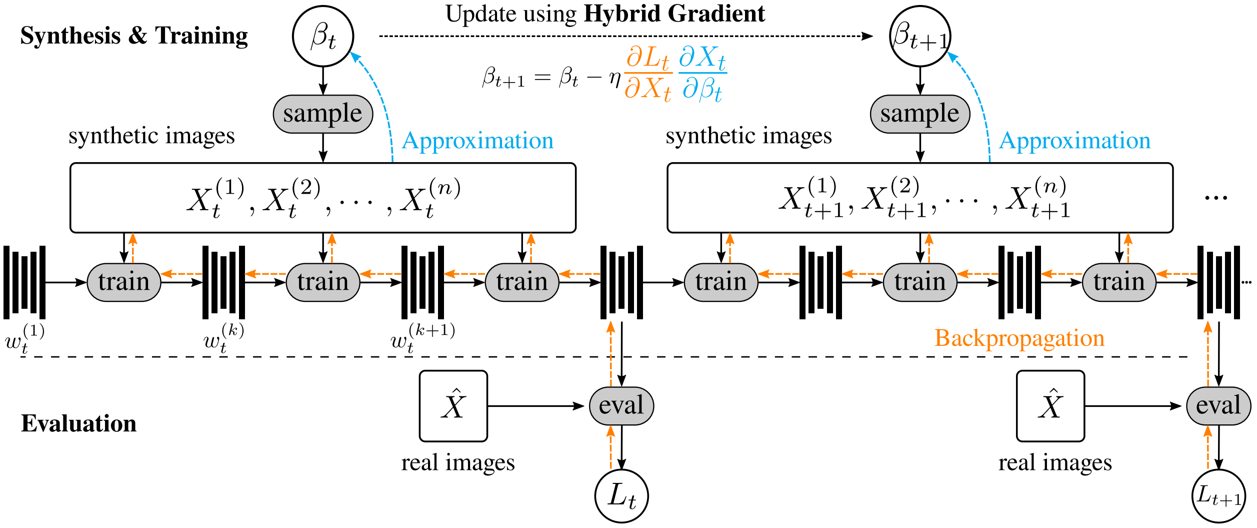

Synthesis & Training
β_t
Update using **Hybrid Gradient**
β_{t+1}
$\beta_{t+1} = \beta_t - \eta \frac{\partial L_t}{\partial X_t} \frac{\partial X_t}{\partial \beta_t}$
sample
synthetic images
Approximation
$X_t^{(1)}, X_t^{(2)}, \cdots, X_t^{(n)}$
$X_{t+1}^{(1)}, X_{t+1}^{(2)}, \cdots, X_{t+1}^{(n)}$
...
train
$w_t^{(1)}$
$w_t^{(k)}$
$w_t^{(k+1)}$
Backpropagation
Evaluation
$\hat{X}$
real images
eval
L_t
L_{t+1}

Figure 3.2: The details of using "hybrid gradient" to incrementally update β and train the network. The analytical gradient is computed by backpropagating through unrolled training steps (colored in orange). The numerical gradient is computed using finite difference approximation by sampling in a neighborhood of β_t (colored in cyan). Then β_t is updated using hybrid gradient, and the trained network weights are retained for the next timestamp $t + 1$.

the gradient of network training $\frac{\partial L_t}{\partial X}$ and the gradient of image generation $\frac{\partial X}{\partial \beta}$, as is shown in Figure 3.2.

Analytical gradient from backpropagation We assume the network is trained on a set of previously generated training images $X^{(1)}, X^{(2)}, \cdots, X^{(n)}$. Without loss of generality, we assume mini-batch stochastic gradient descent (SGD) with a batch size of 1 is used for weight update. Let function g denote the SGD step and let l_{train} denote the training loss:

$$w^{(k+1)} = w^{(k)} - \eta \frac{\partial l_{\text{train}}(w^{(k)}, X^{(k)})}{\partial w^{(k)}} = g(w^{(k)}, X^{(k)}; l_{\text{train}}, \eta) \tag{3.6}$$

Note that the SGD step g is differentiable with respect to the network weights $w^{(k)}$ as well as the training batch $X^{(k)}$, if our training loss l_{train} is twice (sub-)differentiable. This requirement is satisfied in most practical cases. To simplify the equation, we assume the training loss l_{train} and the learning rate η do not change during one update step of β, so the variables can be safely discarded in the equation.

Therefore, the gradient from the generalization loss L to each sample $X^{(k)}$ can be computed through backpropagation. Given Equation 3.6:

$$\begin{aligned} \frac{\partial L}{\partial X^{(k)}} &= \frac{\partial L}{\partial w^{(k+1)}} \cdot \frac{\partial w^{(k+1)}}{\partial X^{(k)}} = \frac{\partial L}{\partial w^{(k+1)}} \cdot g_2'(w^{(k)}, X^{(k)}) \\ \frac{\partial L}{\partial w^{(k)}} &= \frac{\partial L}{\partial w^{(k+1)}} \cdot \frac{\partial w^{(k+1)}}{\partial w^{(k)}} = \frac{\partial L}{\partial w^{(k+1)}} \cdot g_1'(w^{(k)}, X^{(k)}) \end{aligned} \tag{3.7}$$

with the initial value $\frac{\partial L}{\partial w^{(n+1)}}$ computed from the validation loss l_{eval}:

$$\frac{\partial L}{\partial w^{(n+1)}} = l'_{\text{eval}}(w^{(k+1)}, \hat{X}) \tag{3.8}$$

Approximate gradient from finite difference For the formulation in Equation 3.5, the graphics renderer can be a general black box and non-differentiable. We can approximate the gradient of each rendered image with ground truth $X^{(1)}, X^{(2)}, \cdots$ with respect to the generation parameters β using generalized finite difference. We adopt the form of [90] because this gradient approximation algorithm in Random Search has been shown effective for training deep policy networks [90, 127, 141]. Concretely, we sample a set of noise from an uncorrelated multivariate Gaussian distribution:

$$\delta_1, \delta_2, \cdots, \delta_m \sim \mathcal{N}(\mathbf{0}, \sigma I) \tag{3.9}$$

Next, we approximate the Jacobian for each sample ($\otimes$ denotes outer product):

$$\frac{\partial X^{(i)}}{\partial \beta} \approx \frac{1}{m} \sum_{j=1}^{m} \frac{f_{\mathcal{D}}(\beta + \delta_j, r_i) - f_{\mathcal{D}}(\beta - \delta_j, r_i)}{2\|\delta_j\|} \otimes \frac{\delta_j}{\|\delta_j\|} \tag{3.10}$$

Incremental training Similar to what is done in Section 2.4, we incrementally train the network w along with the update of β, instead of initializing $w^{(1)}$ from scratch each time. At timestamp t, we update β_t with the hybrid gradient; for network weights, we keep the trained network in timestamp t for initialization in the next timestamp $t + 1$:

$$\beta_{t+1} = \beta_t - \gamma \frac{\partial L_t}{\partial \beta_t} = \beta_t - \gamma \sum_{i=1}^{n} \frac{\partial L_t}{\partial X_t^{(i)}} \cdot \frac{\partial X_t^{(i)}}{\partial \beta_t} \qquad w_{t+1}^{(1)} = w_t^{(n+1)} \tag{3.11}$$

3.5 Experiments

Datasets We evaluate our algorithm on four different datasets, and three standard prediction tasks for single-image 3D. The input is an RGB image and the output is pixel-wise surface normal, depth, or albedo shading map.

Specifically, we experiment on the task of surface normal estimation on two real datasets: MIT-Berkeley Intrinsic Images Dataset (MBII) [8], which focuses on images of single objects and NYU Depth [138], which focuses on indoor scenes. For the other two datasets, we illustrate that our method can easily extend to other 3D setups. We experiment on the task of depth estimation on the renderings of the scanned human faces in the Basel Face Model dataset [107], and on the task of intrinsic image decomposition and evaluate on the renderings of ShapeNet [17] shapes.

Baselines For comparison, we implemented a black-box optimization method. Random search [3] has been extensively explored [42, 101, 90] as a derivative-free optimization method, and Mania et al. [90] have shown that their simple version, Basic Random Search, has comparable performance compared to typical reinforcement learning algorithms. Therefore, we re-implemented their Basic Random Search such that this baseline has the same setting as in our method, while the only difference is that the gradient from the validation loss is obtained through sampling instead of hybrid gradient. We also compare against baselines with a random β baseline in the following experiments. In these baselines, the networks are trained on a dataset generated using multiple random but fixed β, and the weight snapshots with the best validation performance are used to evaluate on the test set.

These two baselines, along with our hybrid gradient method, all use information from the validation set but in a different way: hybrid gradient backpropagates the gradient of the validation performance to update β; random search samples β to get the gradient from the validation performance; the random β baseline fixes the dataset and uses the validation performance to select the best network snapshot.

In all of our experiments, the network weights are updated using only synthetic images in the training iterations, and the generalization loss is computed only on the validation split of the datasets

Table 3.1: Ablation Study: the diagnostic experiment to compare with random but fixed β. We sample 10 values of β in advance, and then train the networks with the same setting as in hybrid gradient. The best, median and worst performance is reported on the test images, and the corresponding values of β are used to initialize β_0 for hybrid gradient for comparison. The results show that our approach is consistently better than the baselines with fixed β.

		Summary Stats ↑			Errors ↓		
		$\leq 11.25°$	$\leq 22.5°$	$\leq 30°$	MAE	Median	MSE
Fixed β	$\beta = \beta_{\text{best}}$	19.9%	52.7%	70.5%	$24.0°$	$21.5°$	0.2282
	$\beta = \beta_{\text{median}}$	20.7%	50.9%	67.5%	$24.8°$	$22.1°$	0.2461
	$\beta = \beta_{\text{worst}}$	17.9%	46.7%	64.6%	$25.6°$	$23.8°$	0.2553
Hybrid gradient	$\beta_0 = \beta_{\text{best}}$	22.7%	58.5%	73.9%	$22.5°$	$19.3°$	0.2065
	$\beta_0 = \beta_{\text{median}}$	24.0%	60.1%	75.7%	$21.8°$	$18.8°$	0.1938
	$\beta_0 = \beta_{\text{worst}}$	26.0%	58.6%	73.9%	$22.0°$	$19.1°$	0.1998

mentioned above. The decision vector β is updated using RMSprop [150] for hybrid gradient.

For MBII, we use pure synthetic shapes as in Chapter 2 to render training images. We first compare our method with ablation baselines, then show that our algorithm is better than the previous state of the art on MBII. For NYU Depth, we base our generative model on SUNCG [140] and augment the original 3D configurations in Zhang et al. [175]. For Basel Face Model, we sample synthetic faces from a morphable model and evaluate on the renderings of scanned faces. For the intrinsic image decomposition task, we sample textures from a simple procedural pipeline and attach the synthetic textures to SUNCG shapes [140], and evaluate on renderings of ShapeNet shapes [17].

Table 3.2: Our approach compared to previous work, on the test set of MIT-Berkeley images [8]. The results show that our approach is better than the state of the art as reported in Table 2.1.

	Summary Stats ↑			Errors ↓		
	$\leq 11.25°$	$\leq 22.5°$	$\leq 30°$	MAE	Median	MSE
SIRFS [8]	20.4%	53.3%	70.9%	$26.2°$	—	0.2964
Evolution [171] (Implicit Function)	21.6%	55.5%	73.5%	$23.3°$	—	0.2204
Evolution [171] (Mesh Implementation)	23.0%	58.3%	73.8%	$22.5°$	$\mathbf{18.8°}$	0.2042
Basic Random Search [90]	21.9%	$\mathbf{59.6\%}$	74.0%	$22.8°$	$19.2°$	0.2106
Hybrid gradient	$\mathbf{24.5\%}$	59.3%	$\mathbf{74.3\%}$	$\mathbf{22.0°}$	$18.9°$	$\mathbf{0.1984}$

3.5.1 Normal Estimation on MIT-Berkeley Intrinsic Images

Following Chapter 2, we recover the surface normals of an object from a single image.

Synthetic shape generation In Section 2.3.1, a population of primitive shapes such as cylinders, spheres and cubes are evolved and rendered to train deep networks. The evolution operators include transformations of individual shapes and the boolean operations of shapes in Constructive Solid Geometry (CSG) [43]. In our algorithm, we also use the CSG grammar for our PCFG:

```
S => E
E => C(E, T(E)) | P
C => union | subtract
P => sphere | cube | truncated_cone | tetrahedron
T => attach * rand_transl * rand_rotate * rand_scale
```

In this PCFG, the final shape `S` is generated by recursively composing (`C`) other shapes `E` with transformations `T`, until primitives `P` are sampled at all `E` nodes. The parameter vector β consists of three parts: (1) The probability of the different rules; (2) The means and variations of log-normal distributions controlling shape primitives (`P`), such as the radius of the sphere; (3) The means and variations of log-normal distributions controlling transformation parameters (`T`), such as scale values. Examples of sampled shapes are shown in Figure 3.3. For the generalization loss L, we compute the mean angle error of predictions on the training set of the MIT-Berkeley dataset.

Training setup For network training and evaluation, same as in Section 2.5.2 we train the Stacked Hourglass Network [102] on the images, and use the standard split of the MBII dataset for the optimization of β and testing.

We report the performance of surface normal directions with the metrics commonly used in previous works, including mean angle error (MAE), median angle error, mean squared error (MSE), and the proportion of pixels that normals fall in an error range ($\leq N^\circ$). See Appendix A for detailed definitions.

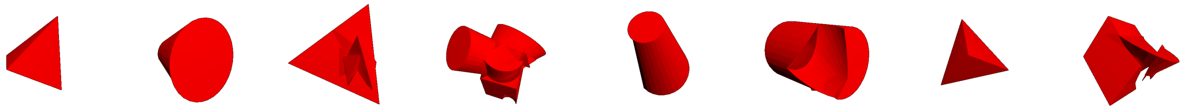


Figure 3.3: Sampled shapes from our probabilistic context-free grammar, with parameters optimized using hybrid gradient.

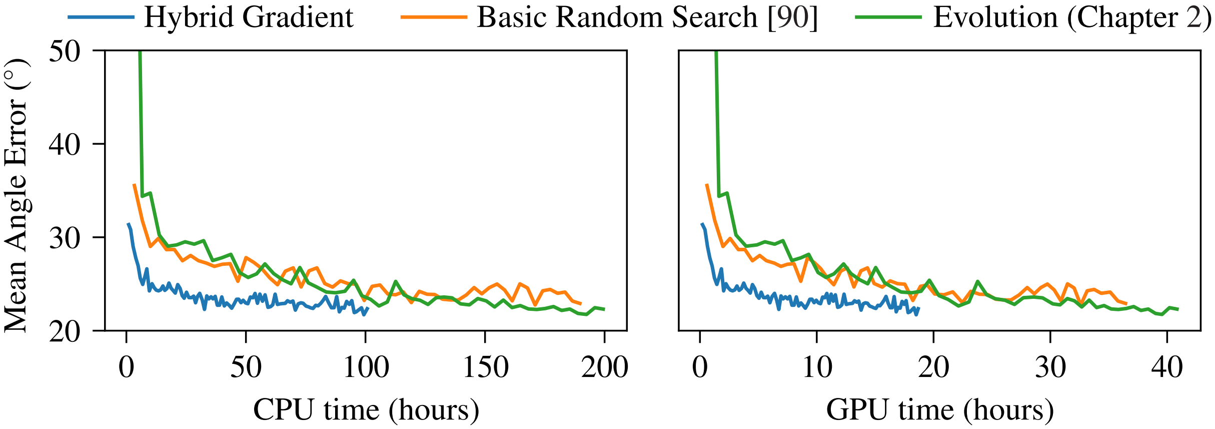


Figure 3.4: Mean angle error on the test images *vs*. computation time, compared to two black-box optimization baselines.

Ablation study We first sample 10 random values of β and fix those values in advance. Then, for each β, we sample 3D shapes and render images to train a network, with the same training and evaluation configurations as in our hybrid gradient, except that we do not update β. We then report the best, median and worst performance of those 10 networks, and label the corresponding β as β_{best}, β_{median} and β_{worst}. In hybrid gradient, we then initialize β_0 from these three values, run our algorithm, and report the performance on test images also in Table 3.1.

From the table we can observe that training with a fixed β can hardly match the performance of our method, even with multiple trials. Instead, our hybrid gradient approach can optimize β to a reasonable performance regardless of different initialization ($\beta_{\text{best/median/worst}}$). This simple diagnostic experiment demonstrates that our algorithm is working correctly: the optimization of β is necessary in order to generate useful synthetic images for training networks.

Comparison with the state of the art In addition to Basic Random Search as mentioned earlier, in this experiment we also compare with Shape Evolution (Chapter 2), which is a state-of-the-art method on MIT-Berkeley Intrinsic Images.

In Shape Evolution, a population of shapes are evolved, and fitness scores for individual shapes are computed using a network trained on an incremental dataset and evaluated on the validation set. We compose our shapes in mesh representations, slightly different from the implicit functions in 2.3.1. Therefore, we re-implemented Shape Evolution with mesh representations for a fair comparison. We initialization β from the hyper-parameters in Shape Evolution, and train the networks and update β for the same number of steps. We then report the test performance of the network that has the best validation performance. The results are shown in Table 3.2.

We also run the experiments on the same set of CPUs and GPUs, sum the computation time, and plot the mean angle error (on the test set) with respect to the CPU time and GPU time Figure 3.4). We see that our algorithm is more efficient than the above baselines. This is natural, because when computing $\partial L/\partial\beta^{(t)}$ in black-box algorithms, for each sample of $\beta^{(t)} + \delta_j$, one needs to train one network to evaluate the performance L, while in hybrid gradient, only a forward training pass and a backpropagation pass for a single network are required to compute $\partial L/\partial X$. Shapes sampled from our optimized PCFG are shown in Figure 3.3.

3.5.2 Normal Estimation on NYU Depth

Scene perturbation We design our scene generation grammar as an augmentation of collected SUNCG scenes [139] with the cameras from Zhang et al. [175]:

```
S => E,P
E => T_shapes * R_shapes * E0
P => T_camera * R_camera * P0
T_shapes => translate(x, y, z)
R_shapes => rotate(yaw, pitch, roll)
```

For each 3D scene `S`, we perturb the positions and poses of the original cameras (`P0`) and shapes (`E0`) using random translations and rotations. The position perturbations follow a mixture of uncorrelated Gaussians, and the perturbations for pose angles (yaw, pitch & roll) follow a mixture of von Mises, *i.e.* wrapped Gaussians. The vector β consists of the parameters of the above distributions.

Training setup Our networks are trained on synthetic images only, and evaluated on NYU Depth V2 [138] with the same setup as in Zhang et al. [175]. For real images in our optimization pipeline, we sample a subset of images from the standard validation images in NYU Depth V2. We initialize

Table 3.3: The performance of the finetuned networks on the test set of NYU Depth V2 [138], compared to the original network in [175]. The networks are trained only on the synthetic images. Without optimizing the parameters (random β), the augmentation hurts the generalization performance. With proper search of β using hybrid gradient, we are able to achieve better performance than the original model.

	Summary Stats ↑			Errors ↓	
	$\leq 11.25°$	$\leq 22.5°$	$\leq 30°$	Mean	Median
Original [175]	24.1%	49.7%	61.5%	$28.8°$	$22.7°$
Random β + [175]	23.0%	48.8%	61.3%	$29.2°$	$23.2°$
Hybrid gradient + [175]	$\mathbf{27.3}\%$	$\mathbf{52.5}\%$	$\mathbf{63.8}\%$	$\mathbf{28.1}°$	$\mathbf{21.1}°$

our network from the synthetically trained model in Zhang et al. [175] and initialize β_0 using a small value. To compare with random β, we construct a dataset of 40k images with a small random β for each image. We then load the same pre-trained network and train for the same number of iterations as in hybrid gradient. We then evaluate the networks on the test set of NYU Depth V2 [138], following the same protocol. The results are reported in Table 3.3. Note that none of these networks has been trained on real images except for validation, and the validation subset of real images is only used to update the decision vector.

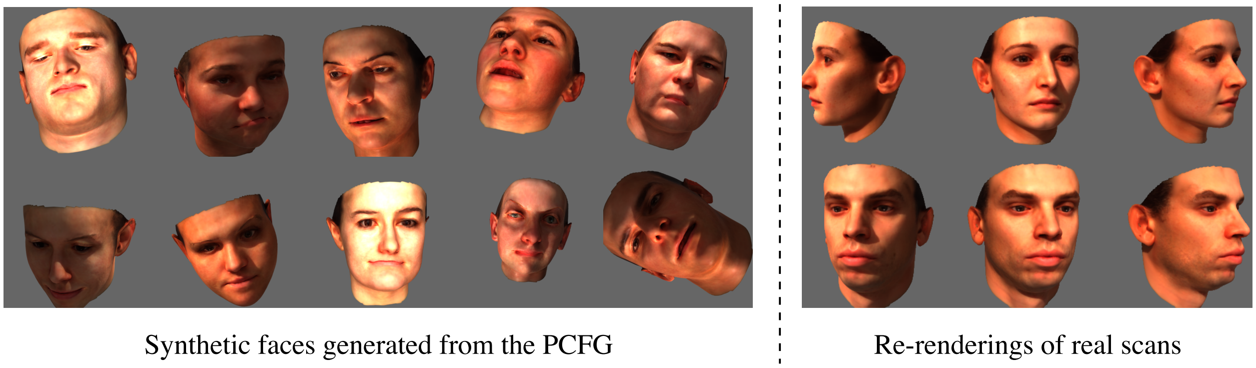
The numbers indicate that our parametrized generation of SUNCG augmentation exceeds the original baseline performance. Note that the network trained with random β is worse than original performance. This means without proper optimization of perturbation parameters, such random augmentation may hurt generalization, demonstrating that good choices of these parameters are crucial for generalization to real images.

3.5.3 Depth Estimation on Basel Face Model

Synthetic face generation We exploit an off-the-shelf 3DMM morphable face and expression model [34, 181, 180] to generating human 3D models, with face and pose parameters randomly sampled from mixtures of Gaussians or von Mises. Since the parameters for 3DMM are PCA coefficients, we only include the first 10 principal dimensions each for geometry, texture and expression parameters in the decision vector β, and uniformly sample for the remaining dimensions to save disk usage.

Training setup We train a stacked hourglass network [102] from scratch with a single-channel output after a ReLU layer to predict the raw depth, and supervise using mean squared error. The learning rate for the network is 0.1 and the batch size is 8.

Evaluation We evaluate on the renderings of the scanned human faces [107]. We split the 10 identities into two disjoint sets for validation and test, then use the rendering parameters provided in the dataset to recreate the renderings as well as depth images. For each scan, there are 3 lighting directions and 9 pose angles, creating 135 validation images and 135 test images. Example images are shown in Figure 3.5. For depth evaluation, we use the standard metrics including the relative difference (absolute and squared) and root mean squared error (linear, log and scale-invariant log). The definitions are listed in Eigen et al. [39] and also detailed in Appendix A.


Synthetic faces generated from the PCFG

Re-renderings of real scans

Figure 3.5: Training images generated using PCFG with 3DMM face model, and example test images.

Table 3.4: The results on the scanned faces of the Basel Face Model. Our method is able to search for the synthetic face parameters such that the trained network can generalize better.

	Relative Difference		RMSE		
	abs	sqr	linear	log	scale inv.
Random β	0.03718	9.701×10^{-3}	0.1395	0.1014	0.09717
Basic Random Search [90]	0.02330	1.728×10^{-3}	0.0581	0.0299	0.02700
Hybrid gradient	$\mathbf{0.02256}$	$\mathbf{1.649 \times 10^{-3}}$	$\mathbf{0.0570}$	$\mathbf{0.0293}$	$\mathbf{0.02603}$

The results in 3.4 show that our algorithm is able to search for better β so that the network trained on the synthetic faces and generalize better on the scanned faces.

3.5.4 Intrinsic Image Decomposition on ShapeNet

Texture generation and rendering We design a painter's algorithm as PCFG for generating the textures. To generate one texture image, we paint Perlin-noise-perturbed polygons sequentially onto a canvas, and then repeat the canvas as the final texture image. The number of repetitions and the number of polygons follow zero-truncated Poisson distributions, the vertex coordinates follow independent truncated Gaussian mixtures, and the number of edges in a polygon are also controlled by sampling probabilities. All the distribution parameters are concatenated to form the decision vector β. Example textures are shown in Figure 3.6.

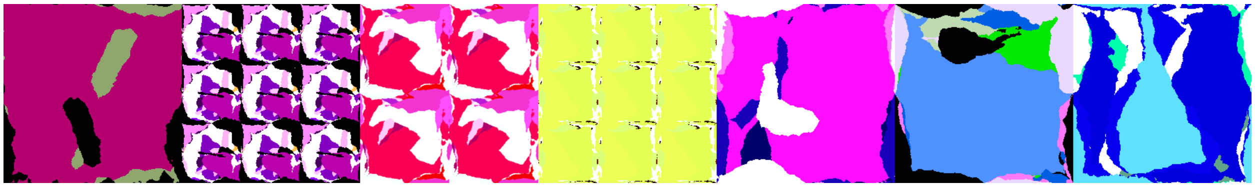


Figure 3.6: Example textures generated using our procedural pipeline with parameters controlled by β.

The texture is then mapped onto the SUNCG shapes [140]. We choose SUNCG shapes because they are well parametrized for texture mapping and we can easily apply our synthetic textures. We then render the textured shapes using random directional lights as training data. For validation and testing, we randomly render ShapeNet [17] shapes with their original textures, and randomly choose 50 as validation and 50 for test. The shapes used in validation or test are mutually exclusive.

Training We use the Stacked Hourglass Network [102] with a 4-channel output (3 for albedo, 1 for shading), and train with a learning rate of 10^{-4} and a batch size of 8. For supervision, we sum the mean squared error for both albedo and shading outputs as the loss.

Evaluation We also compare with our Basic Random Search implementation and with the random β baseline. We evaluate the performance using mean absolute error (abs), root mean squared error (rmse) and scale-invariant rmse for albedo and shading. We also evaluate the reconstruction error of the rendered image, even though we do not have any supervision for the reconstruction error of the image. The results are shown in Table 3.5.

Table 3.5: The results of intrinsic image decomposition on the ShapeNet renderings.

		abs	rmse	rmse (scale inv.)
Random β	Albedo	0.157	0.198	0.175
	Shading	0.118	0.132	0.095
	Reconstruction	0.139	0.169	–
Basic Random Search [90]	Albedo	0.152	0.193	0.177
	Shading	**0.104**	**0.116**	**0.085**
	Reconstruction	0.134	0.166	–
Hybrid gradient	Albedo	**0.147**	**0.189**	**0.168**
	Shading	**0.104**	0.119	0.088
	Reconstruction	**0.118**	**0.150**	–

CHAPTER 4

Learning to Generate 3D Synthetic Data with a Novelty Metric[1]

4.1 Introduction

For pixelwise 3D perception in vision and robotics, rendering synthetic images using a graphics engine offers advantages over manually collecting and annotating real images. Synthetic rendering has low computation cost, readily available groundtruth labels and the flexibility to adjust scene configurations such as 3D objects, lighting, cameras and layouts. These advantages are especially valuable when training deep neural networks for vision tasks as demonstrated by the growing body of work exploiting synthetic data [62, 15, 144, 122, 120, 46, 93, 18, 155, 121, 95, 119, 111, 152, 77, 133].

When training with synthetic data, the variety and diversity of 3D assets are crucial. There is often a substantial domain gap between the synthetic training setting and final the downstream task. Thus it is important to expose the model to as much variety in the synthetic setting for better transfer to the target task [151, 94]. To increase synthetic data diversity, several key strategies are typically encountered in prior work:

- **Manual collection of 3D assets.** Manual collection of 3D assets plays a important role for prior synthetic datasets such as scenes [140, 95, 58] and shapes [17, 166, 62, 27, 179]. One advantage is that 3D assets collected by humans are of high quality and correspond better to real-world distributions. However, a drawback is the significant time, labor, and expertise required to collect data and build new assets.

[1] This chapter is based on a joint work with Jia Deng [170].

- **Empirical priors for randomizing 3D synthetic data.** Researchers automate some of the design choices that go into rendering synthetic data, such as automatically varying object poses, lighting angles, texture colors and camera placement. To improve dataset quality, they encode empirical priors to ensure the dataset is useful as a training set. For example, cameras can be automatically generated and filtered by how much information is in the captured image [175]; objects can be randomly arranged through gravity simulation [95, 152]. The goal of these heuristics is to improve synthetic data quality such that the model performs better when finetuned on a task with real-world data. As such, these heuristics incorporate our knowledge of real-world priors, which have been shown to be helpful.

- **Automatic optimization towards external objectives.** An external real dataset is a good supervision source for learning 3D data generation. For example, one can learn the 3D generator by matching the synthetic distribution to a real distribution [65, 112, 156, 63, 86]. To study the effect of network training and generalization during optimization of a synthetic generator, researchers have also involved network training in the optimization loop as a meta-learning or bi-level optimization scheme [171, 169, 65, 123, 20]. In these examples, an external real dataset provides a coarse feedback signal to update the generator parameters. In these prior works, optimization with external distributions has achieved exceptional performance when evaluated on the same distribution.

However, in these prior works, the diversity and novelty of the data may be implicitly achieved, but are not explicitly explored. In this chapter, we propose an alternative to the above methods, targeting automatic synthetic dataset creation using diversity as the optimization objective. We draw inspiration from novelty search [73]: instead of focusing on extrinsic motivation such as task performance or distribution divergence, we only consider intrinsic motivation of a synthetic data generator—novelty. It means a synthetic generator should always generate a new data sample that looks different from previous ones. Once a novel sample is seen, it is no longer viewed as novel. This novelty motivation constantly pressures the generator to be creative. As a result, compared to a generator without such motivation, it is able to explore a broader range of synthetic samples, thus leading to more variety in the training data.

A natural question is how to evaluate the novelty of a training sample. In typical novelty search literature, novelty is well defined as sparsity using Euclidean distances [98, 72]. However, it is not trivial to define the novelty for image samples that will be used for training deep networks. In this work, we propose a novelty metric using deep autoregressive models [153, 154, 128]. Deep autoregressive models are generative models of 2D images that can evaluate the probability density of an image in a distribution. Our novelty metric is then defined using the probability density of a new synthetic image in a distribution of all previously seen synthetic images.

Data is generated through an evolutionary framework which incorporates this novelty metric. Initially, we have a population of 3D configurations such as shapes, poses, lighting angles and camera parameters. One configuration (genome) corresponds to a set of training samples (phenome). An autoregressive model is then trained on all generated images and evaluate the probabilities for each genome in the current population. The probabilities are then converted to novelty scores to evolve the population. After each evolution epoch, the autoregressive model is finetuned on all seen images, including previously generated ones. The population learns to shift from configurations that generate trivial images to the ones that can produce different, novel images. After evolution, we collect all the training samples generated along the process to train a downstream network for 3D tasks such as normal estimation or intrinsic image decomposition. The framework is shown in Figure 4.1.

Our novelty-guided evolution framework has unique strengths. First, it is task-agnostic. Our method does not rely on any downstream task specification such as predicting depth, normal or intrinsic image decomposition. One only needs to design the components in the evolution domain, such as genomes, genetic operators and evolutionary operators. Once the evolution process is complete, we only need to render the different 3D ground truth for different downstream tasks. Second, it employs no external datasets, compared to previous optimization methods with objectives defined on external datasets. Priors learned from those external datasets may not generalize well, so one needs to finetune or re-train on the new distribution if the target distribution changes. In comparison, the only supervision in our method is the intrinsic driving force of diversity and novelty constraint, which does not rely on external datasets.

To illustrate how our novelty metric helps a vanilla synthetic data generator, we experiment on shape-from-shading of a single object. We experiment in a zero-shot generalization setting: there is no validation set; the test distribution is completely unknown. The only supervision is the novelty metric and nothing else, and the task networks are trained only on the synthetic images and have never seen a real image. We compare with prior supervised methods on MIT-Berkeley Intrinsic Images [8] (MBII) to show that unsupervised method is able to achieve reasonable performance even with zero knowledge of real data. We also test the algorithms on additional renderings of ShapeNet [17] and Pix3D [147]. To demonstrate the flexibility of the framework, we also experiment on intrinsic image decomposition on the Intrinsic Images in the Wild (IIW) dataset [9]. We show that novelty-guided dataset creation is a powerful tool for producing useful synthetic 3D training data.

In a broader context, our method contributes to synthetic data generation in a socially responsible way. First, our method does not require external real images. As a result, our method alleviates the privacy concerns for data collection in 3D perception, in which rich annotated 3D ground truth is preferred. Furthermore, since randomly generated content does not correspond to real identities: shapes and scenes are completely random and do not associate with real-world counterparts. This perfectly avoids privacy issues in training deep networks. Second, our novelty-guided synthetic generator can in some way provide alternatives to copyrighted content such as 3D meshes and scene collections. It is likely that artists and designers may inspire from a novelty-guided data generator for their own creative content. Last, we have reasons to believe our direction can potentially resolve the biases in synthetic data. Our novelty metric has the nature of promoting diversity in the data: underrepresented individual genomes are encouraged to stay in the population. Our experiment results have also shown that such diversity is important to improve the downstream task performance.

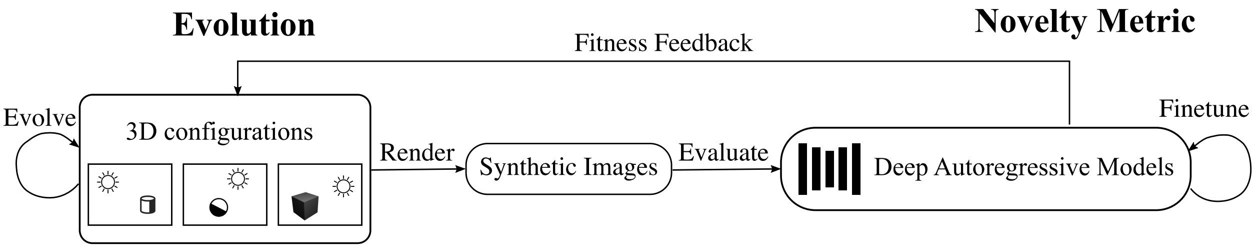


Figure 4.1: The overview of our method. We evolve a set of 3D configurations as genomes, with fitness scores defined as novelty. The genomes are first rendered into synthetic images. The novelty is then computed by applying a deep autoregressive model to calculate bits/subpixel for encoding the image. This indicates how novel a genome is in terms of previously seen images.

4.2 Related Work

4.2.1 Learning to Generate Synthetic Data

In our work, we optimize a synthetic 3D generator. A common direction along these lines is to guide optimization with the objective of matching a real distribution [156, 63, 112, 65, 86]. The parameters of the generators are directly learned from real datasets. For generating synthetic data that are specifically for training deep networks, researchers also integrate network generalization performance into the optimization pipeline [171, 123, 65, 20, 87, 169]. The deep network is evaluated on an external dataset or environment to evaluate the synthetic generator. Specifically, in Chapter 2 we evolve a set of 3D shapes while training a number of deep networks. The networks are then evaluated on a real validation set to obtain the fitness scores for individual shapes.

However, the supervision source is only a partial observation of a true test distribution. The synthetic distribution may become homogeneous and only resemble the observations, thus overfitting the limited set of real images. In this work, we tackle this problem by explicitly promoting novelty and diversity in the generation process. The generator is continuously pressured to be creative, therefore avoiding the collapse of the synthetic distribution.

4.2.2 Novelty Search

In evolutionary computation, the general concept of novelty search has long been a useful addition to performance optimization [143, 98, 161, 73, 72]. The key idea is promoting divergence and difference from others, which frequently occurs in natural evolution [73, 161]. In those works, various diversity-rewarding novelty search algorithms have been proved effective in discovering control policies for maze and biped walking domains.

Recent works that use deep policy networks have also applied novelty search to accelerate policy finding [145, 31, 158, 40]. Novelty in these works is defined as behavioral diversity, which encourages agents that behave differently from previous ones. It is computed using the average distance between k-nearest neighbors in a well-defined parameter space.

Our work is inspired by this line of works, with the target of generating 3D synthetic data through evolution: the 2D synthetic image is encoded as its source 3D configuration in evolution. In this case, designing novelty metric is non-trivial, compared to prior research in which sparsity and diversity is often well defined. As pointed out by Lehman and Stanley [73], the representation of the encoding and the morphology plays a vital role in novelty search. In this work, we propose an innovative method of using autoregressive models to model novelty in 3D synthetic data generation, which has never been done in previous literature.

The most related work that uses novelty to generate interesting images is Innovation Engine by Nguyen et al. [103]. They use Compositional Pattern Producing Networks (CPPN) to create 2D patterns that are novel in deep network feature space. They create artistic 2D images that visually resemble real-world objects such as “obelisk”, “mosque”, “church”. Meanwhile, their method is not readily applicable to generating training data for 3D perception since either 3D or network training is not involved.

4.2.3 Curiosity-driven Learning

In reinforcement learning (RL), curiosity is used as an intrinsic motivation to discover new, novel patterns [131, 124, 132, 104, 130, 5, 10, 45, 106, 100, 146]. Compared to sparse extrinsic rewards, intrinsic motivation is dense and is able to actively search in all interesting directions. To some extent, our novelty score for 3D configurations can be considered as an intrinsic motivation for the synthetic data generator. However, a core difference is that in this chapter we are learning a content generator that produces novel 3D configurations and renders to synthetic training data, rather than a RL agent that learns to explore and interact with a pre-defined environment.

4.2.4 Novelty Detection

The task of novelty detection is to identify test data that differ from the training distribution. It is typically approached as a one-class classification task of detecting the anomalies or outliers in security-critical applications [110]. In image domain, recent works largely exploit deep models such as deep classifiers [97], autoencoders [165, 125], Generative Adversarial Nets [115, 126, 108] and adversarial autoencoders [109, 160]. Specifically, Xia et al. [165] and Sabokrou et al. [126] utilize the reconstruction error of an autoencoder to separate the inliers and outliers: high reconstruction error indicates out-of-distribution samples and vice versa. Pidhorskyi et al. [109] learns a probabilistic adversarial autoencoder to explicitly model the distribution of inliers. The probabilities are then thresholded to detect the outliers. In these inspiring works, the focus is to effectively detect anomalies with a discriminative or generative model; the training set is fixed when learning the classifier.

In our work, we also compute novelty scores, but they are vastly different: the novel images are not anomalies, but interesting valid synthetic images that are rendered by a 3D synthetic data generator. In addition, these novel synthetic samples are added back to the training set to further adjust the novelty model; the novelty model is constantly being updated to ignore seen images and discover new 3D configurations, instead of serving as a static classifier as in novelty detection.

That said, our flexible pipeline allows swapping in any novelty model. This means our approach will certainly benefit from the significant progress on novelty detection, since more accurate novelty evaluation will better guide the evolution of 3D parameters.

4.3 Methodology

Our framework includes two main components: the evolution process and the novelty model. The evolution process continues to generate new 3D configurations with the guidance from the novelty model, and the novelty model is constantly finetuned with all previously seen images in each evolution epoch. Our full algorithm is listed in Algorithm 1.

Algorithm 1: Novelty-Guided Evolution with Autoregressive Models

```
**Result:** A set of synthetic training data
begin
    $\{g_1^{(0)}, g_2^{(0)}, \cdots, g_n^{(0)}\} = \text{InitializePopulation}(n);$
    $M^{(0)} = \text{InitializePixelCNN}();$
    for $k = 0, 1, \cdots, K-1$ do // Evolution
        $\{g_{n+1}^{(k)}, g_{n+2}^{(k)}, \cdots, g_{n+n'}^{(k)}\} = \text{GeneticOperator}(\{g_1^{(k)}, g_2^{(k)}, \cdots, g_n^{(k)}\}, n');$
        for $i = 1, 2, \cdots, n+n'$ do
            $I_i^{(k)} = \{I_{i,1}^{(k)}, I_{i,2}^{(k)}, \cdots, I_{i,m}^{(k)}\} = \{\text{render}(g_i^{(k)}, r_j^{(k)}) \mid j = 1, 2, \cdots, m\};$
            $f_i^{(k)} = \frac{1}{m}\sum_{j=1}^{m} \text{EvaluateBitsPerSubpixel}(M^{(k)}, I_{i,j}^{(k)});$
        end
        $M^{(k+1)} = \text{train}(M^{(k)}, \{I_i^{(k')} \mid k' \leq k, i \leq n+n'\});$
        $\{g_1^{(k+1)}, g_2^{(k+1)}, \cdots, g_n^{(k+1)}\} =$
          $\text{EvolutionOperator}(g_1^{(k)}, f_1^{(k)}, g_2^{(k)}, f_2^{(k)}, \cdots, g_{n+n'}^{(k)}, f_{n+n'}^{(k)})$
    end
    return $\{I_{i,j}^{(k)} \mid i \leq n+n', k \leq K\}$
end
```

4.3.1 Evolution of 3D Configurations

We adopt the genetic algorithm: we represent a set of n 3D configurations as a population of genomes $\{g_1, g_2, \cdots, g_n\}$. In practice, a 3D configuration g_i can include 3D shape instances, the poses of objects in a scene, background wall textures, and camera parameters. We first initialize the population, then evolve it until a given number of images are reached. During each epoch, we create a set of n' offsprings using genetic operators such as mutation and crossover, then evaluate these new configurations along with the current population of n genomes. Among $n + n'$ total genomes, n are selected through evolutionary operators such as roulette. These n genomes form the population for the next epoch.

The genetic operators of mutation and crossover can be easily adapted to the 3D configuration. For example, crossover and mutation can be defined as boolean operations and linear transformations on synthetic 3D shapes (Section 2.3.1). The mutation on lighting is defined as randomly shifting source position for point light or emitting directions for directional lighting. For genomes that include multiple sections (shapes, lighting, cameras), crossover and mutation can be defined as random swapping of each other's sections and mutations on individual segments.

Our evolutionary space for 3D configurations is flexible and does not take a standard, fixed form. In the experiment (Section 4.4) we show more detailed examples of how those are implemented for different scenarios.

4.3.2 Phenotypes

If every scene parameter is defined in a genome g_i in the population, the genome g_i can uniquely render to one single image. Our framework also allows partially defined scenes (such as only shapes instances). In this case, g_i can be rendered into multiple images (and 3D ground truth) $I_{i,1}, I_{i,2}, \cdots$ by randomly varying other free scene parameters r multiple times. For the i-th genome at evolution epoch k, we render m samples as the phenotype for the genome g_i:

$$I_i^{(k)} = \{I_{i,1}^{(k)}, I_{i,2}^{(k)}, \cdots, I_{i,m}^{(k)}\} = \{\text{render}(g_i^{(k)}, r_j^{(k)}) \mid j = 1, 2, \cdots, m\} \tag{4.1}$$

4.3.3 Novelty Evaluation with Autoregressive Models

In evolution, we need to evaluate each genome and associate it with a novelty score. Such novelty score is based on whether the corresponding rendered images (*i.e.* the phenotype, $I_i^{(k)}$) are novel. In novelty search with well-defined behavioral space, this can be done by computing the average distance to neighbors in a high dimensional space, but it is not immediately applicable for synthetic images in specific domains.

To determine whether a rendered image is novel, we use an autoregressive model [128]. It is a specialized deep network, a generative model of images, that can be trained on a dataset and can evaluate the probability of an incoming image. The probability can be further converted into how many bits it needs to store for each pixel in each channel (*i.e.* subpixel) in average. Given a trained model, if an image needs larger bits to encode, more information exists in the image, which means the image is less similar to the ones the model is trained on. Therefore, it perfectly fits our need for evaluating the novelty. That said, our framework allows plugins of any trainable novelty model, as mentioned in the related work (Section 4.2.4). Our choice of the model PixelCNN [128] is based on its explainable metric (bits/subpixel) and the availability of the off-the-shelf implementation.

Given a trained PixelCNN and a genome, we simply evaluate the bits/subpixel of the corresponding rendered images and use the mean as the novelty fitness.

4.3.4 Joint Update of Autoregressive Models

The evolution process relies on the autoregressive model to determine the next population. In turn, the autoregressive model also needs to update to include the newly generated data. Recall that we render a genome into m images by varying free scene parameters. Prior to the k-th evolution epoch, for a population of size n with n' offsprings, we have already rendered $m(k-1)(n+n')$ in total. The autoregressive model $M^{(k-1)}$ is finetuned on all of these images. This finetuned model $M^{(k)}$ is then used for novelty evaluation for the k-th population.

4.4 Experiments

4.4.1 Single Shape Normal Recovery

Same as in Chapter 2, we experiment on the shape-from-shading task. The input is an image of a single shape and the output is pixelwise normal prediction. The task network is a Stacked Hourglass Network [102]. We test our algorithm on the MIT-Berkeley Intrinsic Images (MBII) dataset [8]. In MBII, the input is real images and the ground truth normals are computed through an optimization algorithm. For additional testing, we also render 100 shapes from ShapeNet [17] and 94 shapes from Pix3D [147].

In our method, the genotype is the shape mesh itself. The corresponding phenotype is a set of rendered images with varied lighting and shape rotations. Initially, the population is a set of primitive shapes such as spheres, cubes, cones and cylinders. These shapes are then combined to create more complex shapes in the evolutionary process. Each shape is assigned a fitness score, and then the roulette operator is used for selecting the next population. To compute the fitness score, we train a PixelCNN with the version from [128]. The fitness score is exactly the novelty metric, *i.e.* the average bits/subpixel of the rendered images associated with each shape. Since the background is empty pixels and always dark, we only consider the masked area when computing the average bits/subpixel.

We then collect all the genomes throughout the evolution and render them into training samples. The task network is then trained on all these training samples until converged. We then directly evaluate this network on the test set of MBII and rendered ShapeNet and Pix3D images.

Baselines We compare to a line of supervised methods [8, 171, 169] that optimize their synthetic generation process using the training split of MBII. We also compare to an ablated version of our method by setting the novelty to a constant (No novelty). For SIRFS [8], it is an optimization-based method with pre-defined priors. Since they have a few parameters to tune, their model needs to be trained on the training set of MBII. For [171], it is an evolutionary approach shown in Chapter 2. In Section 2.5.2, in order to compute a fitness score for one single genome, we need to train a task network on the generated images, and evaluate it on the MBII training set. The generalization performance of the network is then used as fitness scores. The number of networks trained in each epoch is the same as the population size. In this chapter, we only finetune one PixelCNN during evolution, and train one task network on all data after evolution is finished. In Chapter 3, we have achieved the state-of-the-art performance for this task on MBII, using an improved optimization strategy: we propagate the validation performance on the MBII training set back to adjust the generator parameters.

Evaluation Following prior works, we evaluate the percentage of normal prediction whose errors are not larger than a threshold ($\leq 11.25°$, $\leq 22.5°$, $\leq 30°$), mean angle error (MAE), median angle error, and mean squared error (MSE) between the normal prediction and ground truth normal on all pixels of the MBII test set.

Table 4.1: The evaluation results on MBII [8]. Compared to supervised methods, we are able to achieve reasonable results even without seeing a single image in MBII. Compared to the baseline "No novelty", our novelty metric helps to generate novel shapes such that the trained network can generalize better on unseen images.

		Summary Stats ↑			Errors ↓		
		≤ 11.25°	≤ 22.5°	≤ 30°	MAE	Median	MSE
Supervised	SIRFS* [8]	20.4%	53.3%	70.9%	26.2°	—	0.2964
	Shape Evolution [171]	21.6%	55.5%	73.5%	23.3°	—	0.2204
	Hybrid Gradient [169]	**24.5%**	59.3%	**74.3%**	**22.0°**	18.9°	**0.1984**
Unsupervised	No novelty	17.1%	46.7%	63.2%	27.5°	23.9°	0.3039
	Novelty	**19.4%**	**52.3%**	**68.4%**	**25.6°**	**21.6°**	**0.2697**

Table 4.2: Results on ShapeNet [17] and Pix3D [147] renderings. We obtain the model from [169] and directly evaluate on this dataset. The results show that the network trained on our synthetic dataset generalizes better.

		Summary Stats ↑			Errors ↓		
		≤ 11.25°	≤ 22.5°	≤ 30°	MAE	Median	MSE
ShapeNet	Trained Model from [169]	14.3%	41.3%	58.3%	30.1°	26.1°	0.3552
	No novelty	21.0%	49.8%	66.0%	28.1°	22.6°	0.3428
	Novelty	**21.3%**	**52.2%**	**67.5%**	**27.7°**	**21.6°**	**0.3356**
Pix3D	Trained Model from [169]	11.3%	36.6%	54.8%	31.2°	27.7°	0.3552
	No novelty	17.8%	46.9%	62.2%	28.5°	23.9°	0.3353
	Novelty	**18.4%**	**48.1%**	**63.4%**	**28.0°**	**23.3°**	**0.3250**

Results on MBII The results on MBII are listed in Table 4.1. The results of supervised methods from Table 3.2. Note that it is an unfair comparison since the supervised methods are trained on the training split of MBII, where they are able to learn the priors in MBII for generating synthetic data. In contrast, our trained network has never seen an real image, but is still able to achieve reasonable performance. The ablation "No novelty" is worse than our full pipeline, demonstrating that novelty is able to generate better synthetic data. We show the images of most/least novel shapes in Figure 4.2 across different evolution epochs, and plot the distribution of fitnesses and the training trajectory of PixelCNN in Figure 4.3 and 4.4.

Results on ShapeNet and Pix3D In Table 4.2, we directly use the trained model in Chapter 3 and evaluate on these two datasets. The trained model is doing worst on both datasets, indicating that the learned priors from MBII are difficult to generalize to other datasets. In comparison, our full pipeline with novelty metric is able to achieve reasonable performance. Our novelty metric always helps, regardless of choices of test datasets.

4.4.2 Intrinsic Images in the Wild

To demonstrate that our novelty-guided evolution can be easily adapted to other scenarios, we experiment on the Intrinsic Images in the Wild [9] dataset. The task is to recover the reflectance image given a single image as input. We use the same stacked hourglass network as the task network. For genome, we represent a scene with a set of shapes, a set of point lights in a textured cuboid. The shapes are sampled from ShapeNet [17]. The sampled scenes are shown in Figure 4.5.

For crossover, we combine different segments from parent genomes to create offsprings. For mutation of shapes, the shape instances have a probability of mutating to another random shape instance from ShapeNet. We also allow random rotation, scaling and translation of shapes inside the cube. For mutation of lighting, we randomly adjust the position and RGB intensity of the light source.

We evaluate the weighted human disagreement rate (WHDR) with a threshold of 10%, which indicates how much the prediction agrees with human judgement (lower the better). The WHDR is computed on all edges in all images in the IIW dataset. Without novelty, the task network trained on synthetic data is able to reach a WHDR of 34.8%. After adding novelty, we are able to achieve 33.9%. Note that our task network has never seen an image in IIW. This shows that using novelty is able to encourage the pipeline to produce more synthetic scenes that can improve the generalization performance of the task network. We also demonstrate how novelty helps in terms of eliminating trivial cases in Figure 4.6.

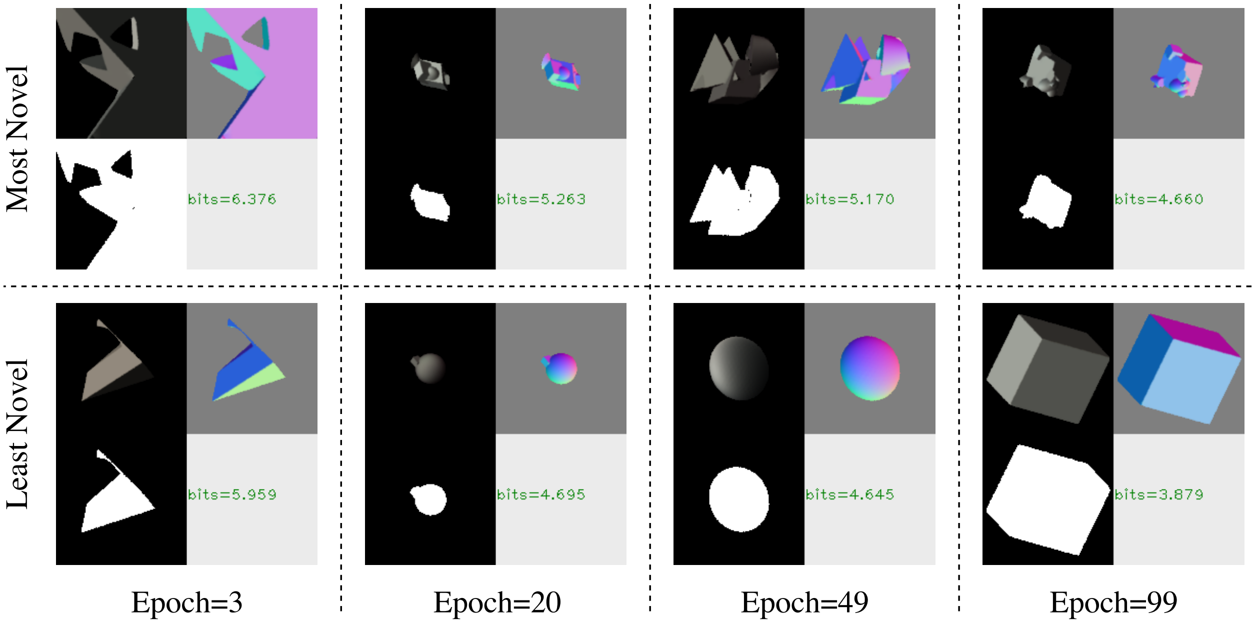


Figure 4.2: The most and least novel shapes at different epochs in the evolution process. The PixelCNN trained on synthetic shapes is able to assign a low novelty score for primitive shapes such as spheres and cubes, and assign a high score for interesting compositions. Top left: input image Top left: input image. (Top right: visualization of ground truth normals. Bottom left: mask. Bottom right: novelty score.)

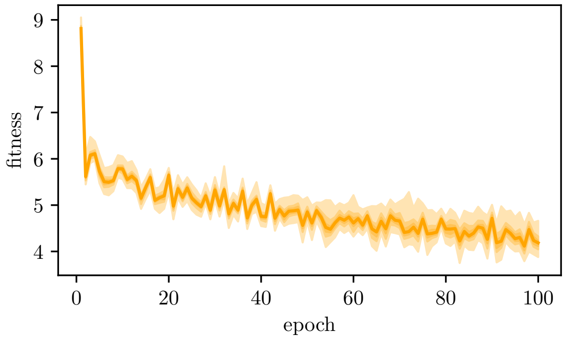


Figure 4.3: The fitness distribution changes over time. Different shades represent minimum, maximum and 25%, 50%, 75% percentiles.

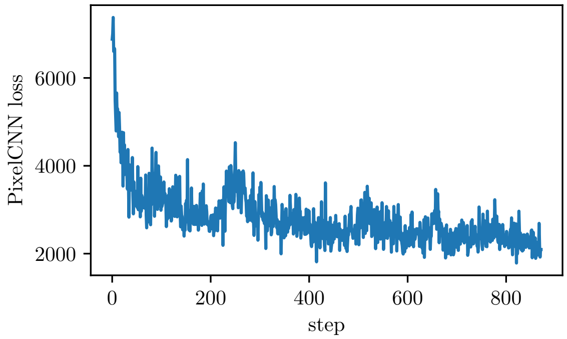


Figure 4.4: The trajectory of training PixelCNN over time.

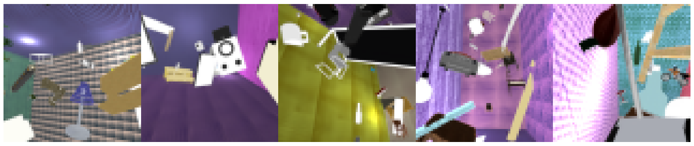


Figure 4.5: Randomly sampled scenes in the initial population. The objects and randomly placed inside a cube with background textures. The light sources are point lights with random locations and color intensity.

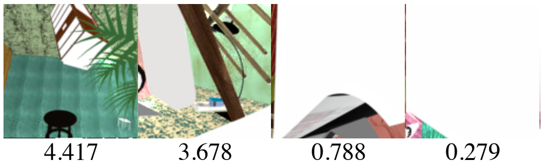


Figure 4.6: The novelty scores for novel random scenes. It learns to assign trivial cases with a low score, such as occluded cameras or degraded lighting.

CHAPTER 5

Conclusion

5.1 Advantages

Through the exploration of supervised and unsupervised learning, we conclude that our approaches in both directions have helped us improve an automatic generation pipeline to produce useful synthetic training data.

In Chapter 2, we show initial prototypes of automatically evolving pure synthetic shapes that can help deep networks on shape-from-shading tasks. We have introduced an algorithm to jointly evolve 3D shapes and train a shape-from-shading network through synthetic images. We show that our approach can achieve exceptional performance on real images without using an external shape dataset. Our framework could also be applied to robotic tasks such as object manipulation: we only need to change the usefulness metric of a shape, from how well a trained task network can generalize, to how well a trained RL agent can perform in a synthetic environment.

In Chapter 3, we further formulate the problem setting for more generalized scenarios such as human faces and indoor scenes. We have proposed hybrid gradient, a novel approach to the problem of automatically optimizing a generation pipeline of synthetic 3D training data. We evaluate our approach on the task of estimating surface normal, depth and intrinsic decomposition from a single image. Our experiments show that our algorithm can outperform the prior state of the art on optimizing the generation of 3D training data, particularly in terms of computational efficiency.

In Chapter 4, we design an unsupervised task-agnostic and dataset-agnostic approach that can universally help a random synthetic data generator to produce more diverse images, which in turn implicitly help the task network to generalize.

5.2 Limitations and Future Work

This dissertation has shown promising results in optimizing an automatic synthetic generation pipeline for 3D perception tasks. We minimize the human designs in our framework and propose novel algorithms to optimize the generation pipeline. Meanwhile, we also need to keep in mind that completely automatic generation of synthetic data has not been fully applied in research and industry; we still have a long way to go in order to scale our algorithms to fit today's computationally-intensive deep neural networks on 3D perception tasks. There are multiple factors that can affect the optimization of the automatic pipeline, but are not systematically studied in this dissertation. We still need to overcome these obstacles in order to apply our algorithms to large-scale settings.

Cost of meta-optimization Although we enjoy cheaper and faster dedicated hardware for deep learning and computer graphics, the state-of-the-art models that researchers have developed also become larger in a faster pace. In Chapter 2, the tweaking of the generation pipeline can be seen as meta-optimization, with rendering synthetic images and training deep networks as inner optimization. Each outer iteration requires a large number of inner optimization, which is rendering the synthetic dataset and training a deep network to convergence. We alleviate the cost of training and rendering using joint training and evolution (Section 2.5.2), while this may affect the optimization landscape for meta-optimization and could result in suboptimal performance.

In Chapter 3 we have successfully reduced the number of training instances to be as low as one, but the cost of sampling and rendering of the synthetic generator is still large. The unrolling of training step also requires large memory space in order to enable back-propagation over multiple steps. Besides, a large number of sampled synthetic images are used only to compute the gradient and are discarded afterward.

In Chapter 4 we avoid meta-optimization using unsupervised learning. In this way, we further maximize the use of rendering: every single image rendered is included in the final dataset, and the only overhead is the endless training and finetuning of the novelty evaluator. Furthermore, there is no training or evaluation of downstream task networks inside this optimization process. The downside is that on specific datasets for specific tasks, the performance may not be as good as supervised learning using meta-optimization.

Therefore, in the future, more efficient and faster meta-optimization algorithms need to be developed to fully enable the potential of an automatic synthetic generation pipeline.

Capacity of a deep neural network In this dissertation, we assume the difficulty of 3D perception mainly lies in the training data, instead of the task network capacity. For example, in Chapter 4, we simply assume that more diverse training images will eventually help a deep network to generalize in unseen settings. The network capacity is sufficient for the experiments in this dissertation, so the assumption is adequate. However, the capacity may become a bottleneck when the task itself is extremely difficult and the synthetic data design space is huge. Specifically, in an ideal setting, synthetic samples are diverse enough to cover test examples that the deep network can encounter in the real world. However, a deep neural network with limited capacity can not perform perfectly on the diverse training set due to its capacity constraint. Instead, it is optimized towards average performance. This means it is possible that endless diverse examples may harm the trained network in terms of performance on a limited set of real test examples.

Therefore, we believe building optimal network architecture and larger deep models are as important as developing automatic synthetic generation pipelines. They have to complement each other in order to reach human-level 3D perception ability.

5.3 Broader Implications

As researchers, while our studies may not immediately affect the environment and society, we bear the responsibility of our technologies in ethical, legal and social implications. Therefore, it is necessary for us to include a discussion of what kind of effect our algorithms may potentially have in those aspects.

To summarize, for human-centric synthetic data generation, we firmly believe that using an automatic synthetic pipeline will advocate privacy, creativity and fairness in 3D synthetic data generation.

Privacy Today's dataset collection is based on the assumption that the distribution divergence between training and test should be minimized. This causes researchers to collect data that are close to application scenarios. Especially for 3D perception using deep learning, large-scale datasets of real indoor/outdoor scenes that include rich annotations are common practice. Collecting datasets with this natural tendency can easily intrude people's privacy and even be exploited for malicious use, especially for 3D perception in which dense 3D ground truth may contain sensitive information.

This dissertation potentially alleviates the privacy concerns of collecting data. First, we minimize the real-world examples in our approaches. In Chapter 2 and 3, we only require one single value from a tiny set of real examples. In Chapter 4, we refrain from using any external data for optimizing the synthetic generation pipeline. In addition, randomly generated synthetic content does not correspond to any real identities. For example, our generated shapes and scenes are completely random and are not related to any shapes or scenes in the real world. The synthetic human faces (Figure 3.5, left) have a zero probability of corresponding to a real human. Therefore, we believe our method can improve privacy in deep learning in general.

Creativity Careless use and collection of online CAD models can easily cause copyright infringement, especially for acquiring large-scale datasets for training deep networks. Our automatic synthetic generation pipeline can relieve such concern and provide innovation in designing 3D shapes and scenes. In this dissertation, our novelty-based generation framework encourages automatic synthesis of novel, interesting content, such as 3D meshes and scenes. The automatically generated content can be an alternative to copyrighted content such as 3D shapes, scene collections that are not in public domain. Artists and designers may inspire from a collection of 3D assets generated using our method. Automatic optimization as an innovation source for artists has been seen in prior works [29, 103].

Fairness Datasets that generated by an automatic synthetic generator may be prone to biases, especially when the generator is learned from partial observations of the real world. For example, the demography of a human face dataset may impact a learned synthetic human face generator, thus further creating hidden biases for downstream deep models. This could potentially affect our supervised learning approaches (Chapter 2 and 3), so we should carefully choose the real datasets when used as the supervision source.

We also point a potential direction for this. In Chapter 4, our novelty-guided evolution in some way promotes the diversity of the data: underrepresented data are encouraged to stay in the population to play a more important role, thus potentially reducing the biases for the networks trained on our data.

However, we also need to be cautious about the novelty model used in this approach. A properly designed novelty model can increase the sparsity and the diversity of the data and can potentially reduce the bias in the generated data. Meanwhile, it is not a panacea for eliminating all biases existing in training the deep networks, as the data, the task network, and the downstream applications all may lead to societal implications. It is also possible that the novelty model has biases itself, creating the deceptive delusion of such problems being solved, which is even more dangerous. Therefore, we need to closely and thoroughly inspect the technology behind before putting our work into production.

5.4 Summary

This dissertation is a tiny step towards our holy grail of human-level 3D perception ability. We approach the problem using automatic generation of synthetic data, and optimize the generation process using supervised learning and unsupervised learning methods. Multiple experiments have shown that our approaches are promising down this path. We also discussed about limitations and possible future work, along with research responsibility in a broader context. We firmly believe that although there might be obstacles ahead, we are on the right path to lead to our end goal of human-level 3D perception ability.

APPENDIX A

Learning to Generate 3D Synthetic Data through Hybrid Gradient

A.1 Metrics

Here we detail the metrics that we used in the paper. Assume $\mathbf{n}_i$ and $\mathbf{n}_i^*$ are the unit normal vector at i-th pixel (of N total) in the prediction and ground truth normal maps, respectively. d_i and d_i^* are depth values of the i-th pixel in the prediction and ground truth depth maps, respectively.

- Mean Angle Error (MAE): $\frac{1}{N}\sum_i \arccos(\mathbf{n}_i \cdot \mathbf{n}_i^*)$
- Median Angle Error (MAE): $\underset{i}{\mathrm{median}}[\arccos(\mathbf{n}_i \cdot \mathbf{n}_i^*)]$
- Threshold δ: Percentage of $\mathbf{n}_i$ such that $\arccos(\mathbf{n}_i \cdot \mathbf{n}_i^*) \leq \delta$
- Mean Squared Error (MSE): $\frac{1}{N}\sum_i [\arccos(\mathbf{n}_i \cdot \mathbf{n}_i^*)]^2$
- Absolute Relative Difference: $\frac{1}{N}\sum_i |d_i - d_i^*|/d_i^*$
- Squared Relative Difference: $\frac{1}{N}\sum_i (d_i - d_i^*)^2/d_i^*$
- RMSE (linear): $\sqrt{\frac{1}{N}\sum_i (d_i - d_i^*)^2}$
- RMSE (log): $\sqrt{\frac{1}{N}\sum_i (\log d_i - \log d_i^*)^2}$
- RMSE (log, scale-invariant): $\sqrt{\frac{1}{N}\sum_i (\log d_i - \log d_i^* \cdot [\frac{1}{N}\sum_i (\log d_i - \log d_i^*)])^2}$

A.2 MIT-Berkeley Intrinsic Image Dataset

Our decision vector β for PCFG is a 29-d vector, with 4 dimensions representing the probabilities of sampling different primitives, 2 for sampling union or difference, 1 for whether to expand the tree node or replace it with a terminal, 6 for translation mean/variance, 6 for scaling log mean/variance, 2 for sphere radius log mean/variance, 2 for box length mean and variance, 4 for cylinder radius and height log mean/variance, and 2 for tetrahedron length log mean/variance.

For optimizing β, we use the mean angle error loss on the validation set as the generalization loss. Note that some dimensions of β are constrained (such as probability needs to be non-negative), so we simply clip the value of β to valid ranges when sampling near β for finite difference computation and updating β. We present the qualitative results in Figure A.1.

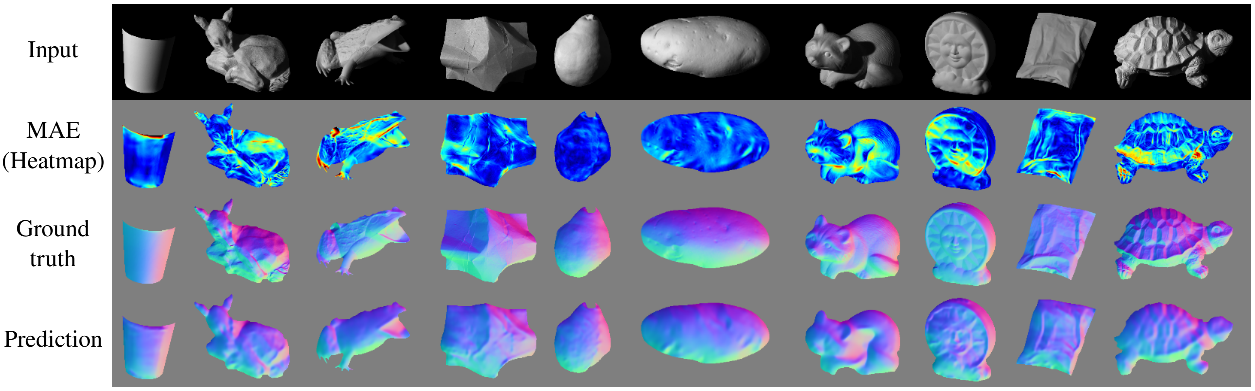


Figure A.1: The test set of the MIT-Berkeley Intrinsic Images dataset.

A.3 NYU Depth V2

The decision vector β is 108-d. It includes the parameters for mixtures of Gaussians/Von Mises for 6 degree-of-freedom (vertical, horizontal and fordinal displacement, yaw, pitch, roll rotation) for shapes in the scene and the camera. Each mixture contains 9 parameters (3 probabilities, 3 means and 3 variances). Examples of perturbed scenes and the original scenes are shown in Figure A.2. The distributions for translation perturbation of shapes are shown in Figure A.5.

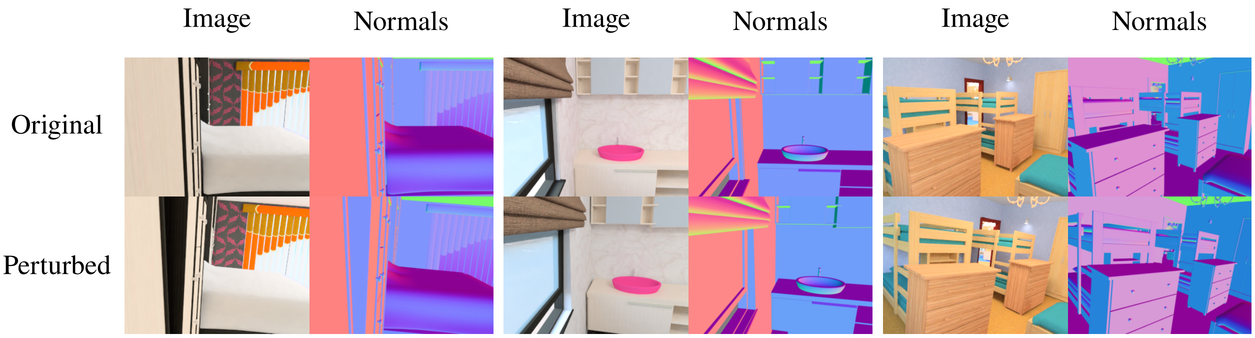


Figure A.2: The original scenes in the SUNCG dataset, and our scenes with camera and objects perturbed using our PCFG.

A.4 Basel Face Model

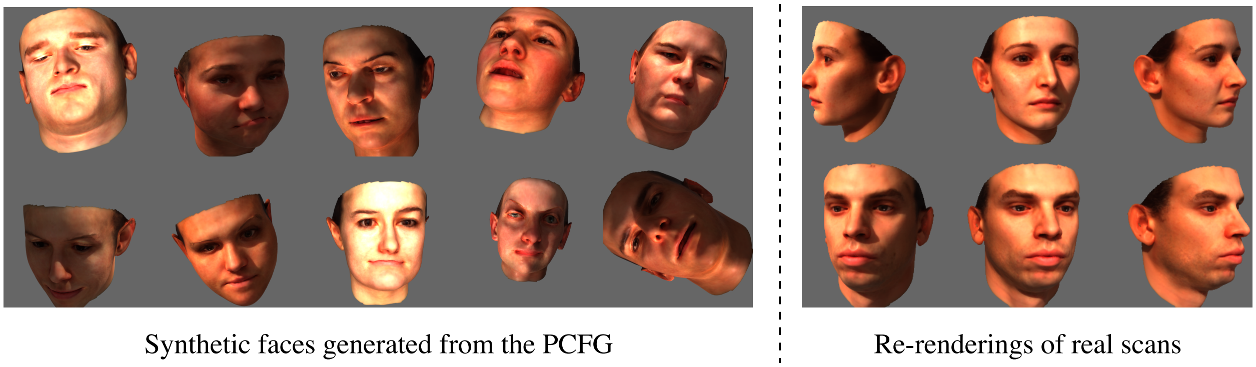
The decision vector β has 204 dimensions. We use an off-the-shelf 3DMM implementation[1] to generate face meshes and textures for training. The 3DMM has 199 parameters for face identity, 29 for expression and 199 for texture. In implementation, we use 10 principal dimension for face/expression/texture respectively, and randomly sample from a mixture of 3 multivariate Gaussians. Note that the dimensions are independent, so we have a total of 183 parameters for generating the face mesh. For the 3-dof face pose angle, we also use mixtures of 3 von Mises, which have 21 parameters in total.

For rendering the training set, we apply a human skin subsurface model using Blender [13], with a random white directional light uniformly distributed on $-z$ hemisphere. For rendering the test set (the scanned faces in the Basel Face Model), we render with the same 3 lighting angles and 9 pose angles, and the same camera intrinsics as in the original dataset. Figure A.3 shows the training images randomly generated by the PCFG (left) and example test images(right).

A.5 Synthetic Texture Generation for Intrinsic Image Decomposition

The decision vector β has 36 dimensions. To sample a texture, we first sample the number of polygons using a zero-truncated Poisson distribution. For each polygon, we then sample the number of vertices (from 3 to 6) according to the probabilities specified in β. The vertex coordinates of the

[1] https://github.com/YadiraF/face3d



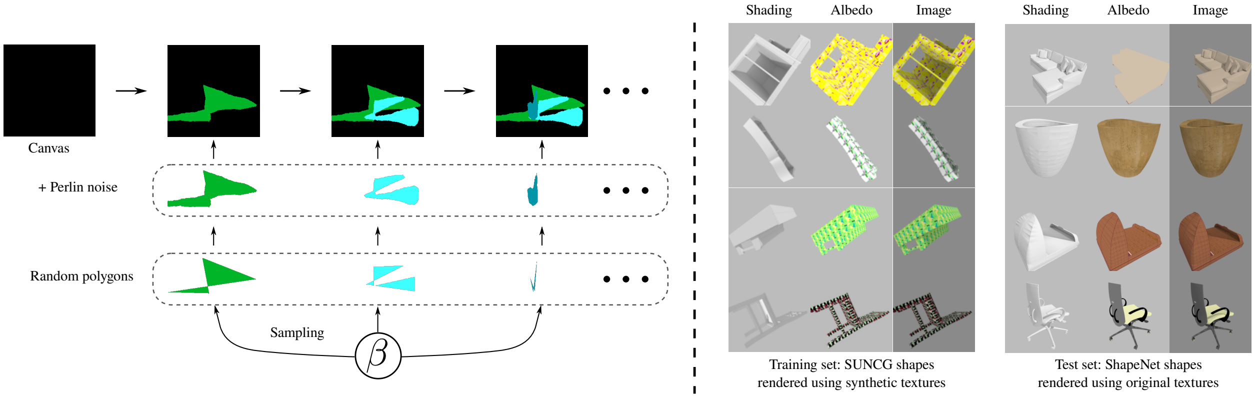
Synthetic faces generated from the PCFG

Re-renderings of real scans

Figure A.3: Training images generated using PCFG with 3DMM face model, and 6 example images from the test set.

polygon follow mixed truncated Gaussians. The polygons are then perturbed using Perlin noise: we first build the signed distance map to the boundary of the polygons, and then perturb the distance using Perlin noise. Finally, we re-compute the boundary according to the distance map to produce the perturbed polygons. These perturbed polygons are then painted onto a canvas to form a texture. This procedure is shown in Figure A.4.

For rendering, we assume the shading is greyscale, and set up a random white directional light. We use Blender [13] to render the albedo image and the shading image, then multiple the two images together as the final rendered image. We render SUNCG [140] shapes using our synthetic textures for training, and render ShapeNet [17] with original textures for validation and test. The examples are also shown in Figure A.4.



Training set: SUNCG shapes rendered using synthetic textures

Test set: ShapeNet shapes rendered using original textures

Figure A.4: Our texture generation pipeline and example images of the training and test set.

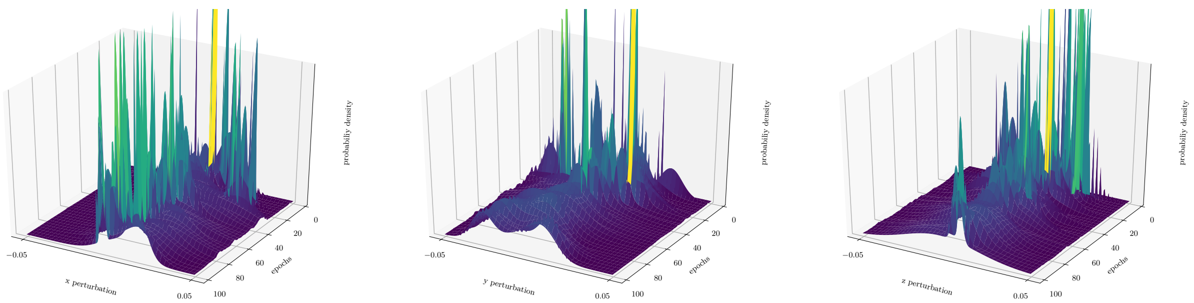


Figure A.5: How probability distributions change over time for SUNCG perturbation parameters. The three images plot the probability density of shape displacement along x, y, z axes respectively.

BIBLIOGRAPHY

[1] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3981–3989, 2016.

[2] M. Aubry, D. Maturana, A. Efros, B. Russell, and J. Sivic. Seeing 3d chairs: exemplar part-based 2d-3d alignment using a large dataset of cad models. In *CVPR*, 2014.

[3] N. Baba. Convergence of a random optimization method for constrained optimization problems. *Journal of Optimization Theory and Applications*, 33(4):451–461, Apr. 1981.

[4] A. Bansal, B. Russell, and A. Gupta. Marr revisited: 2d-3d alignment via surface normal prediction. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[5] A. Baranes and P.-Y. Oudeyer. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1):49–73, 2013.

[6] J. T. Barron and J. Malik. High-frequency shape and albedo from shading using natural image statistics. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 2521–2528. IEEE, 2011.

[7] J. T. Barron and J. Malik. Shape, albedo, and illumination from a single image of an unknown object. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 334–341. IEEE, 2012.

[8] J. T. Barron and J. Malik. Shape, illumination, and reflectance from shading. *TPAMI*, 2015.

[9] S. Bell, K. Bala, and N. Snavely. Intrinsic images in the wild. *ACM Trans. on Graphics (SIGGRAPH)*, 33(4), 2014.

[10] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count-based exploration and intrinsic motivation. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 1471–1479. Curran Associates, Inc., 2016.

[11] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pages 2546–2554, Granada, Spain. Curran Associates Inc., 2011.

[12] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(1):281–305, Feb. 2012.

[13] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation. Blender Institute, Amsterdam, 2019.

[14] E. Brochu, V. M. Cora, and N. de Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *CoRR*, abs/1012.2599, 2010. arXiv: `1012.2599`.

[15] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black. A naturalistic open source movie for optical flow evaluation. In A. Fitzgibbon et al. (Eds.), editor, *European Conf. on Computer Vision (ECCV)*, Part IV, LNCS 7577, pages 611–625. Springer-Verlag, Oct. 2012.

[16] A. Chakrabarti, J. Shao, and G. Shakhnarovich. Depth from a single image by harmonizing overcomplete local network predictions. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2658–2666. Curran Associates, Inc., 2016.

[17] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[18] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. Niessner, M. Savva, S. Song, A. Zeng, and Y. Zhang. Matterport3d: learning from rgb-d data in indoor environments. *International Conference on 3D Vision (3DV)*, 2017.

[19] C. Che, F. Luan, S. Zhao, K. Bala, and I. Gkioulekas. Inverse transport networks. *arXiv preprint arXiv:1809.10820*, 2018.

[20] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. D. Ratliff, and D. Fox. Closing the sim-to-real loop: adapting simulation randomization with real world experience. In *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*, pages 8973–8979. IEEE, 2019.

[21] W. Chen, Z. Fu, D. Yang, and J. Deng. Single-image depth perception in the wild. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 730–738. Curran Associates, Inc., 2016.

[22] W. Chen, D. Xiang, and J. Deng. Surface normals in the wild. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017.

[23] W. Chen, H. Wang, Y. Li, H. Su, Z. Wang, C. Tu, D. Lischinski, D. Cohen-Or, and B. Chen. Synthesizing training images for boosting human 3d pose estimation. In *3D Vision (3DV)*, 2016.

[24] X. Chen, X. Chen, and Z.-J. Zha. Structure-aware residual pyramid network for monocular depth estimation. In *International Joint Conferences on Artificial Intelligence*, 2019.

[25] Y. Chen, W. Yu, and T. Pock. On learning optimized reaction diffusion processes for effective image restoration. In *ICCV*, 2015.

[26] Z. Chen and H. Zhang. Learning implicit fields for generative shape modeling. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[27] S. Choi, Q.-Y. Zhou, S. Miller, and V. Koltun. A large dataset of object scans. *arXiv:1602.02481*, 2016.

[28] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese. 3d-r2n2: a unified approach for single and multi-view 3d object reconstruction. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016.

[29] J. Clune and H. Lipson. Evolving 3d objects with a generative encoding inspired by developmental biology. *SIGEVOlution*, 5(4):2–12, Nov. 2011.

[30] F. Cole, P. Isola, W. T. Freeman, F. Durand, and E. H. Adelson. Shapecollage: occlusion-aware, example-based shape interpretation. In *Computer Vision–ECCV 2012*, pages 665–678. Springer, 2012.

[31] E. Conti, V. Madhavan, F. Petroski Such, J. Lehman, K. Stanley, and J. Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 5027–5038. Curran Associates, Inc., 2018.

[32] A. Criminisi and A. Zisserman. Shape from texture: homogeneity revisited. In *BMVC*, pages 1–10, 2000.

[33] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner. Scannet: richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[34] H. Dai, N. Pears, W. A. P. Smith, and C. Duncan. A 3d morphable model of craniofacial shape and texture variation. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017.

[35] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: a large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[36] T. Du, J. P. Inala, Y. Pu, A. Spielberg, A. Schulz, D. Rus, A. Solar-Lezama, and W. Matusik. Inversecsg: automatic conversion of 3d models to csg trees. *ACM Trans. Graph.*, 37(6), Dec. 2018.

[37] A. Ecker and A. D. Jepson. Polynomial shape from shading. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 145–152, June 2010.

[38] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pages 2650–2658. IEEE Computer Society, 2015.

[39] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2366–2374. Curran Associates, Inc., 2014.

[40] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. Diversity is all you need: learning skills without a reward function. In *International Conference on Learning Representations*, 2019.

[41] C. Finn, K. Xu, and S. Levine. Probabilistic model-agnostic meta-learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 9516–9527. Curran Associates, Inc., 2018.

[42] A. D. Flaxman, A. T. Kalai, A. T. Kalai, and H. B. McMahan. Online convex optimization in the bandit setting: gradient descent without a gradient. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 385–394, Vancouver, British Columbia. Society for Industrial and Applied Mathematics, 2005.

[43] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice (2Nd Ed.)* Addison-Wesley Longman Publishing Co., Inc., 1990. Chapter 12.7.

[44] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao. Deep ordinal regression network for monocular depth estimation. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2002–2011, June 2018.

[45] J. Fu, J. Co-Reyes, and S. Levine. Ex2: exploration with exemplar models for deep reinforcement learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2577–2587. Curran Associates, Inc., 2017.

[46] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig. Virtual worlds as proxy for multi-object tracking analysis. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[47] E. Garces, A. Munoz, J. Lopez-Moreno, and D. Gutierrez. Intrinsic images by clustering. *Comput. Graph. Forum*, 31(4):1415–1424, June 2012.

[48] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: the kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[49] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[50] K. Genova, F. Cole, D. Vlasic, A. Sarna, W. T. Freeman, and T. Funkhouser. Learning shape templates with structured implicit functions. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct. 2019.

[51] R. Grosse, M. K. Johnson, E. H. Adelson, and W. T. Freeman. Ground truth dataset and baseline evaluations for intrinsic image algorithms. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2335–2342. IEEE, 2009.

[52] D. Ha, A. Dai, and Q. Le. Hypernetworks. In *ICLR*, 2017.

[53] Z. Hao, Y. Li, S. You, and F. Lu. Detail preserving depth estimation from a single image using attention guided networks. *2018 International Conference on 3D Vision (3DV)*:304–313, 2018.

[54] M. A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1978.

[55] J. C. Hart. Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545, 1996.

[56] D. Hoiem, A. N. Stein, A. Efros, M. Hebert, et al. Recovering occlusion boundaries from a single image. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[57] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975. second edition, 1992.

[58] B.-S. Hua, Q.-H. Pham, D. T. Nguyen, M.-K. Tran, L.-F. Yu, and S.-K. Yeung. Scenenn: a scene meshes dataset with annotations. In *International Conference on 3D Vision (3DV)*, 2016.

[59] L. Huynh, P. Nguyen-Ha, J. Matas, E. Rahtu, and J. Heikkila. Guiding monocular depth estimation using depth-attention volume, 2020. arXiv: `2004.02760 [cs.CV]`.

[60] W. Jakob. Mitsuba renderer, 2010. http://www.mitsuba-renderer.org.

[61] M. Janner, J. Wu, T. D. Kulkarni, I. Yildirim, and J. Tenenbaum. Self-supervised intrinsic image decomposition. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5936–5946. Curran Associates, Inc., 2017.

[62] A. Janoch, S. Karayev, Y. Jia, J. T. Barron, M. Fritz, K. Saenko, and T. Darrell. A category-level 3-d object dataset: putting the kinect to work. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 1168–1174, Nov. 2011.

[63] C. Jiang, S. Qi, Y. Zhu, S. Huang, J. Lin, L. Yu, D. Terzopoulos, and S. Zhu. Configurable 3d scene synthesis and 2d image rendering with per-pixel ground truth using stochastic grammars. *International Journal of Computer Vision*, 126(9):920–941, 2018.

[64] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. Lawrence Zitnick, and R. Girshick. Clevr: a diagnostic dataset for compositional language and elementary visual reasoning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[65] A. Kar, A. Prakash, M.-Y. Liu, E. Cameracci, J. Yuan, M. Rusiniak, D. Acuna, A. Torralba, and S. Fidler. Meta-sim: learning to generate synthetic datasets. In *ICCV*, 2019.

[66] H. Kato, Y. Ushiku, and T. Harada. Neural 3d mesh renderer. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[67] T. Klatzer and T. Pock. Continuous hyper-parameter learning for support vector machines. In *CVWW*, 2015.

[68] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi. Ai2-thor: an interactive 3d environment for visual ai, 2017. arXiv: `1712.05474 [cs.CV]`.

[69] A. Lacoste, H. Larochelle, M. Marchand, and F. Laviolette. Sequential model-based ensemble optimization. In *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence*, UAI'14, pages 440–448, Quebec City, Quebec, Canada. AUAI Press, 2014.

[70] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab. Deeper depth prediction with fully convolutional residual networks. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 239–248, Oct. 2016.

[71] J.-H. Lee and C.-S. Kim. Monocular depth estimation using relative depth maps. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[72] J. Lehman and K. O. Stanley. Abandoning objectives: evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011. eprint: `https://doi.org/10.1162/EVCO_a_00025`. PMID: 20868264.

[73] J. Lehman and K. O. Stanley. Evolving a diversity of virtual creatures through novelty search and local competition. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 211–218, Dublin, Ireland. Association for Computing Machinery, 2011.

[74] B. Li, C. Shen, Y. Dai, A. van den Hengel, and M. He. Depth and surface normal estimation from monocular images using regression on deep features and hierarchical crfs. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[75] K. Li and J. Malik. Learning to optimize. In *ICLR*, 2017.

[76] T.-M. Li, M. Aittala, F. Durand, and J. Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Trans. Graph.*, 37(6):222:1–222:11, Dec. 2018.

[77] Z. Li and N. Snavely. Cgintrinsics: better intrinsic image decomposition through physically-based rendering. In *The European Conference on Computer Vision (ECCV)*, Sept. 2018.

[78] Z. Li, M. Shafiei, R. Ramamoorthi, K. Sunkavalli, and M. Chandraker. Inverse rendering for complex indoor scenes: shape, spatially-varying lighting and svbrdf from a single image. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[79] Z. Li, Z. Xu, R. Ramamoorthi, K. Sunkavalli, and M. Chandraker. Learning to reconstruct shape and spatially-varying reflectance from a single image. *ACM Trans. Graph.*, 37(6), Dec. 2018.

[80] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: common objects in context. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham. Springer International Publishing, 2014.

[81] C. Liu, J. Yang, D. Ceylan, E. Yumer, and Y. Furukawa. Planenet: piece-wise planar reconstruction from a single rgb image. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[82] F. Liu, Chunhua Shen, and Guosheng Lin. Deep convolutional neural fields for depth estimation from a single image. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5162–5170, June 2015.

[83] S. Liu, S. Saito, W. Chen, and H. Li. Learning to infer implicit surfaces without 3d supervision. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8295–8306. Curran Associates, Inc., 2019.

[84] M. M. Loper and M. J. Black. Opendr: an approximate differentiable renderer. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 154–169, Cham. Springer International Publishing, 2014.

[85] W. E. Lorensen and H. E. Cline. Marching cubes: a high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pages 163–169, New York, NY, USA. ACM, 1987.

[86] G. Louppe, J. Hermans, and K. Cranmer. Adversarial variational optimization of non-differentiable simulators. In K. Chaudhuri and M. Sugiyama, editors, *Proceedings of Machine Learning Research*, volume 89 of *Proceedings of Machine Learning Research*, pages 1438–1447. PMLR, Apr. 2019.

[87] L. Lundmark. *Synthetic Meta-Learning: Learning to learn real-world tasks with synthetic data*. Master's thesis, KTH, School of Electrical Engineering and Computer Science (EECS), 2019, page 64.

[88] W.-C. Ma, H. Chu, B. Zhou, R. Urtasun, and A. Torralba. Single image intrinsic decomposition without a single intrinsic image. In V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, editors, *Computer Vision – ECCV 2018*, pages 211–229, Cham. Springer International Publishing, 2018.

[89] D. Maclaurin, D. Duvenaud, and R. Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122, 2015.

[90] H. Mania, A. Guy, and B. Recht. Simple random search of static linear policies is competitive for reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 1800–1809. Curran Associates, Inc., 2018.

[91] P. Martinez-Gonzalez, S. Oprea, A. Garcia-Garcia, A. Jover-Alvarez, S. Orts-Escolano, and J. Garcia-Rodriguez. UnrealROX: an extremely photorealistic virtual reality environment for robotics simulations and synthetic data generation. *ArXiv e-prints*, 2018. eprint: `1810.06936`.

[92] F. Massa, B. Russell, and M. Aubry. Deep exemplar 2d-3d detection by adapting from real to rendered views. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[93] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[94] N. Mayer, E. Ilg, P. Fischer, C. Hazirbas, D. Cremers, A. Dosovitskiy, and T. Brox. What makes good synthetic training data for learning disparity and optical flow estimation? *Int. J. Comput. Vision*, 126(9):942–960, Sept. 2018.

[95] J. McCormac, A. Handa, S. Leutenegger, and A. J. Davison. Scenenet rgb-d: can 5m synthetic images beat generic imagenet pre-training on indoor segmentation? In *The IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017.

[96] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. Occupancy networks: learning 3d reconstruction in function space. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[97] S. Mohseni, M. Pitale, J. Yadawa, and Z. Wang. Self-supervised learning for generalizable out-of-distribution detection. In *AAAI 2020*, 2020.

[98] J. Mouret and S. Doncieux. Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity. In *2009 IEEE Congress on Evolutionary Computation*, pages 1161–1168, 2009.

[99] T. Munkhdalai and H. Yu. Meta networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 2554–2563. JMLR.org, 2017.

[100] A. V. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine. Visual reinforcement learning with imagined goals. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 9191–9200. Curran Associates, Inc., 2018.

[101] Y. Nesterov and V. Spokoiny. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, Apr. 2017.

[102] A. Newell, K. Yang, and J. Deng. Stacked hourglass networks for human pose estimation. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VIII*, volume 9912 of *Lecture Notes in Computer Science*, pages 483–499. Springer, 2016.

[103] A. M. Nguyen, J. Yosinski, and J. Clune. Innovation engines: automated creativity and improved stochastic optimization via deep learning. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, pages 959–966, Madrid, Spain. Association for Computing Machinery, 2015.

[104] P. Oudeyer, F. Kaplan, and V. V. Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286, 2007.

[105] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. Deepsdf: learning continuous signed distance functions for shape representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[106] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, 2017.

[107] P. Paysan, R. Knothe, B. Amberg, S. Romdhani, and T. Vetter. A 3d face model for pose and illumination invariant face recognition. In S. Tubaro and J. Dugelay, editors, *Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance, AVSS 2009, 2-4 September 2009, Genova, Italy*, pages 296–301. IEEE Computer Society, 2009.

[108] P. Perera, R. Nallapati, and B. Xiang. Ocgan: one-class novelty detection using gans with constrained latent representations. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[109] S. Pidhorskyi, R. Almohsen, and G. Doretto. Generative probabilistic novelty detection with adversarial autoencoders. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6822–6833. Curran Associates, Inc., 2018.

[110] M. A. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko. A review of novelty detection. *Signal Processing*, 99:215–249, 2014.

[111] X. Puig, K. Ra, M. Boben, J. Li, T. Wang, S. Fidler, and A. Torralba. Virtualhome: simulating household activities via programs. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[112] S. Qi, Y. Zhu, S. Huang, C. Jiang, and S.-C. Zhu. Human-centric indoor scene synthesis using stochastic grammar. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[113] X. Qi, R. Liao, Z. Liu, R. Urtasun, and J. Jia. Geonet: geometric neural network for joint depth and surface normal estimation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[114] W. Qiu, F. Zhong, Y. Zhang, S. Qiao, Z. Xiao, T. S. Kim, Y. Wang, and A. Yuille. Unrealcv: virtual worlds for computer vision. *ACM Multimedia Open Source Software Competition*, 2017.

[115] M. Ravanbakhsh, M. Nabi, E. Sangineto, L. Marcenaro, C. Regazzoni, and N. Sebe. Abnormal event detection in videos using generative adversarial nets. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 1577–1581, 2017.

[116] H. Ren, M. El-khamy, and J. Lee. Deep robust single image depth estimation neural network using scene understanding. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019.

[117] A. Ricci. A constructive geometry for computer graphics. *The Computer Journal*, 16(2):157–160, 1973.

[118] S. R. Richter and S. Roth. Discriminative shape from shading in uncalibrated illumination. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1128–1136, June 2015.

[119] S. R. Richter, Z. Hayder, and V. Koltun. Playing for benchmarks. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017.

[120] S. R. Richter, V. Vineet, S. Roth, and V. Koltun. Playing for data: ground truth from computer games. In *Computer Vision – ECCV 2016*, pages 102–118. Springer International Publishing, 2016.

[121] C. Roberto de Souza, A. Gaidon, Y. Cabon, and A. Manuel Lopez. Procedural generation of videos to train deep action recognition networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[122] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. Lopez. The SYNTHIA Dataset: a large collection of synthetic images for semantic segmentation of urban scenes. In *CVPR*, 2016.

[123] N. Ruiz, S. Schulter, and M. Chandraker. Learning to simulate. In *International Conference on Learning Representations*, 2019.

[124] R. M. Ryan and E. L. Deci. Intrinsic and extrinsic motivations: classic definitions and new directions. *Contemporary Educational Psychology*, 25(1):54–67, 2000.

[125] M. Sabokrou, M. Fathy, and M. Hoseini. Video anomaly detection and localisation based on the sparsity and reconstruction error of auto-encoder. *Electronics Letters*, 52(13):1122–1124, 2016.

[126] M. Sabokrou, M. Khalooei, M. Fathy, and E. Adeli. Adversarially learned one-class classifier for novelty detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[127] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution strategies as a scalable alternative to reinforcement learning, 2017. arXiv: `1703.03864 [stat.ML]`.

[128] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. Pixelcnn++: a pixelcnn implementation with discretized logistic mixture likelihood and other modifications. In *ICLR*, 2017.

[129] A. Saxena, M. Sun, and A. Y. Ng. Make3d: learning 3d scene structure from a single still image. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(5):824–840, 2009.

[130] J. Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247, 2010.

[131] J. Schmidhuber. Curious model-building control systems. In *In Proc. International Joint Conference on Neural Networks, Singapore*, pages 1458–1463. IEEE, 1991.

[132] J. Schmidhuber. Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. *Connection Science*, 18(2):173–187, 2006. eprint: `https://doi.org/10.1080/09540090600768658`.

[133] M. Schwarz and S. Behnke. Stillleben: realistic scene synthesis for deep learning in robotics. In *ICRA*, 2020.

[134] S. Sengupta, J. Gu, K. Kim, G. Liu, D. W. Jacobs, and J. Kautz. Neural inverse rendering of an indoor scene from a single image. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct. 2019.

[135] S. Sengupta, A. Kanazawa, C. D. Castillo, and D. W. Jacobs. Sfsnet: learning shape, reflectance and illuminance of faces 'in the wild'. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[136] G. Sharma, R. Goyal, D. Liu, E. Kalogerakis, and S. Maji. Csgnet: neural shape parser for constructive solid geometry. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[137] J. Shen, X. Yang, Y. Jia, and X. Li. Intrinsic images using optimization. In *CVPR 2011*, pages 3481–3487, June 2011.

[138] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from rgbd images. In *Computer Vision – ECCV 2012*, pages 746–760. Springer Berlin Heidelberg, 2012.

[139] S. Song, S. P. Lichtenberg, and J. Xiao. Sun rgb-d: a rgb-d scene understanding benchmark suite. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 567–576, 2015.

[140] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser. Semantic scene completion from a single depth image. *Proceedings of 29th IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[141] X. Song, W. Gao, Y. Yang, K. Choromanski, A. Pacchiano, and Y. Tang. Es-maml: simple hessian-free meta learning. In *ICLR*, 2020.

[142] K. O. Stanley. Compositional pattern producing networks: a novel abstraction of development. *Genetic programming and evolvable machines*, 8(2):131–162, 2007.

[143] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evol. Comput.*, 10(2):99–127, June 2002.

[144] H. Su, C. R. Qi, Y. Li, and L. J. Guibas. Render for cnn: viewpoint estimation in images using cnns trained with rendered 3d model views. In *The IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015.

[145] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune. Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning, 2017. arXiv: `1712.06567 [cs.NE]`.

[146] S. Sukhbaatar, Z. Lin, I. Kostrikov, G. Synnaeve, A. Szlam, and R. Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. In *International Conference on Learning Representations*, 2018.

[147] X. Sun, J. Wu, X. Zhang, Z. Zhang, C. Zhang, T. Xue, J. B. Tenenbaum, and W. T. Freeman. Pix3d: dataset and methods for single-image 3d shape modeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[148] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Multi-view 3d models from single images with a convolutional network. In *European Conference on Computer Vision (ECCV)*, 2016.

[149] Y. Tian, A. Luo, X. Sun, K. Ellis, W. T. Freeman, J. B. Tenenbaum, and J. Wu. Learning to infer and execute 3d shape programs. In *International Conference on Learning Representations*, 2019.

[150] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[151] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, Sept. 2017.

[152] J. Tremblay, T. To, and S. Birchfield. Falling things: a synthetic dataset for 3d object detection and pose estimation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018.

[153] A. Van Den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1747–1756, New York, NY, USA. JMLR.org, 2016.

[154] A. van den Oord, N. Kalchbrenner, L. Espeholt, k. kavukcuoglu koray, O. Vinyals, and A. Graves. Conditional image generation with pixelcnn decoders. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4790–4798. Curran Associates, Inc., 2016.

[155] G. Varol, J. Romero, X. Martin, N. Mahmood, M. J. Black, I. Laptev, and C. Schmid. Learning from synthetic humans. In *CVPR*, 2017.

[156] V. Veeravasarapu, C. Rothkopf, and R. Visvanathan. Adversarially tuned scene generation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[157] P. Wang, X. Shen, Z. Lin, S. Cohen, B. Price, and A. L. Yuille. Towards unified depth and semantic prediction from a single image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2800–2809, 2015.

[158] R. Wang, J. Lehman, J. Clune, and K. O. Stanley. Paired open-ended trailblazer (poet): endlessly generating increasingly complex and diverse learning environments and their solutions, 2019. arXiv: `1901.01753 [cs.NE]`.

[159] X. Wang, D. Fouhey, and A. Gupta. Designing deep networks for surface normal estimation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[160] X. Wang, Y. Du, S. Lin, P. Cui, Y. Shen, and Y. Yang. Advae: a self-adversarial variational autoencoder with gaussian anomaly prior knowledge for anomaly detection. *Knowledge-Based Systems*, 190:105187, 2020.

[161] B. G. Woolley and K. O. Stanley. On the deleterious effects of a priori objectives on evolution and representation. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 957–964, Dublin, Ireland. Association for Computing Machinery, 2011.

[162] J. Wu, J. B. Tenenbaum, and P. Kohli. Neural scene de-rendering. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[163] Y. Wu, Y. Wu, G. Gkioxari, and Y. Tian. Building generalizable agents with a realistic and rich 3d environment. In *ICLR (Workshop)*. OpenReview.net, 2018.

[164] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese. Gibson env: real-world perception for embodied agents. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[165] Y. Xia, X. Cao, F. Wen, G. Hua, and J. Sun. Learning discriminative reconstructions for unsupervised outlier removal. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1511–1519, Dec. 2015.

[166] Y. Xiang, W. Kim, W. Chen, J. Ji, C. Choy, H. Su, R. Mottaghi, L. Guibas, and S. Savarese. Objectnet3d: a large scale database for 3d object recognition. In *European Conference Computer Vision (ECCV)*, 2016.

[167] Y. Xiong, A. Chakrabarti, R. Basri, S. J. Gortler, D. W. Jacobs, and T. Zickler. From shading to local shape. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 37(1):67–79, 2015.

[168] Q. Xu, W. Wang, D. Ceylan, R. Mech, and U. Neumann. Disn: deep implicit surface network for high-quality single-view 3d reconstruction. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 492–502. Curran Associates, Inc., 2019.

[169] D. Yang and J. Deng. Learning to generate synthetic 3d training data through hybrid gradient. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[170] D. Yang and J. Deng. Novelty-guided evolution for generating synthetic 3d training data. Manuscript submitted for publication, 2020.

[171] D. Yang and J. Deng. Shape from shading through shape evolution. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[172] Y.-T. Yeh, L. Yang, M. Watson, N. D. Goodman, and P. Hanrahan. Synthesizing open worlds with constraints using locally annealed reversible jump mcmc. *ACM Trans. Graph.*, 31(4):56:1–56:11, July 2012.

[173] Y. Yu and W. A. P. Smith. Inverserendernet: learning single image inverse rendering. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[174] R. Zhang, P.-S. Tsai, J. E. Cryer, and M. Shah. Shape from shading: a survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(8):690–706, Aug. 1999.

[175] Y. Zhang, S. Song, E. Yumer, M. Savva, J.-Y. Lee, H. Jin, and T. Funkhouser. Physically-based rendering for indoor scene understanding using convolutional neural networks. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[176] Z. Zhang, Z. Cui, C. Xu, Y. Yan, N. Sebe, and J. Yang. Pattern-affinitive propagation across depth, surface normal and semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[177] Q. Zhao, P. Tan, Q. Dai, L. Shen, E. Wu, and S. Lin. A closed-form solution to retinex with nonlocal texture constraints. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1437–1444, 2012.

[178] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. Torr. Conditional random fields as recurrent neural networks. In *ICCV*, 2015.

[179] Zhirong Wu, S. Song, A. Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and J. Xiao. 3d shapenets: a deep representation for volumetric shapes. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, 2015.

[180] X. Zhu, Z. Lei, X. Liu, H. Shi, and S. Z. Li. Face alignment across large poses: a 3d solution. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 146–155. IEEE Computer Society, 2016.

[181] X. Zhu, Z. Lei, J. Yan, D. Yi, and S. Z. Li. High-fidelity pose and expression normalization for face recognition in the wild. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 787–796. IEEE Computer Society, 2015.

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