博士論文

Learning 3D mesh reconstruction from 2D images

(2D 画像からの 3D メッシュの再構成の学習)

加藤 大晴

Learning 3D mesh reconstruction from 2D images

Hiroharu Kato

Department of Mechano Informatics
Graduate School of Information Science and Technology
The University of Tokyo

This thesis is submitted for the degree of
Doctor of Philosophy

August 2020

Acknowledgements

This research was conducted under the supervision of Prof. Tatsuya Harada at Department of Mechano-Informatics, Graduate School of Information Science and Technology, the University of Tokyo.

Firstly, I would like to express the most profound appreciation to my supervisor Prof. Tatsuya Harada. Since I was an undergraduate student in 2010, he has always supported me with encouraging, constructive, and insightful conversations. His advices often led me to new ideas. Beyond the technical aspects, he has strongly influenced my attitude and approach to research. Without his guidance and constant feedback, this PhD would not have been achievable.

I would like to extend my sincere thanks to Prof. Yasuo Kuniyoshi, Prof. Takeo Igarashi, Prof. Kei Okada, and Assoc. Prof. Takuji Narumi for their constructive comments and suggestions. They are sub-chief examiners of this doctoral thesis and their insightful feedback have further refined this thesis.

My comfortable life in the laboratory was supported by the help of Assoc. Prof. Tomoyuki Takahata, Lecturer Yoshitaka Ushiku, Lecturer Yusuke Mukuta, Asst. Prof. Mamoru Nakamura, Dr. Tejero de Pablos Antonio, Dr. Yusuke Kurose, Dr. Hiroaki Yamane, Ms. Aoi Kaneko, and Ms. Miyuki Kajisa. They have never spared any efforts for improving research environment.

I spent a wonderful life in the lab, especially with Yusuke Mori, Dr. Takuhiro Kaneko, and Naoyuki Gunji, who were in the same grade at the undergraduate school. Conversations with them have helped me get through the pleasant but sometimes tough days.

Discussion with my colleagues were always inspirational. Unfortunately, I don't have space to list the names of all the people who have influenced me. Here, I especially would like to thank Dr. Yoshitaka Ushiku, Dr. Yusuke Mukuta, Dr. Antonio Tejero de Pablos, Dr. Hiroaki Yamane, Dr. Takuhiro Kaneko, Dr. Atsushi Kanehira Yusuke Mori, Naoyuki Gunji, Kuniaki Saito, Katsunori Ohnishi, Keisuke Fukuta, Kosuke Arase, Kentaro Takemoto, Shunya Wakasugi, and Daisuke Kasuga.

Finally, I am extremely grateful to my partner Hitomi for her support, encouragement, and dedication to me.

Abstract

Understanding the 3D structure of an object at a glance, called *single-view 3D object reconstruction* in computer vision, is a useful ability for machines. Because this is an ill-posed problem, it is solved by learning 3D reconstruction from the data, in contrast to traditional 3D reconstruction methods that employ geometric estimation. The question to consider is what kind of data to learn from. Because of the large number of object categories in the world, 3D object reconstruction has to be learned at a low cost. This thesis explores methods for learning 3D object reconstruction using 2D images instead of costly 3D shapes. First, we develop a novel rasterization method to incorporate mesh rendering into neural networks. We also present a detailed comparison of differentiable rasterization methods. Second, to learn single-view 3D object reconstruction from multi-view images with camera parameter and foreground mask annotations, we develop a mesh reconstruction method using the renderer. Third, to use single-view images rather than multi-view images, we propose to address the shape ambiguity inherent in single-view images. Fourth, to use images without annotations, we propose learning reconstruction while separating the foreground and background. By developing these techniques, we significantly reduce the cost of training data required for 3D object reconstruction. We are the first to realize learning 3D mesh reconstruction from images while existing works use voxels, improve the accuracy of reconstruction learned from single-view images by learning priors of views, and achieve learning 3D object reconstruction from only images without additional supervision.

Table of contents

Chapter 1

Introduction

1.1 Background and objective

We humans can grasp the three-dimensional (3D) structure of an object, including its invisible parts, at a glance. This ability is essential to many activities, such as grasping objects, avoiding obstacles, and creating 3D models using computer-aided design (CAD) software. This skill is called *single-view 3D object reconstruction* in computer vision. It has many practical applications, such as robot grasping, 3D scene understanding for robots, object placement in augmented reality, and CAD.

Recovering the 3D structure of an object from only a 2D image is an inherently ill-posed problem. A 2D image does not provide either depth of visible surfaces or any information of occluded structures. However, humans can solve this task easily[1]. Why is this possible? Geometric reasoning is not the primary mechanism because of the ill-posedness. Instead, we leverage knowledge about the shapes of objects that we have learned from experience. We know the possible shapes of certain categories of objects and estimate the shapes in the observed images by applying them. Similarly, single-view 3D object reconstruction requires learning possible shapes of objects in advance from data.

The most straightforward approach to single-view 3D reconstruction is to employ supervised learning using 2D images and their corresponding ground truth 3D models [9, 13, 21, 34, 39, 85, 101] as illustrated in Figure 1.1. To this purpose, most existing works use ShapeNet [7], a large-scale 3D CAD dataset. Recent technical advancement has realized to generate a high-quality 3D shape from a single image in the object categories in ShapeNet [9, 13, 19, 107, 126]. However, this supervised learning is not scalable to the

[1]Though humans can use depth information estimated by both eyes, this ability is hardly degraded even when depth information is not available. For example, we can do single-view 3D reconstruction when one eye is closed or from a photograph where we cannot know the depth.

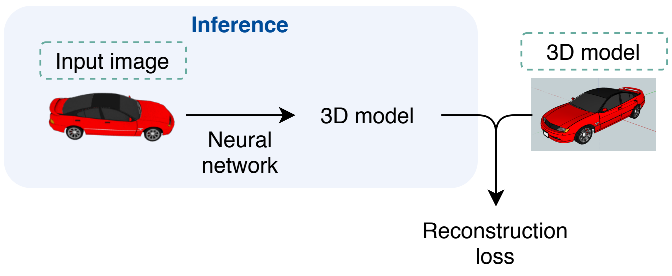


Figure 1.1 Learning single-view 3D object reconstruction using 3D model annotations. The dotted squares represent the training data. This framework is beyond the scope of this thesis.

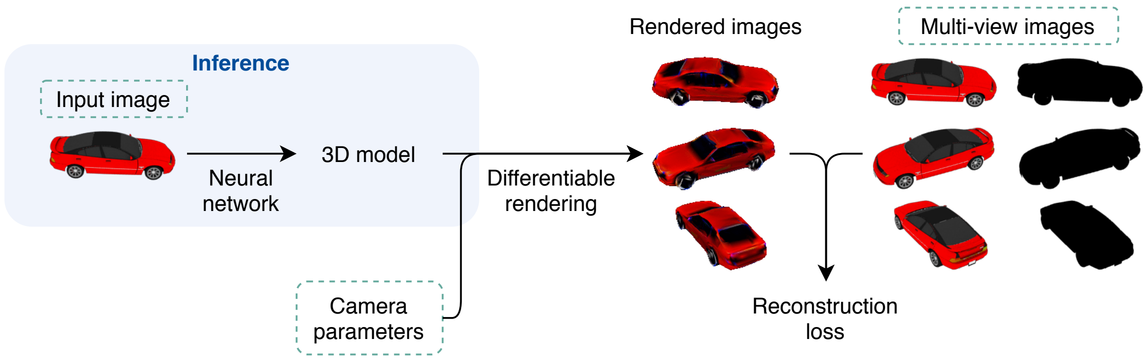


Figure 1.2 Learning single-view 3D object reconstruction using multi-view 2D images with camera parameter and silhouette annotations. The dotted squares represent the training data. This framework corresponds to Chapter 4.

number of object categories because creating 3D annotations requires extraordinary effort by professional 3D designers. Only thirteen object categories on ShapeNet have a sufficient amount of 3D models for supervised learning. Therefore, single-view 3D object reconstruction in more diverse categories is still challenging so far. Instead, we propose to leverage 2D images without 3D shape annotations for training as less costly datasets.

Learning from images is achieved by comparing ground truth images with views of an estimated 3D shape. Figures 1.2, 1.3, and 1.4 show variants of training pipelines. A pipeline is composed of the following operations.

1. Given an object image, we estimate the parameters of the underlying 3D object. We estimate only a 3D shape in the simplest case. Optionally, we infer other parameters such as camera pose and the color of object surfaces.

2. We generate an image by rendering the estimated 3D model and a given or estimated camera pose.

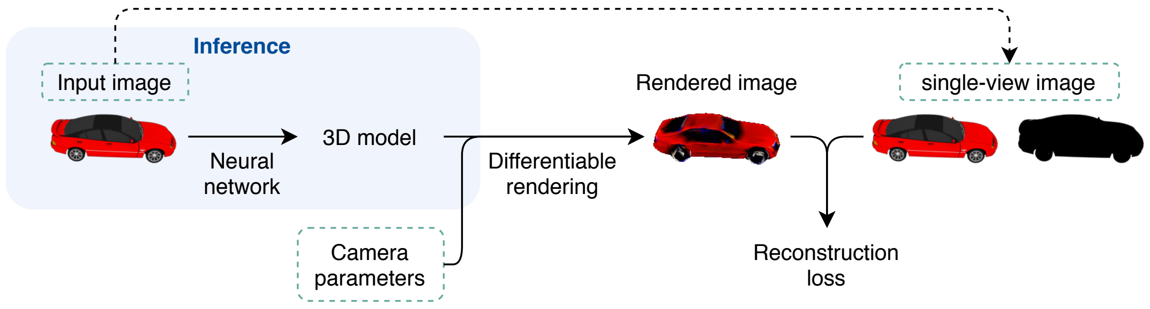


Figure 1.3 Learning single-view 3D object reconstruction using a single-view 2D image with camera parameter and silhouette annotations. The dotted squares represent the training data. The input image is reused to measure the reconstruction error. This framework corresponds to Chapter 5.

3. We compute the difference between the input image and the reconstructed image. Optionally, we use images of the same object from other viewpoints as additional supervision.

4. We update the parameters of the 3D reconstruction model by back-propagating the gradient of the loss function and gradient descent.

As explained above, there are several variants in training according to available datasets. We aim to reduce the cost of creating training datasets. As expected, when there is less information available, training becomes more difficult. In this thesis, we focus on three cases, as discussed later.

For this render-and-compare framework to work well, high-quality images must be generated through rendering. Therefore, we use polygon mesh as a 3D geometry representation. Though voxels, a 3D extension of pixels, are the most widely used format in 3D machine learning because they can be processed by convolutional neural networks (CNNs) [9, 61, 82, 86, 101, 107, 117, 121, 126], it is hard to process high-resolution voxels because of their high memory consumption. Point clouds, sets of 3D points, have difficulty handling textures and lighting because they do not have explicit surfaces. Contrary to these representations, a polygon mesh is compact and it supports textures.

In summary, we realize learning single-view 3D object reconstruction from images, where a mesh represents a 3D model. Previously, 3D shapes were used as a training signal instead of images, or voxels were used as a 3D representation instead of meshes. We address three cases illustrated in Figures 1.2, 1.3, and 1.4. We develop several novel methods that are essential for this objective.

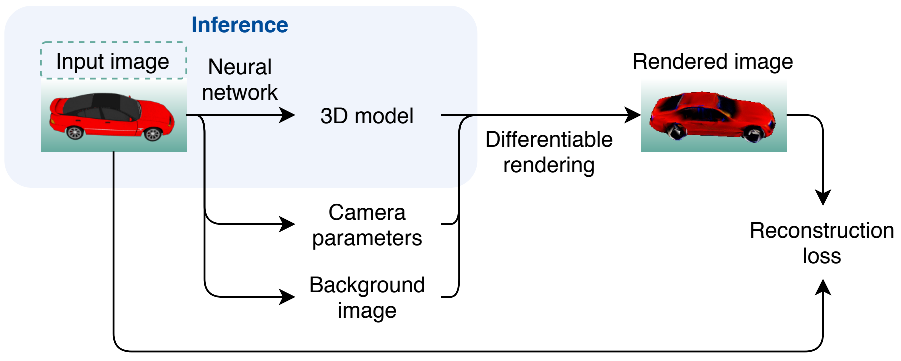


Figure 1.4 Learning single-view 3D object reconstruction using a single-view 2D image without camera parameter and silhouette annotations. The dotted squares represent the training data. The input image is reused to measure the reconstruction error. Camera parameters and a background image are estimated in addition to the 3D object because their annotations are not given. This framework corresponds to Chapter 6

1.1.1 Differentiable rendering

First, we develop a differentiable rendering module of a mesh for neural networks. As neural networks are known to be extremely powerful for image recognition [47], we use a neural network for the reconstruction module, and train it in an end-to-end manner. However, training a system with rendering as a neural network is a challenging problem. Rendering consists of projecting a mesh's vertices onto the screen coordinate system and generating an image through regular grid sampling [60]. Although the former is a differentiable operation, the latter, referred to as *rasterization*, prevents back-propagation because of its discrete nature. Therefore, we propose an approximate gradient for rasterization peculiar to neural networks, facilitating end-to-end training. Our proposed renderer can flow gradients into texture, lighting, and cameras as with object shapes. We name our renderer *Neural Renderer*.

1.1.2 Variants of training data

There are several variants in learning single-view 3D reconstruction from images depending on available training data. We address the following three cases in this thesis.

1. The case that a set of multi-view images of an object and their viewpoint and silhouette annotations are available. We can obtain this kind of data by a calibrated 3D scanning system composed of multiple RGB cameras or a turntable. Though this is technically the most straightforward case, it is costly to collect such data for many objects. Besides,

Training data	Possible data source	Architecture	Chapter
Images with 3D shape annotations	CAD	Figure 1.1	Out of scope
Multi-view images with annotations	3D scanner	Figure 1.2	Chapter 4
Single-view images with annotations	Image collection + manual annotation	Figure 1.3	Chapter 5
Single-view images without annotations	Image collection	Figure 1.4	Chapter 6

Table 1.1 Variants of training data considered in this thesis. *Annotations* are composed of silhouette and camera pose information.

there are object categories, such as airplanes, which are practically impossible to obtain images in such a system. The training architecture is shown in Figure 1.2.

2. The case that only a single image is available for an object, but its viewpoint and silhouette annotations are still available. This data can be obtained when the cost of creating 3D annotations for 2D images is not negligible, but the cost of creating silhouette and viewpoint annotations[2] is allowed. This training data can be obtained in most object categories. However, since creating annotations are still costly, this data is also not very scalable to the number of object categories. The training architecture is shown in Figure 1.3.

3. The case that an image collection of a certain object category is available, but no additional annotations are provided. This kind of data is obtained by collecting images and classifying them into object categories. Since the cost for training data creation is low, this approach easily scales to various object categories. However, learning with this supervision is very challenging. The training architecture is shown in Figure 1.4.

Table 1.1 shows the summary of these variants.

Chapter 4: Learning from multi-view images

Several prior works addressed learning single-view 3D object reconstruction from multi-view images with silhouette and viewpoint annotations using voxels as 3D representation [107, 126]. However, voxels are not well suitable for this task because of two reasons. One is that voxels cannot represent high-resolution shapes because their memory consumption is cubic to their spatial resolution. The other is that images rendered from voxels are not suited

[2]Viewpoint annotation is typically made by overlaying a 3D template object on an image or via creating keypoint annotations. In contrast to 3D shape annotation, creating viewpoint annotations does not require professional 3D modeling skills.

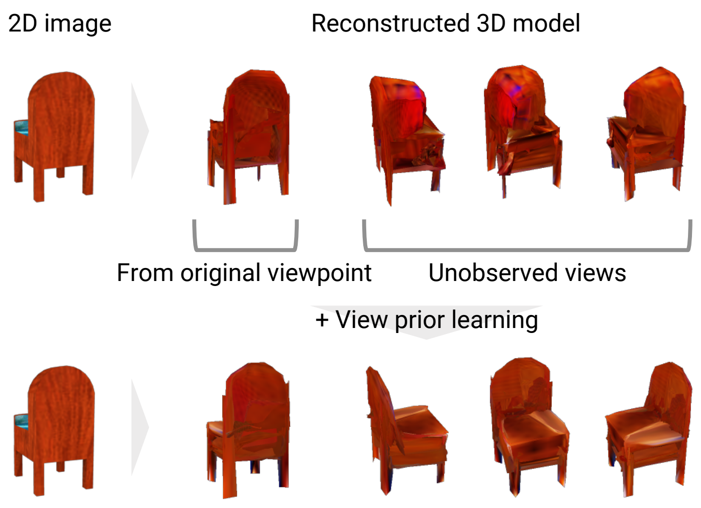


Figure 1.5 When a 3D reconstructor is trained using only a single view per object, because of ambiguity in the 3D shape of an object, it reconstructs a shape which only fits the observed view and looks incorrect from unobserved viewpoints (upper). By introducing a discriminator that learns prior knowledge of correct views, the reconstructor is able to generate a shape that is viewed as reasonable from any viewpoint (lower).

for the render-and-compare framework because they have cubic artifacts in views because of their grid representation. Instead, our differentiable renderer allows us to use a mesh in single-view 3D object reconstruction, which does not suffer from these two problems. In the experiment section, we demonstrate that the accuracy of 3D reconstruction using mesh outperforms that of a voxel-based method [126].

Chapter 5: Learning from single-view images with annotations

Because we do not know a ground truth 3D shape in view-based training, there is some ambiguity in the possible shapes. In other words, several different 3D shapes can be projected into the same 2D view, as shown in Figure 1.5. A standard way to reduce this ambiguity is to use twenty or more views per object in training. However, this is impractical in many cases in terms of feasibility and scalability. When creating a dataset by taking photos, it is difficult to take pictures from many viewpoints if an object is moving or deforming. Also, when creating a dataset using many images from the Internet, it is not always possible to collect multiple views of an object. Therefore, it is desirable that a reconstructor can be trained using a single view of an object.

Therefore, next, we focus on training a reconstructor using a single view for single-view 3D object reconstruction. In this case, the ambiguity in shapes in training is not negligible.

The upper half of Figure 1.5 shows single-view 3D reconstruction by the method proposed in Chapter 4 Although this method requires multiple views per object for training, we used a single view. As a result, though the reconstructed shape looks correct when viewed from the same viewpoint as the input image, it looks incorrect from other viewpoints because the reconstructor is unaware of unobserved views and generates shapes that only fit the observed views.

How can a reconstructor overcome shape ambiguity and correctly estimate shapes? The hint is in Figure 1.5. Humans can recognize that the three views in the upper are incorrect because we have prior knowledge of how a chair look by having seen many chairs in the past. If machines also know the correct views, they would use it to estimate shapes more accurately. We implement this idea on machines by using a discriminator and adversarial training [17]. While the unobserved views of estimated shapes do not always correct, observed views converge to the correct ones, as seen in the figure. Therefore, we train a discriminator that distinguishes the observed views from the unobserved views. This results in the discriminator obtaining knowledge regarding correct views. By training the reconstructor to fool the discriminator, reconstructed shapes from all viewpoints become indistinguishable and look reasonable from any viewpoint. The lower half of Figure 1.5 shows the results from the proposed method.

Learning prior knowledge of 3D shapes using 3D models was tackled in other publications [21, 118]. In contrast, we focus on prior of 2D views rather than 3D shapes. Because our method does not require any 3D models for training, ours can scale to various categories where 3D models are hard to obtain.

Chapter 6: Learning from single-view images without annotations

The approach introduced in the previous section requires annotations of objects' foreground mask and viewpoint annotations, which are still not easy to obtain. Therefore, next, we propose a novel method for training without these annotations and demonstrate that we can learn objects' 3D shape, pose, and texture from categorized natural images without additional supervision. Figure 1.6 shows our result on the test set of CIFAR-10 [46], a dataset for image classification. This dataset has no ground truth 3D shapes, no multiple views of the same object, no foreground masks of objects, and no viewpoint information. The success in single-view 3D object reconstruction on this challenging dataset indicates leveraging natural images for 3D reconstruction and realizing 3D object reconstruction on various object categories.

Because there is no supervision about 3D objects, we have to use images themselves as supervision. Therefore, we adopt the analysis-by-synthesis framework and train this model by

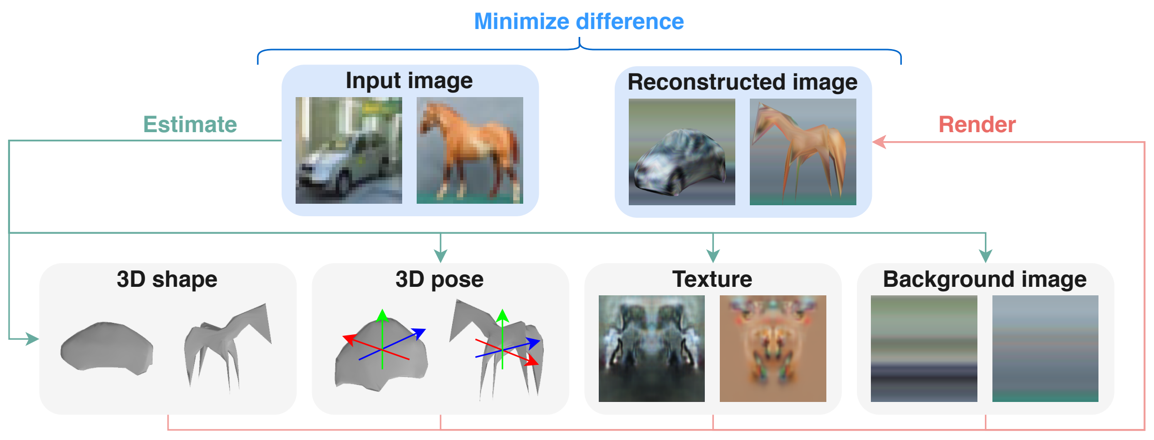


Figure 1.6 Our proposed model estimates an object's 3D shape, pose, texture, and background from an image. It requires only categorized object images for training. This figure shows the system architecture and result on CIFAR-10.

comparing an input image with a reconstructed 3D object's rendered image. The procedure is as follows. At first, we estimate 3D shape, pose, texture, and background from the input using neural networks. Secondly, we render an image of the estimated 3D scene. Finally, we compute image reconstruction error and optimize the neural networks by minimizing the error. Though this framework is simple, training a meaningful model is not straightforward because it easily fails to learn, as depicted in Figure 1.7. One example is to expand an object's shape to cover the whole image and copy the input image into the texture. Another example is to shrink an object and copy the input image into the background. Though image reconstruction is almost perfect in both cases, we know that these 3D scenes are unrealistic. The technical key point is to explicitly induct our knowledge about 3D structures into neural networks in the form of training methods, constraints, and regularization. Concretely, our main priors are (1) all shapes in the same object category are similar, and they can be made by small deformation of a category-specific common shape, (2) surfaces of objects are smooth, and (3) texture and background images are sufficiently simple. Based on these assumptions, we separate training steps into category-specific common shape generation and full training using a common shape, and carefully design constraints and regularization. We demonstrate the effectiveness of our training strategy and constraints based on these priors in experiments on CIFAR-10 and PASCAL 3D+ [123] datasets.

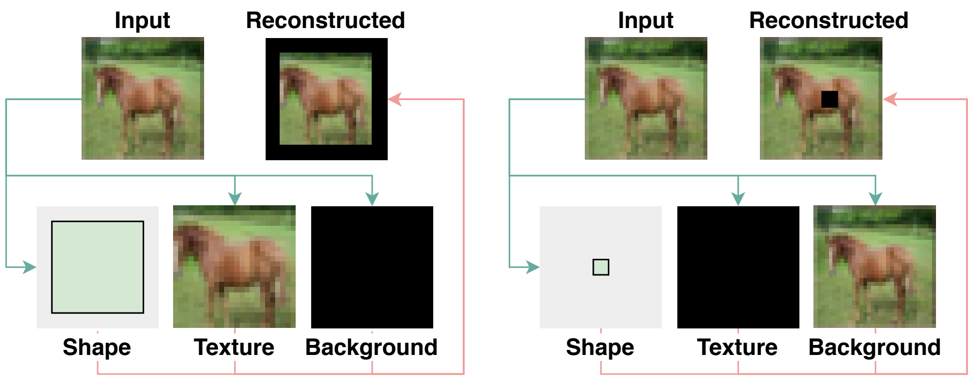


Figure 1.7 Examples of trivial and inferior solutions. We know these 3D scenes are unrealistic. However, neural networks cannot know it by self-supervision. Therefore, we induct several kinds of structural knowledge of 3D scenes into our model.

1.2 Summary of contributions

The major contributions can be summarized as follows.

- Chapter 3. We propose the first approximated gradient of mesh rendering for neural networks. We compare ours and other differentiable rasterizers [8, 54, 57] in experiments and clarify their performances and features. Also, we make the code of the renderer publicly available.

- Chapter 4. We perform learning 3D mesh reconstruction from a single image without 3D supervision. Experiments demonstrate the practicality of our renderer and the advantage of mesh reconstruction over voxel reconstruction.

- Chapter 5. We propose a method to predict shapes that look reasonable from any viewpoint by learning prior knowledge of object views. We also conduct experiments on both synthetic and natural image datasets and observe a significant performance improvement for both datasets, especially when only a single image per object is provided in training.

- Chapter 6. Ours is the first study to train single-view 3D object reconstruction, pose estimation, and texture estimation using only categorized natural images. We demonstrate that introducing training of a category-specific common shape, and constraints about 3D scenes are essential.

1.3 Structure of the thesis

In this thesis, we aim to develop methods to learn single-view 3D mesh reconstruction from images. In Chapter 1, we describe the background and objective. Specifically, we state the necessity to develop a differentiable rendering module for neural networks, and the three essential training data variants and their technical challenges. In Chapter 2, we have more comprehensive discussion on existing works and clarify the relationship between them and ours. In Chapter 3, we propose a rendering module, which is requisite for learning single-view mesh reconstruction from images in an end-to-end manner, and compare it and related methods in experiments to reveal their features. In Chapter 4, 5, 6, we study learning single-view 3D reconstruction in the three settings shown in Table 1.1. Finally, in Chapter 7, we conclude this thesis and remark future directions.

Chapter 2

Related work

This chapter presents related work on differentiable rendering, an important technical element of this thesis, and learning single-view 3D object reconstruction.

2.1 3D reconstruction in computer vision

There is a vast amount of 3D reconstruction works in computer vision. One well-known task is to reconstruct 3D geometry from multiple views of a scene. While multi-view stereo assumes that camera parameters are given, structure-from-motion [109] optimizes both geometry and camera parameters. Another research direction is to estimate the depth of surfaces using additional cues such as shading [28], texture [113], and camera focus [70, 78]. However, all of these approaches cannot predict an object's whole shape because its part may not be observed. Learning-based 3D reconstruction handles whole shape reconstruction, and it becomes getting popular since the appearance of large-scale 3D CAD model dataset [7] and the success of deep neural networks in computer vision [47]. This thesis also follows this direction.

2.2 Differentiable rendering

A renderer takes geometry, material, light, and camera parameters and outputs an image. *Differentiable renderer* indicates a renderer such that the gradient of an output image with respect to input parameters is defined. Differentiable rendering is used to optimize 3D scene parameters or neural networks that produce them by minimizing a loss function defined on rendered 2D images. A gradient needs not to be a mathematically valid one. Since the objective is optimization, rather than mathematical correctness, practical effectiveness for optimization is more critical. Our proposal is also an inaccurate but useful pseudo gradient.

There are two major rendering methods for mesh. One is real-time rendering using rasterization, and the other is photorealistic rendering using path tracing. Since these two differ significantly, their differentiable rendering methods have been developed individually.

In real-time rendering, an image is generated by converting vertices and lights from the world space into the camera and screen space (projection), selecting a triangle that corresponds to a pixel (rasterization), and computing the color of a pixel using selected triangle, material, and light parameters (shading). Differentiation of projection is trivial because a matrix multiplication does it. Differentiation of shading is also easy because most popular shading models such as Lambert and Phong are trivially differentiable. However, rasterization sometimes causes a optimization problem because a pixel can affect only the selected triangle and cannot affect neighboring triangles that potentially influence the pixel color. OpenDR [57], the first general differentiable rendering method, was the only paper before the publication of our renderer [41]. It points out that the analytic gradient of rasterization in rendering is not useful for shape optimization. And, it proposes an approximate gradient so that a gradient of a pixel effect to triangles of neighboring pixels. However, because it does not use the gradient of a loss function with respect to pixels in back-propagation, its gradient would not be very useful for optimization. We describe our solution to this problem in the next chapter. Another early approach to incorporating renderings into neural networks is to estimate the gradient of a black-box non-differentiable renderer [84]. However, its gradient is noisy and not very good for optimization compared to differentiable rendering. After the publication of our work, several differentiable rendering methods have been proposed. Liu *et al.* [54] propose to approximate the forward pass of rasterization by blurring triangles instead of approximating gradients so that multiple triangles are assigned to one pixel. Chen *et al.* [8] classify pixels into foreground and background, and propose to use a method similar to Liu *et al.* [54] for the background pixels, and analytical gradients using barycentric coordinates for the foreground pixels.

In the path-tracing of photorealistic rendering, the main difficulty of differentiable rendering is in a non-differentiability in the rendering equation. Though most reflection functions are differentiable, because the integral of incident lights is evaluated using Monte Carlo estimation, auto differentiation frameworks cannot evaluate the effect of geometry change to pixel colors. To this issue, Li *et al.* [53] propose to sample points on all edges for derivative computation. Because edge finding and sampling are slow, Loubet *et al.* [58] propose a biased but efficient gradient computation by reparameterization and integrate it into Mitsuba

3D representation	Method
Mesh (local illumination)	OpenDR [57] NMR (ours) [41] SoftRas [54] DIB-R [8]
Mesh (global illumination)	Redner [53] Mitsuba 2 [58, 73] PSDR [135]
Voxel	PTN [126] DRC [107] Henzler *et al.* [27]
Point cloud	Insafutdinov *et al.* [31] DSS [130]
Neural implicit function	Liu *et al.* [55] Niemeyer *et al.* [72] Mildenhall *et al.* [63]

Table 2.1 Differentiable rendering methods.

2 renderer [73]. Instead, Zhang *et al.* [135] propose an efficient and unbiased method by differentiating path integral instead of the original rendering equation.

There are differentiable renderers for other 3D representations too. Table 2.1 shows representative methods in this field. Voxels, which are 3D extensions of pixels, are widely used with neural networks for classification [61, 82, 86, 117, 121], 3D reconstruction and generation [9, 26, 101, 107, 117, 126] because CNN easily processes them. Because the memory efficiency of voxels is not very good, some recent works have incorporated more efficient representations [86, 101, 112]. Compared with differentiable rendering of mesh, differentiable rendering of voxels is simply achieved by sampling points in 3D space from voxels using trilinear sampling [107, 126, 26]. Point cloud represents a 3D scene by a set of points. It has been used for both recognition [44, 80, 81] and reconstruction [13, 31, 130]. Differentiable rendering of point cloud is achieved by projecting it to 2D screen assuming that the points have a size [31, 130]. Recently, representing a 3D shape implicitly in the weights of neural networks become getting popular [8, 62, 76]. Liu *et al.* [55], Niemeyer *et al.* [72], and Mildenhall *et al.* [63] concurrently propose differentiable rendering methods for this neural implicit representations.

Physics-based vision, such as shape-from-shading [28] and photometric stereo [115], also models a generative process of an image as with differentiable rendering. Their advantage is to design optimization methods and additional constraints jointly with image generation models, and their disadvantage is difficulty in handling complex light and illumination models like global illumination. In contrast, though differentiable photorealistic rendering can manage sophisticated image generation processes, optimization over it may not be easy. Our work is not in physics-based vision because we do not use light models for geometry estimation. However, a combination of differentiable rendering and shading is demonstrated to be effective in another work [25].

Please refer to our recent survey [42] for a more comprehensive review.

2.3 Single-view 3D object reconstruction

We classify single-view 3D object reconstruction works by supervision for 3D geometry. The simplest case is *3D supervision* where pairs of a 2D image and its corresponding 3D shape is available for standard supervised training. We call a weakly-supervised setting where multi-view images of an object are provided instead of its 3D shape *multi-view training*. Finally, we consider *single-view training*, a case where only a single view per object is given for training.

2.3.1 3D supervision

When plenty of 3D object models are available, using 3D shapes as training signals, as illustrated in Figure 1.1, would be the most straightforward way. The advent of large-scale 3D shape datasets, such as ShapeNet [7], has popularized this approach. A significant advantage of 3D supervision is that there is no ambiguity in 3D object shape in an object image. One research direction is handling irregular 3D representations, such as high-resolution voxels [23, 101], point clouds [13], meshes [19, 111], and neural implicit representations [8, 62, 76]. Another path is generalization to novel object categories that are not observed in training [96, 102, 137].

2.3.2 Multi-view supervision

3D geometry is understandable from multi-view images without exact 3D data, and we can use them as supervision with silhouette and camera parameter annotations, as illustrated in Figure 1.2. We need a differentiable 3D-to-2D projection module to use 2D views as supervision. To this purpose, several differentiable projection modules have been developed as shown in Table 2.1. Compared with 3D supervision, creating a multi-view dataset is less costly. However, collecting multiple views is still expensive because it requires a specialized 3D capture system.

We address learning single-view 3D object reconstruction with multi-view supervision in Chapter 4. The closest work to ours is Yan et al. [126] because it is the first multi-view training work. Though they use voxels, we demonstrate that using mesh is preferable.

2.3.3 Single-view supervision

Collecting a single-view image per object is not hard and less costly. However, since learning only from images is very challenging, additional supervision is typically required so far. A typical case is to use silhouette and camera parameter annotations, as illustrated in Figure 1.3.

Supervision	Chapter 5	[84]	[108]	[25]	[26]	[140]†	[71]‡	Chapter 6
Natural images	✓				✓	✓	✓	✓
No viewpoint annotation		✓	✓	✓	✓	✓	✓	✓
No silhouette annotation						✓	✓	✓

Table 2.2 Works that aim supervision reduction in single-view training. *No silhouette annotation* includes works that use images without backgrounds. †Zuffi *et al.* [140] leverage a simulator. ‡Nguyen-Phuoc *et al.* [71] cannot represent 3D shapes explicitly.

	Work	Discrimination
(a)	[34, 127]	Positive: Predicted 3D shapes
		Negative: Their corresponding ground truth 3D shapes
(b)	[21, 118]	Positive: Predicted 3D shapes
		Negative: 3D shape collections
(c)	Ours	Positive: Views of predicted 3D shapes from observed viewpoints
		Negative: Views of predicted 3D shapes from random viewpoints
(d)	-	Positive: Views of predicted 3D shapes
		Negative: Views in a training dataset

Table 2.3 Summary of discriminators in learning-based 3D reconstruction. Discriminator (d) is described in Section 5.2.2.

There is a variety of additional supervision in single-view training. Kar *et al.* [38] propose a method to recover 3D shapes and viewpoints from natural images with annotations of silhouettes and keypoints of objects. Kanazawa*et al.* [36] translate this framework into neural networks and incorporate texture prediction. Tulsiani *et al.* [107] applie their multi-view training method that requires silhouette and viewpoint supervision onto a single-view dataset, and later they relax this dataset requirement by integrating pose prediction [108]. Rezende *et al.* [84] train 3D reconstruction from images. However, their dataset is composed of very simple primitives without backgrounds. Henzler *et al.* [26] train 3D object reconstruction from natural images with silhouette annotations. Nguyen-Phuoc *et al.* [71] train neural 3D representation from natural images. Zuffi *et al.* [140] leverage simulators to learn rare 3D objects. Henderson and Ferrari [25] propose to use shading for shape estimation. All existing methods require at least silhouette annotations. Contrary to these works, our method learns explicit 3D structures from categorized natural images without any supervision. Table 2.2 shows a summary.

Even when silhouette and viewpoint annotations are provided in single-view training, the ambiguity of 3D shape is not negligible. For this problem, some methods use human knowledge of shapes as regularizers or constraints. For example, the graph Laplacian of meshes is regularized [36, 111], and shapes are assumed to be symmetric [36]. Instead of designing constraints manually, there are some attempts to acquire prior knowledge of shapes from data. Learning category-specific mean shapes [36, 38] is an example. Adversarial training is another way to learn shape priors. Yang *et al.* [127] and Jiang *et al.* [34] use discriminators on an estimated shape and its corresponding ground truth shape to make the estimated shapes more realistic. Gwak *et al.* [21] and Wu *et al.* [118] use discriminators on generated shapes and a shape collection. In contrast, our method does not require 3D models to learn prior knowledge. Table 2.3 lists a summary of these discriminators.

We address learning single-view 3D object reconstruction with single-view supervision in Chapter 5 and Chapter 6. The closest works to ours in Chapter 5 are Gwak *et al.* [21] and Wu *et al.* [118] because they are also about learning geometric prior from data. While prior of 3D shapes is learned by leveraging 3D models in these works, prior of 2D views is learned in our work. The relation of Chapter 6 and existing works is summarized in Table 2.2.

Chapter 3

Differentiable rendering for neural networks

3.1 Importance of a mesh renderer for neural networks

A rendering function takes 3D geometry, material, lighting, and camera parameters as input and outputs a pixel image. A differentiable rendering function is a rendering function in that the derivative of an output pixel with respect to each input is defined. It is important to note that the derivatives do not have to be the same as the analytical derivatives. Rather, values that are approximated but useful for optimization are better, as demonstrated in several works [57, 58].

Differentiable rendering is necessary to learn 3D reconstruction from 2D images without using 3D shapes. To use images as supervision, images must be rendered from an estimated 3D shape. To optimize the 3D reconstruction function with the gradient method, the gradients about the reconstruction error have to flow through the rendering function.

Differentiable rendering functions can also be applied to other problem settings. For example, in learning to estimate the 3D pose of a human body or hand, 3D shapes are often also given as supervision. A loss of 2D image reconstruction can be used in conjunction with the loss of 3D shape reconstruction. There are also applications where image reconstruction is not an objective function, such as generating 3D adversarial examples [124, 134] or style transfer of 3D models [68].

In this chapter, we propose a novel differentiable rendering layer that is specific to deep learning. Since it is widely known that deep neural networks are remarkably effective in computer vision [47], the integration of differentiable rendering and neural networks is a promising direction. In contrast to general differentiable rendering functions [57], a

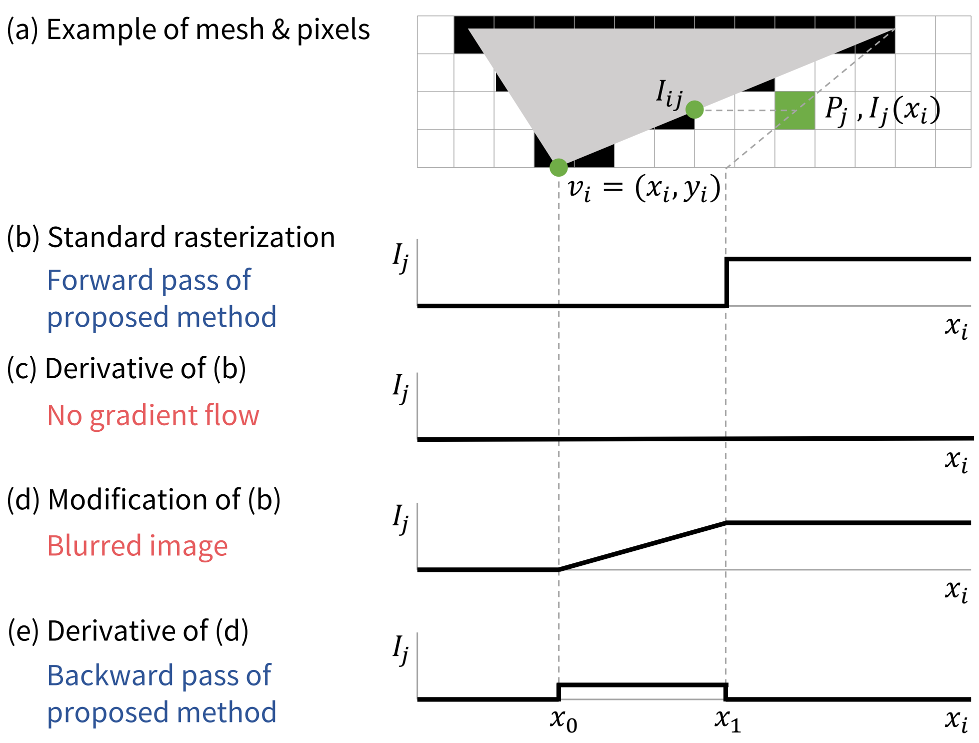


Figure 3.1 Illustration of our method. $\boldsymbol{v}_i = \{x_i, y_i\}$ is one vertex of the face. I_j is the color of pixel P_j. The current position of x_i is x_0. x_1 is the location of x_i where an edge of the face collides with the center of P_j when x_i moves to the right. I_j becomes I_{ij} when $x_i = x_1$.

derivative of a loss function to be optimized with respect to pixels can be used to compute an optimal shape deformation in the case of deep learning. We employ this approach to define approximate gradients of a rendering function.

3.2 Pseudo gradient for rendering

In this section, we describe the technical details of our differentiable renderer named *Neural Renderer*, which is a 3D mesh renderer with gradient flow.

3.2.1 Rendering pipeline and its derivative

A 3D mesh consists of a set of vertices $\{\boldsymbol{v}_1^o, \boldsymbol{v}_2^o, .., \boldsymbol{v}_{N_v}^o\}$ and faces $\{\boldsymbol{f}_1, \boldsymbol{f}_2, .., \boldsymbol{f}_{N_f}\}$, where the object has N_v vertices and N_f faces. $\boldsymbol{v}_i^o \in \mathbb{R}^3$ represents the position of the i-th vertex in the 3D object space and $\boldsymbol{f}_j \in \mathbb{N}^3$ represents the indices of the three vertices corresponding to the j-th triangle face. To render this object, vertices $\{\boldsymbol{v}_i^o\}$ in the object space are transformed into

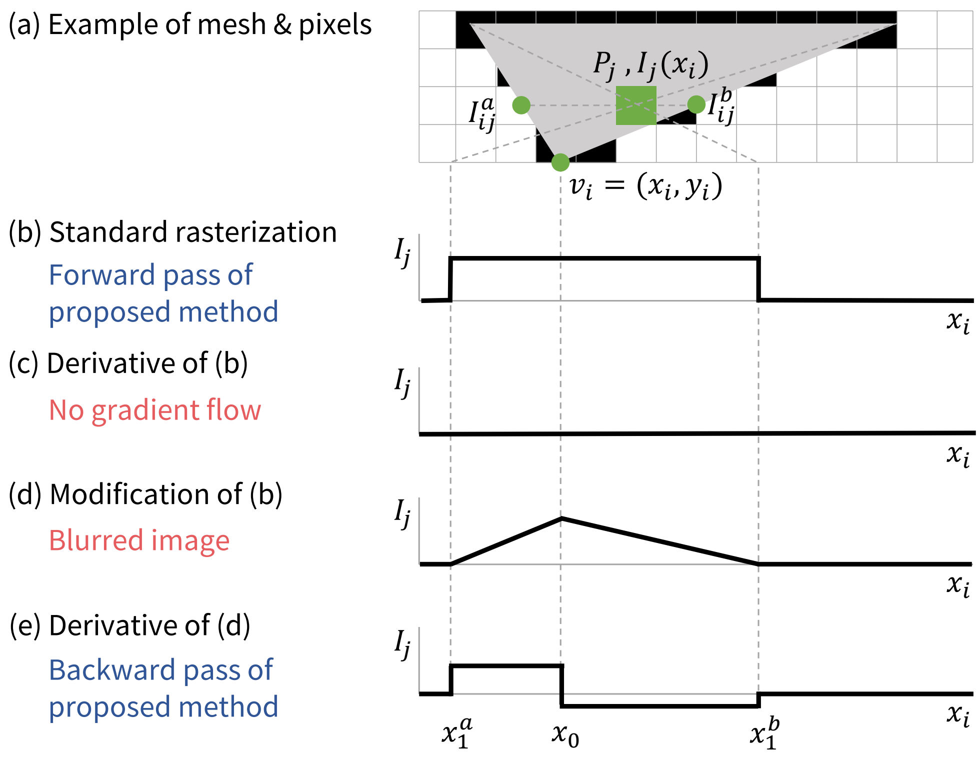


Figure 3.2 Illustration of our method in the case where P_j is inside the face. I_j changes when x_i moves to the right or left.

vertices $\{\boldsymbol{v}^s_i\}$, $\boldsymbol{v}^s_i \in \mathbb{R}^2$ in the screen space using camera parameters. This transformation is represented by a combination of differentiable transformations such as matrix product [60].

An image is generated from $\{\boldsymbol{v}^s_i\}$ and $\{\boldsymbol{f}_j\}$ via sampling in real-time rendering by *rasterization*. Figure 3.1 (a) illustrates rasterization in the case of single triangle. If the center of a pixel P_j is inside of the face, the color I_j of the pixel P_j becomes the color of the overlapping face I_{ij}. Because this is a discrete operation, assuming that I_{ij} is independent of $\boldsymbol{v}_i$, $\frac{\partial I_j}{\partial \boldsymbol{v}_i}$ is zero almost everywhere, as shown in Figure 3.1 (b–c). This means that the error signal back-propagated from a loss function to pixel P_j does not flow into the vertex $\boldsymbol{v}_i$.

3.2.2 Rasterization of a single face

For ease of explanation, we describe our method using the x-coordinate x_i of a single vertex $\boldsymbol{v}_i = \boldsymbol{v}^s_i$ in the screen space and a single gray-scale pixel P_j. We consider the color of P_j to be a function $I_j(x_i)$ on x_i and freeze all variables other than x_i.

First, we assume that P_j is outside the face, as shown in Figure 3.1 (a). The color of P_j is $I(x_0)$ when x_i is at the current position x_0. If x_i moves to the right and reaches the point x_1, where an edge of the face collides with the center of P_j, $I_j(x_i)$ suddenly turns to the color of hitting point I_{ij}. Let δ_i^x be the distance traveled by x_i, let $\delta_i^x = x_1 - x_0$, and let δ_j^I represent the change in the color $\delta_j^I = I(x_1) - I(x_0)$. The partial derivative $\frac{\partial I_j(x_i)}{\partial x_i}$ is zero almost everywhere, as illustrated in Figure 3.1 (b–c).

Because the gradient is zero, the information that $I_j(x_0)$ can be changed by δ_j^I if x_i moves by δ_i^x to the right is not transmitted to x_i. This is because $I_j(x_i)$ suddenly changes. Therefore, we replace the sudden change with a gradual change between x_0 and x_1 using linear interpolation. Then, $\frac{\partial I_j}{\partial x_i}$ becomes $\frac{\delta_j^I}{\delta_i^x}$ between x_0 and x_1, as shown in Figure 3.1 (d–e).

The derivative of $I_j(x_i)$ is different on the right and left sides of x_0. How should one define a derivative at $x_i = x_0$? We propose switching the values using the error signal δ_j^P back-propagated to P_j. The sign of δ_j^P indicates whether P_j should be brighter or darker. To minimize the loss, if $\delta_j^P > 0$, then P_j must be darker. On the other hand, the sign of δ_j^I indicates whether P_j can be brighter or darker. If $\delta_j^I > 0$, P_j becomes brighter by pulling in x_i, but P_j cannot become darker by moving x_i. Therefore, a gradient should not flow if $\delta_j^P > 0$ and $\delta_j^I > 0$. From this viewpoint, we define $\frac{\partial I_j(x_i)}{\partial x_i}|_{x_i=x_0}$ as follows.

$$\left.\frac{\partial I_j(x_i)}{\partial x_i}\right|_{x_i=x_0} = \begin{cases} \frac{\delta_j^I}{\delta_i^x}; & \delta_j^P \delta_j^I < 0. \\ 0; & \delta_j^P \delta_j^I \geq 0. \end{cases} \tag{3.1}$$

Sometimes, the face does not overlap P_j regardless of where x_i moves. This means that x_1 does not exist. In this case, we define $\frac{\partial I_j(x_i)}{\partial x_i}|_{x_i=x_0} = 0$.

We use Figure 3.1 (b) for the forward pass because if we use Figure 3.1 (d), the color of a face leaks outside of the face. Therefore, our rasterizer produces the same images as the standard rasterizer, but it has non-zero gradients.

The derivative with respect to y_i can be obtained by swapping the x-axis and y-axis in the above discussion.

Next, we consider a case where P_j is inside the face, as shown in Figure 3.2 (a). In this case, $I(x_i)$ changes when x_i moves to the right or left. Standard rasterization, its derivative, an interpolated function, and its derivative are shown in Figure 3.2 (b–e). We first compute the derivatives on the left and right sides of x_0 and let their sum be the gradient at x_0. Specifically, using the notation in Figure 3.2, $\delta_j^{I^a} = I(x_1^a) - I(x_0)$, $\delta_j^{I^b} = I(x_1^b) - I(x_0)$, $\delta_x^a = x_1^a - x_0$ and

$\delta_x^b = x_1^b - x_0$, we define the gradients as follows.

$$\left.\frac{\partial I_j(x_i)}{\partial x_i}\right|_{x_i=x_0} = \left.\frac{\partial I_j(x_i)}{\partial x_i}\right|_{x_i=x_0}^{a} + \left.\frac{\partial I_j(x_i)}{\partial x_i}\right|_{x_i=x_0}^{b}. \tag{3.2}$$

$$\left.\frac{\partial I_j(x_i)}{\partial x_i}\right|_{x_i=x_0}^{a} = \begin{cases} \frac{\delta_j^{I^a}}{\delta_x^a}; & \delta_j^P \delta_j^{I^a} < 0. \\ 0; & \delta_j^P \delta_j^{I^a} \geq 0. \end{cases} \tag{3.3}$$

$$\left.\frac{\partial I_j(x_i)}{\partial x_i}\right|_{x_i=x_0}^{b} = \begin{cases} \frac{\delta_j^{I^b}}{\delta_x^b}; & \delta_j^P \delta_j^{I^b} < 0. \\ 0; & \delta_j^P \delta_j^{I^a} \geq 0. \end{cases} \tag{3.4}$$

3.2.3 Rasterization of multiple faces

If there are multiple faces, our rasterizer draws only the frontmost face at each pixel, which is the same as the standard method [60]. During the backward pass, we first check whether or not the cross points I_{ij}, I_{ij}^a, and I_{ij}^b are drawn, and do not flow gradients if they are occluded by surfaces not including $\boldsymbol{v}_i$.

3.2.4 Texture

Textures can be mapped onto faces. In our implementation, each face has its own texture image of size $s_t \times s_t \times s_t$. We determine the coordinates in the texture space corresponding to a position $\boldsymbol{p}$ on a triangle $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3\}$ using the barycentric coordinate system. In other words, if $\boldsymbol{p}$ is expressed as $\boldsymbol{p} = w_1\boldsymbol{v}_1 + w_2\boldsymbol{v}_2 + w_3\boldsymbol{v}_3$, let (w_1, w_2, w_3) be the corresponding coordinates in the texture space. Bilinear interpolation is used for sampling from a texture image.

3.2.5 Lighting

Lighting can be applied directly to a mesh, unlike voxels and point clouds. In this work, we use a simple ambient light and directional light without shadows. Let l^a and l^d be the intensities of the ambient light and directional light, respectively, $\boldsymbol{n}^d$ be a unit vector indicating the direction of the directional light, and $\boldsymbol{n}_j$ be the normal vector of a surface. We then define the modified color of a pixel I_j^l on the surface as $I_j^l = \left(l^a + \left(\boldsymbol{n}^d \cdot \boldsymbol{n}_j\right) l^d\right) I_j$.

In this formulation, gradients also flow into the intensities l^a and l^d, as well as the direction $\boldsymbol{n}^d$ of the directional light. Therefore, light sources can also be optimized.

3.3 Comparison with other rasterization methods

There are three other representative differentiable rasterization methods. OpenDR [57] associates the screen coordinates of a pixel with the gradients of neighboring pixels using Sobel differential filter. SoftRas [54] and DIB-R [8] replace the discrete assignment of a triangle to a pixel with a *soft* continuous assignment of multiple triangles by blurring edges. These two differ in blurring method. Unlike our method, all of these methods do not consider the gradient of a loss function with respect to pixel colors. In this section, we compare rasterization methods by (1) visualizing gradients in a simple case and (2) optimizing a shape with multi-view silhouettes.

3.3.1 Evaluation methodology

Evaluation of differentiable rasterization methods is not so straightforward. The objective is to obtain the derivative of a rasterization function that is useful for shape optimization. As described, analytically-correct derivatives are not always useful. Therefore, the performance of differentiable rasterization has to be measured by optimization. This is in contrast to evaluation of differentiable photorealistic rendering methods. Their objective is to obtain the derivative of Monte Carlo integration. Therefore, approximated analytical derivative by finite differences can be used as ground-truth.

Several papers provide evaluation of differentiable rendering methods by training single-view 3D object reconstruction [8, 54]. However, a fair comparison is hard because setting all conditions the same other than rasterization is difficult. Shape regularization functions are often co-used for training, but different rasterization functions may be suited to different regularization functions. Fair setting of their hyperparameters is also non-trivial. In addition, when RGB images are used for training, the difference of shading function implementations would affect reconstruction accuracy. Analyzing properties of each method by reconstruction is also hard because many factors affect it. Therefore, we think single-view 3D object reconstruction is a too complicated task for evaluation.

We argue that evaluation must be by simple optimization tasks for a fair and informative comparison. Superficially, we conduct two experiments in this thesis.

- Visualizing the gradients of a loss function with respect to vertices in a very simple scene. We also give the derivative of the loss function with respect to pixels. A scene and gradients at pixels are controlled so that the desired movement or deformation of the shape is intuitively understandable. By comparing the desired movement and the visualized gradients, we can qualitatively analyze the behavior of differentiable rasterization methods.

- Optimizing a shape so that its multi-view silhouettes fit given silhouette images. This tasks is relatively complicated because we have to tune some hyperparameters such as learning rate, but useful for measuring the practical effectiveness in optimization quantitatively. We use silhouette only and do not use shaded images to avoid differences in shading implementations. Also, we do not use regularization terms to avoid hyperparameter tuning of them.

These tasks may look too simplified. Nonetheless, in addition to more complicated but practical tasks like learning single-view 3D object reconstruction, we think these tasks are useful for analysis.

3.3.2 Gradient visualization

We visualize the gradients of different rasterization methods in a simple scene illustrated in Figure 3.3 and Figure 3.4. We consider a scene composed of a single triangle colored white ($p = 1$) and black background ($p = 0$). Also, with a loss function $\mathscr{L}$ to be minimized, we assume that the gradient of $\mathscr{L}$ with respect to pixels $\frac{\partial \mathscr{L}}{\partial p}$ has a non-zero value at one pixel. From $\frac{\partial \mathscr{L}}{\partial p}$, we can know that a preferred sign of pixel color change to minimize the loss $\mathscr{L}$. For example, when $0 < \frac{\partial \mathscr{L}}{\partial p}$, p should be smaller to decrease $\mathscr{L}$. In this scene, p becomes smaller when it changes from white to black. Therefore, given a scene and gradients, we can intuitively determine a preferred direction of triangle movement. The first to third rows in these figures represent a scene, gradients, and preferable triangle movements.

The green arrows in Figure 3.3 (d) and Figure 3.4 (d) show the direction of triangle movements computed by our rasterizer. All the results match the intuitively preferable ones. Figure 3.3 (e) and Figure 3.4 (e) show the result of OpenDR [57]. In the rightmost column in Figure 3.3 (e), the triangle does not move because OpenDR uses Sobel differential filter that does not relate pixels that are distant more than two pixels[1]. Because only the sign of gradients differ in Figure 3.3 and Figure 3.4, the moving directions are flipped. However, in general, flipping the sign of gradient does not indicate that the movement must be flipped. For example, the triangle moves left while it should not do in the third column in Figure 3.4 (e). Leveraging $\frac{\partial \mathscr{L}}{\partial p}$ in back-propagation solves this issue. Figure 3.3 (f–g) and Figure 3.4 (f–g) show the result of SoftRas [54] and DIB-R [8]. The triangle does not move in some cases because of small blurring radius. However, if we set it larger, the rendered image becomes too blurry.

[1]In the experiments of the OpenDR paper, the authors use Gaussian blur before computing image reconstruction error probably because of this gradient locality issue. However, handling this locality issue in a rendering function would have advantages since Gaussian blur cannot be used in some cases, such as adversarial training and style transfer.

The rendered images of SoftRas and DIB-R are blurred because they provide gradients by approximating the forward-pass of rendering. On the contrary, ours and OpenDR do not affect rendering quality because they approximate the backward-pass of rendering. Though OpenDR sometimes provides unfavorable gradients, ours improve it by using $\frac{\partial \mathcal{L}}{\partial p}$. This is possible because ours is for neural networks. In general rendering functions, $\frac{\partial \mathcal{L}}{\partial p}$ is not always available.

3.3.3 Optimization

We compare the practical effectiveness of differentiable rasterizers by applying them to optimization. We optimize the vertices of a sphere so that the difference between their silhouettes and given object silhouettes become small. We use 3D models of *bunny* and *teapot*. We render their silhouettes from 24 silhouettes and use them as optimization signal. We also evaluate a modified rasterizer that will be introduced in Chapter 5. We notated it *Ours v2*. In the newer version, we associate a pixel with triangles at its neighboring pixels only instead of all pixels to reduce computation cost. Also, we slightly modify the definition of gradients in Eq. 3.1 to improve stability when δ_i^x is small.

We used Adam optimizer. We tuned learning rate by using OpenDR for *bunny* model and Gaussian pyramid for image comparison, and we set it to 0.03 for all methods. Blurring parameters of Soft Rasterizer (SoftRas) and DIB-R were tuned to obtain the best results. The number of optimization iterations was set to 100. Image size is set to 256×256 pixels.

First, we use the sum of squared error of pixels as a loss function. Figure 3.5 shows the optimized shapes. The optimized silhouettes by our methods are very close to the target silhouette. The silhouette by OpenDR is not good. The gradients by OpenDR are defined only on object edges as demonstrated in Figure 3.3 and Figure 3.4, which probably caused optimization difficulty. The silhouettes by Soft Rasterizer and DIB-R look fine as ours. Figure 3.6 shows the image reconstruction error during optimization. Ours v2 is the best and it is followed by SoftRas, ours v1, DIB-R, and OpenDR. Except for OpenDR, the difference of reconstruction errors is relatively small and it is visually unnoticeable. The reconstructed surfaces are not smooth because we use silhouettes for optimization and we do not use shading information. This is to avoid differences in shading implementation affecting reconstruction. Also, we do not use any regularization methods to make the surfaces smooth to avoid the effects of hyperparameter selection.

In OpenDR paper, Gaussian pyramid is used for image comparison. Figure 3.7 and Figure 3.8 show optimized shapes and convergence when we use Gaussian pyramid in a loss function. We set the number of downsampling to 1. We do not observe significant improvements if we increase the number of downsampling. With Gaussian pyramid, the results by OpenDR are greatly improved. This suggests that using the Gaussian pyramid to make object edges blurred is essential for OpenDR. However, the optimization result of OpenDR is still inferior to other methods both visually and quantitatively. The optimization results using other rasterizers do not differ much if we use the Gaussian pyramid. Note that the Gaussian pyramid can be used only for pixel-level image comparison. When the objective function is a comparison of image features like the perceptual loss [35], image style transfer, or adversarial training, it cannot be used.

Ours and Soft Rasterizer support anti-aliasing by generating a larger image and downscaling it. Figure 3.9 shows the difference of convergence with anti-aliasing. When anti-aliasing is used, convergence is consistently improved. The gradients of pixels are binary when anti-aliasing is not used, and they become continuous and smoother when it is used. This property probably avoids sudden large movement of the vertices and enable more careful and detailed reconstruction. The effect is the most significant in ours v1, and it becomes better than Soft Rasterizer in this optimization experiment.

The discrepancy between ours and approximated rendering (Soft Rasterizer and DIB-R) can be interpreted by blurring. Since ours has no blurred region in rendered images, our convergence is better than the others. As illustrated in Figure 3.3, the blurred area is outside the shape. Therefore, when the difference between blurred images and ground-truth images is minimized, optimized shapes become slightly smaller than correct ones. Though the center of the blurring radius of Soft Rasterizer is on object edges, optimized shapes become smaller. For example, let us assume an estimated shape and ground-truth shape are circles, its estimated radius is r_c, and we blur it with width r_b. The blurred area outside the shape is $(r_c + r_b)^2 - r_c^2$, which is larger than the blurred area inside the shape $r_c^2 - (r_c - r_b)^2$. Therefore, to minimize reconstruction error, the estimated radius r_c should be slightly smaller than the true one. Figure 3.10 shows *false positive* and *false negative* during optimization. False positive is the ratio of pixels that are estimated as an object silhouette, but their corresponding ground-truth are in the background. False negative is the ratio of pixels that are estimated as background, but their corresponding ground-truth are foreground. In Soft Rasterizer and DIB-R, the false negative rate is consistently higher than the false positive rate, which implies that the estimated shape is smaller than the correct one. In contrast, the false negative rate and false positive rate are similar values in our methods[2], which means the estimated size is almost the same as the ground-truth. This data shows that not blurring rendered images is better for optimization.

[2]When anti-aliasing is employed, the curves become smooth. The reduction of the variance is the most significant in *ours v1*. This would be the reason for the improvement of optimization by anti-aliasing.

Method	Implementation	Optimization time (seconds)
Ours	GPU	8
Ours v2	GPU	5
OpenDR	CPU	804
Soft Rasterizer	GPU	60
DIB-R	GPU	19

Table 3.1 Comparison of speed of differentiable rasterizers. Note that this comparison is by our Chainer wrapper for the author implementations of OpenDR, Soft Rasterizer, DIB-R. The wrapper is not well optimized, so the speed of these rasterizers can be several times faster in practice.

Table 3.1 shows the speed for optimization. For each optimization iteration, 24 silhouette images of 224×224 pixels are generated, and their back-propagation is computed. The number of iterations is 100. Since OpenDR is not intended to be used in neural networks, it is implemented for CPU only and does not support parallel rendering of multiple images in a minibatch. Therefore, the speed is much slower than the others. Though we consider the relation of a pixel and its nearby triangles for gradient computation, Soft Rasterizer uses all pairs of pixels and triangles in a scene. Therefore its computation is slow in GPU-based methods.

3.3.4 Discussion

We summarize the advantages and disadvantages of each differentiable rasterization method observed in these two experiments.

OpenDR This is a pioneering work in differentiable rasterization. The main characteristic in optimization is that it does not modify forward rendering, so rendered images are not blurred. However, the gradients are less useful for optimization than the newer methods. Using the Gaussian pyramid improves the performance, but it is still inferior. Rendering speed is also slow because it does not support GPU and parallel rendering of a minibatch because it is not designed for neural networks.

Soft Rasterizer In this method, the gradients are provided by rendering blurred silhouettes. The derivative is non-approximated and mathematically valid. The implementation is simple because we can take the benefit of auto-differentiation frameworks. In shape optimization, convergence is as fast as ours and DIB-R. Estimated shapes become slightly smaller than correct ones because of blurring.

DIB-R The difference between DIB-R and Soft Rasterizer in rasterization is how to blur silhouettes. While Soft Rasterizer distributes blurred region equally in outside and inside silhouettes, DIB-R distributes it only outside silhouettes. Therefore, estimated shapes are further smaller.

Ours This method does not affect rendered images as with OpenDR. It works well for pixel-level optimization, and convergence in optimization is slightly better than Soft Rasterizer and DIB-R because it does not blur rendered images. Also, it is the only method that consider the derivative of a loss function with respect to pixels in back-propagation, which is expected to be stabilize optimization. Though the gradients of our first version are less stable, probably because the upper bound is not limited. Our second version improves the stability.

As a renderer, not as a rasterizer, ours has some drawbacks. One only supports a simple Lambert model for shading, and it does not have more advanced shading methods. Lighting is also limited. Though the implementation is in CUDA, it is not optimized well, and it could be more efficient. Also, it only supports Chainer in popular neural network libraries.

3.4 Simple applications

Before applying our render to learning single-view 3D object reconstruction, we apply it to gradient-based 3D mesh editing without training neural networks to test its differentiability with respect to vertices and texture images. Specifically, we conduct a 3D version of style transfer [16] and DeepDream [67]. We denote an image of a mesh m rendered from a viewpoint ϕ_i by $R(m, \phi_i)$. Gradient-based image editing techniques [16, 67] generate an image by minimizing a loss function $\mathscr{L}(x)$ on a 2D image x via gradient descent. In this work, instead of generating an image, we optimize a 3D mesh m consisting of vertices $\{\boldsymbol{v}_i\}$, faces $\{\boldsymbol{f}_i\}$, and textures $\{\boldsymbol{t}_i\}$ based on its rendered image $R(m|\phi_i)$. Figure 3.11 and Figure 3.12 illustrate the optimization procedure of the proposed 2D-to-3D style transfer and 3D DeepDream, respectively.

For 2D images, style transfer is achieved by minimizing *content loss* and *style loss* simultaneously [16]. Specifically, content loss is defined using a feature extractor $f_c(x)$ and content image x^c as $\mathscr{L}_c(x|x^c) = |f_c(x) - f_c(x^c)|_2^2$. Style loss is defined using another feature extractor $f_s(x)$ and style image x^s as $\mathscr{L}_s(x|x^s) = |M(f_s(x)) - M(f_s(x^s))|_F^2$. $M(x)$ transforms a vector into a Gram matrix.

In 2D-to-3D style transfer, content is specified as a 3D mesh m^c. To make the shape of the generated mesh similar to that of m^c, assuming that the vertices-to-faces relationships $\{\boldsymbol{f}_i\}$ are the same for both meshes, we redefine content loss as $\mathscr{L}_c(m|m^c) = \sum_{\{\boldsymbol{v}_i, \boldsymbol{v}_i^c\} \in (m, m^c)} |\boldsymbol{v}_i - \boldsymbol{v}_i^c|_2^2$. We use the same style loss as that in the 2D application. Specifically, $\mathscr{L}_s(m|x^s, \phi) = |M(f_s(R(m, \phi))) - M(f_s(x_s))|_F^2$. We also use a regularizer for noise reduction. Let $\mathscr{P}$ denote the a set of colors of all pairs of adjacent pixels in an image $R(m, \phi)$. We define this loss as $\mathscr{L}_t(m|\phi) = \sum_{\{\boldsymbol{p}_a, \boldsymbol{p}_b\} \in \mathscr{P}} |\boldsymbol{p}_a - \boldsymbol{p}_b|_2^2$.

The objective function is $\mathscr{L} = \lambda_c \mathscr{L}_c + \lambda_s \mathscr{L}_s + \lambda_t \mathscr{L}_t$. We set an initial solution of m as m^c and minimize $\mathscr{L}$ with respect to $\{\boldsymbol{v}_i\}$ and $\{\boldsymbol{t}_i\}$.

We applied 2D-to-3D style transfer to the objects shown in Figure 3.13. Optimization was conducted using the Adam optimizer [43] with $\beta_1 = 0.9$, and $\beta_2 = 0.999$. We rendered images of size 448×448 and downsampled them to 244×224 to eliminate aliasing. The batch size was set to 4. During optimization, images were rendered at random elevations and azimuth angles. Texture size was set to $s_t = 4$. The style images we used were selected from [11, 35]. λ_c, λ_s, and λ_t are manually tuned for each input. The feature extractors f_s for style loss were `conv1_2`, `conv2_3`, `conv3_3`, and `conv4_3` from the VGG-16 network [97]. The intensities of the lights were $l^a = 0.5$ and $l^d = 0.5$, and the direction of the light was randomly set during optimization. The α value of Adam was set to $2.5\text{e}{-4}, 5\text{e}{-2}$ for $\{\boldsymbol{v}_i\}, \{\boldsymbol{t}_i\}$. The number of parameter updates was set to $5{,}000$.

Figure 3.14 presents the representative results of 2D-to-3D style transfer. Additional results are shown in Figures 3.16, 3.17, 3.18, 3.19. The styles of the paintings were accurately transferred to the textures and shapes. From the outline of the bunny and the lid of the teapot, we can see the straight style of Coupland and Gris. The wavy style of Munch was also transferred to the side of the teapot. Interestingly, the side of the tower of Babel was transferred only to the side, not to the upside, of the bunny.

3.4.1 3D DeepDream

We extend DeepDream for 2D images to 3D shapes and textures. Figure 3.12 illustrates the optimization procedure of the proposed method.

Let $f(x)$ be a function that outputs a feature map of an image x. For 2D images, a DeepDream of image x_0 is achieved by minimizing $-|f(x)|_F^2$ via gradient descent starting from $x = x_0$. Optimization is halted after a few iterations. Following a similar process, we minimize $-|f(R(m, \phi))|_F^2$ with respect to $\{\boldsymbol{v}_i\}$ and $\{\boldsymbol{t}_i\}$.

In the experiments of 3D DeepDream, we use the same objects, optimizer, and rendering setting as with 2D-to-3D style transfer. Different from 2D-to-3D style transfer, images are rendered without lighting. The feature extractor was the `inception_4c` layer from GoogLeNet [100]. The α value of Adam was set to $5\text{e}{-5}, 1\text{e}{-2}$ for $\{\boldsymbol{v}_i\}, \{\boldsymbol{t}_i\}$. Optimization is stopped after $1{,}000$ iterations.

Figure 3.15 presents the results of DeepDream. A nose and eyes emerged on the face of the bunny. The spout of the teapot expanded and became the face of a bird, while the body appeared similar to a bus. These transformations matched the 3D shape of each object.

3.5 Summary

In this section, we proposed approximated gradients for mesh rendering functions. The derivative of a loss function to be optimized with respect to pixels is given in neural networks. We found that gradients computed with this information are more useful for optimizing vertices than those of a general differentiable rendering function [57]. We also found that differentiable rendering methods by blurring images tend to estimate object shapes smaller, whereas our non-approximated rendering does not suffer from this issue. We verified gradients of textures and vertices by our renderer using 2D-to-3D style transfer and 3D DeepDream, which are novel applications.

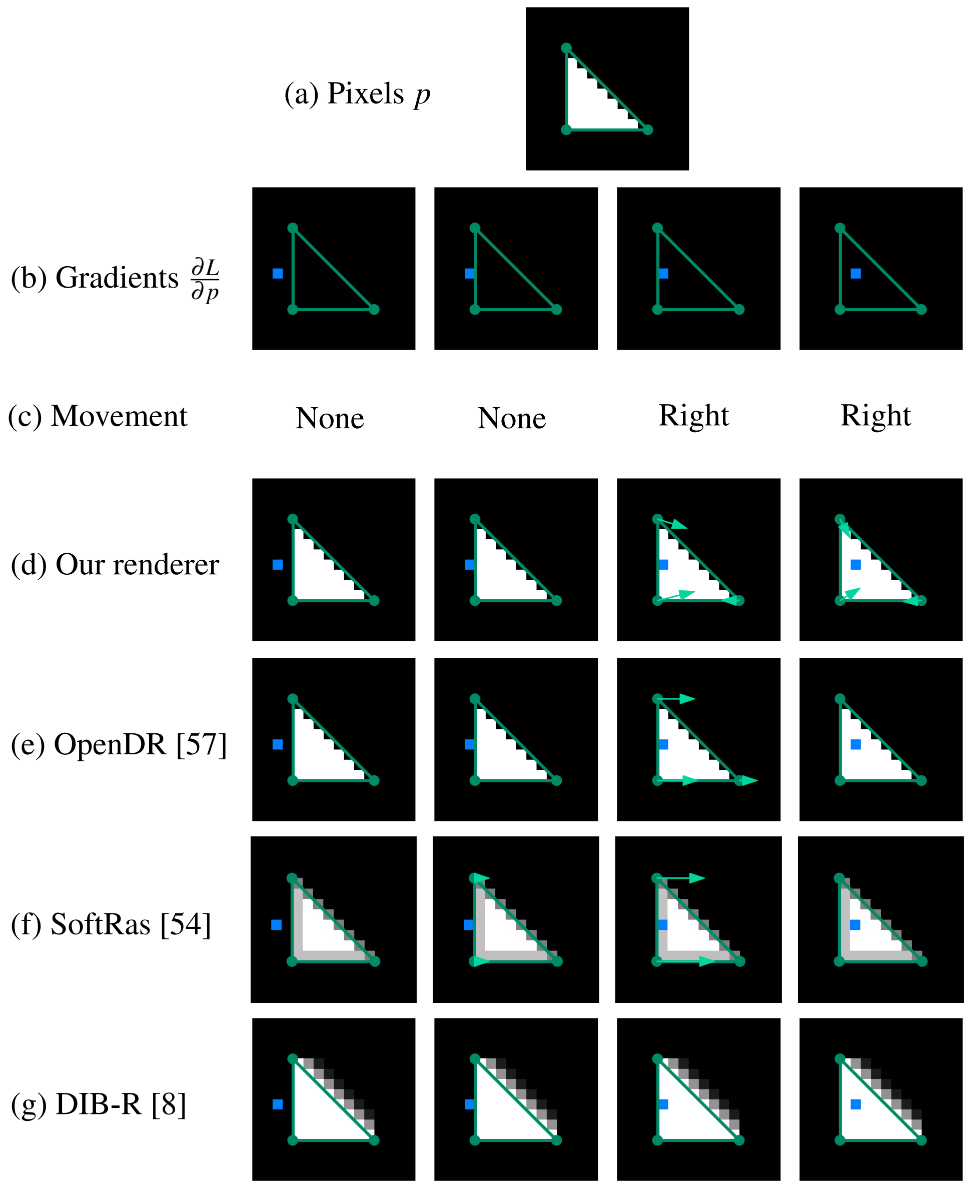


Figure 3.3 Comparison of differentiable rasterization methods. (a) A scene to be studied. Green points and lines show a triangle. Pixels on and outside the triangle are colored white ($p = 1$) and black ($p = 0$),respectively. (b) Four cases of gradients. Black pixels indicate $\frac{\partial \mathscr{L}}{\partial p} = 0$ and blue pixels indicate $0 < \frac{\partial \mathscr{L}}{\partial p}$. (c) Preferable movement of triangles. Blue pixels have to become black from white to minimize $\mathscr{L}$. When blue pixels are on the triangle, it is achieved by moving the triangle to the right. (d–f) Green arrows show the moving directions of the vertices by gradient decent. Ours is the only one that matches (c).

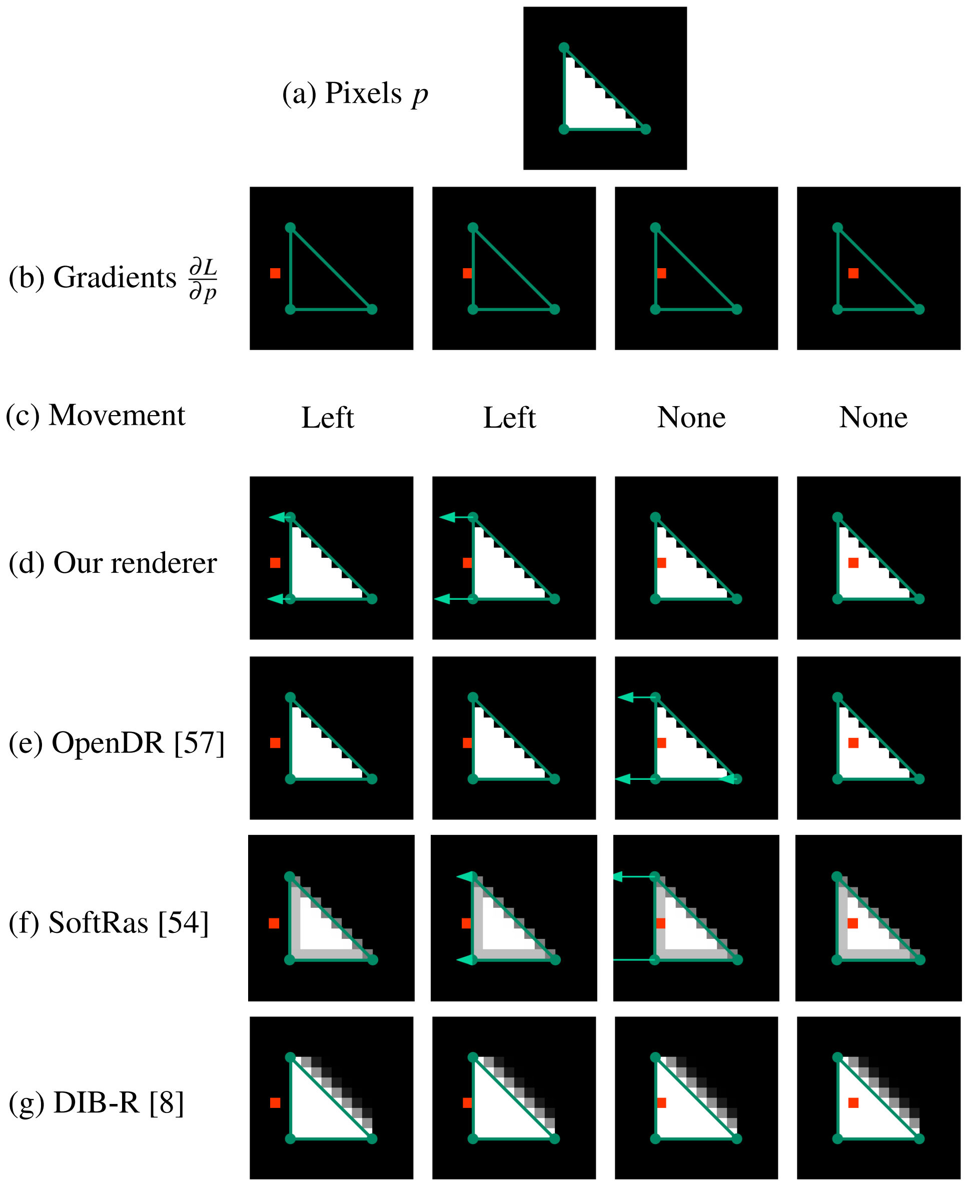


Figure 3.4 Comparison of differentiable rasterization methods. (a) A scene to be studied. (b) Four cases of gradients. Red pixels indicate $\frac{\partial \mathscr{L}}{\partial p} < 0$. (c) Preferable movement of triangles. Red pixels have to become white from black to minimize $\mathscr{L}$. When red pixels are outside the triangle, it is achieved by moving the triangle to the left. (d–f) Green arrows show the moving directions of the vertices by gradient decent. Ours is the only one that matches (c).

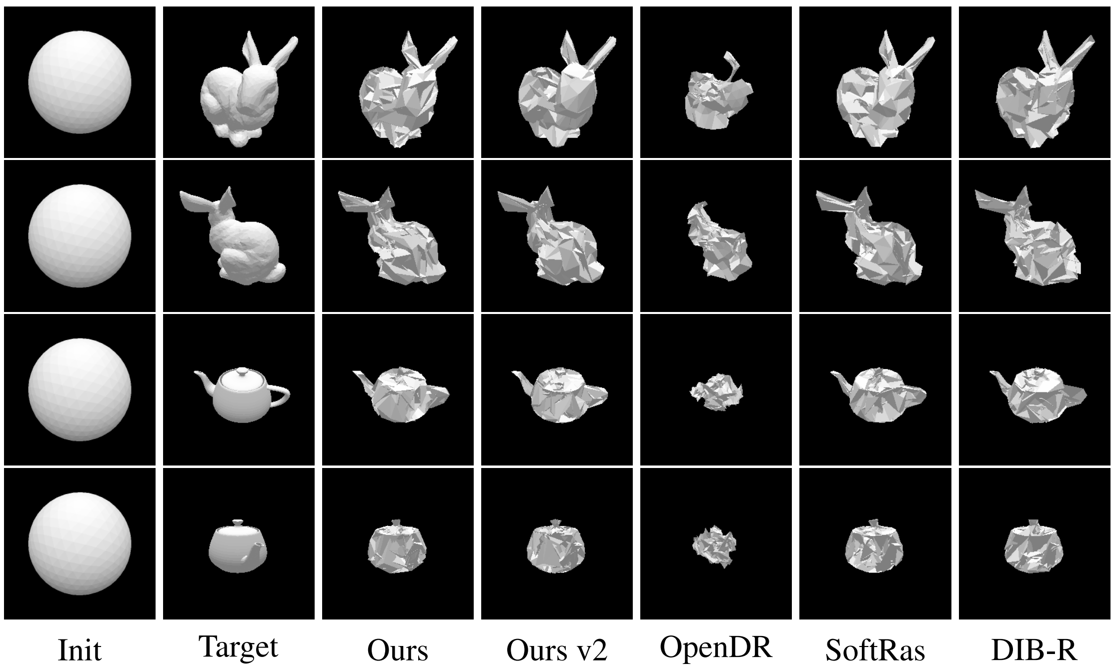
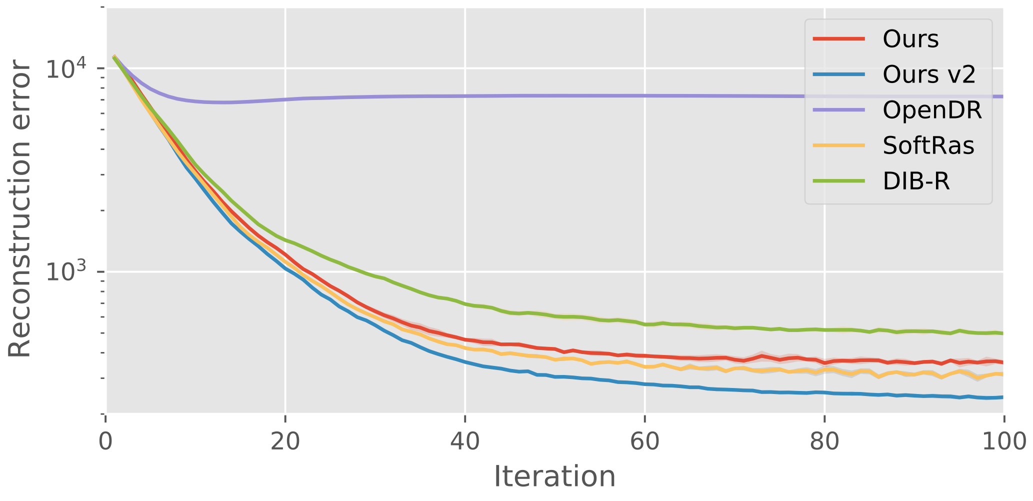


Figure 3.5 Optimized shapes by different differentiable rasterization methods. Two views are shown for each object.



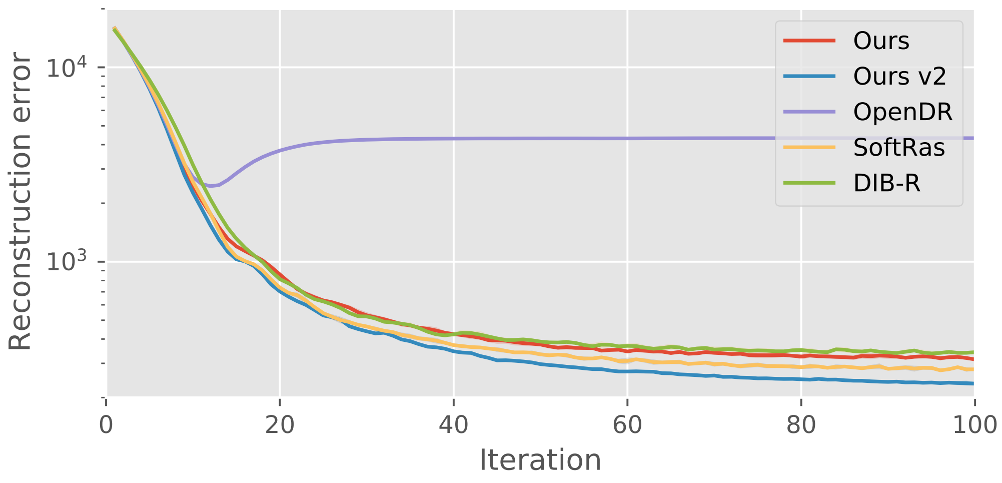


Figure 3.6 Image reconstruction error during optimization. The used object models are *bunny* (upper) and *teapot* (lower).

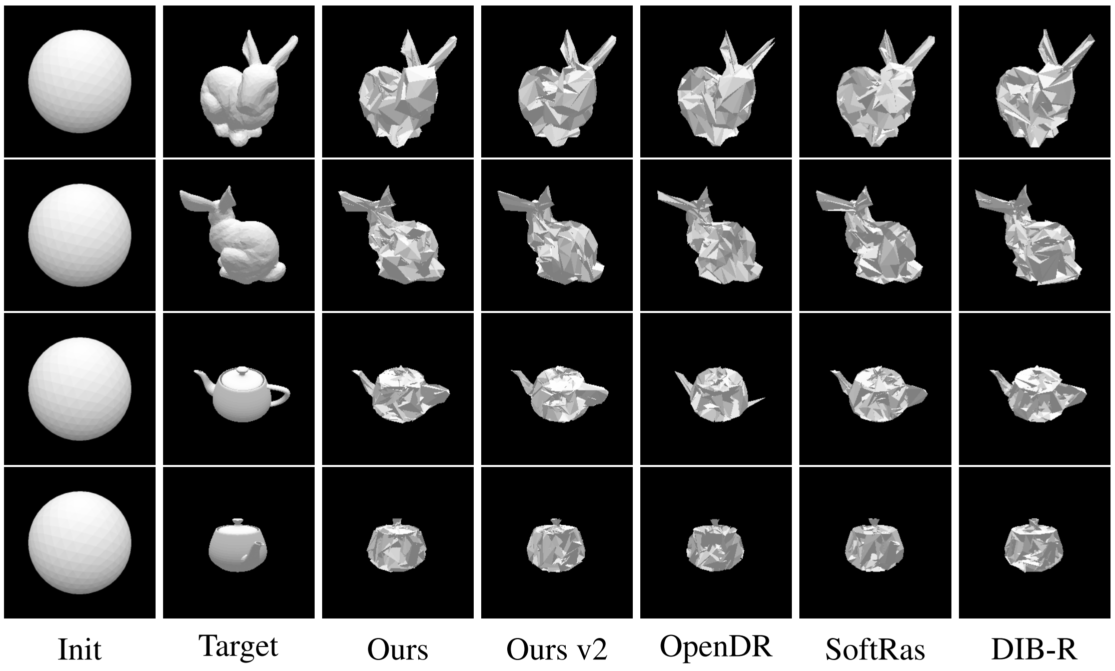
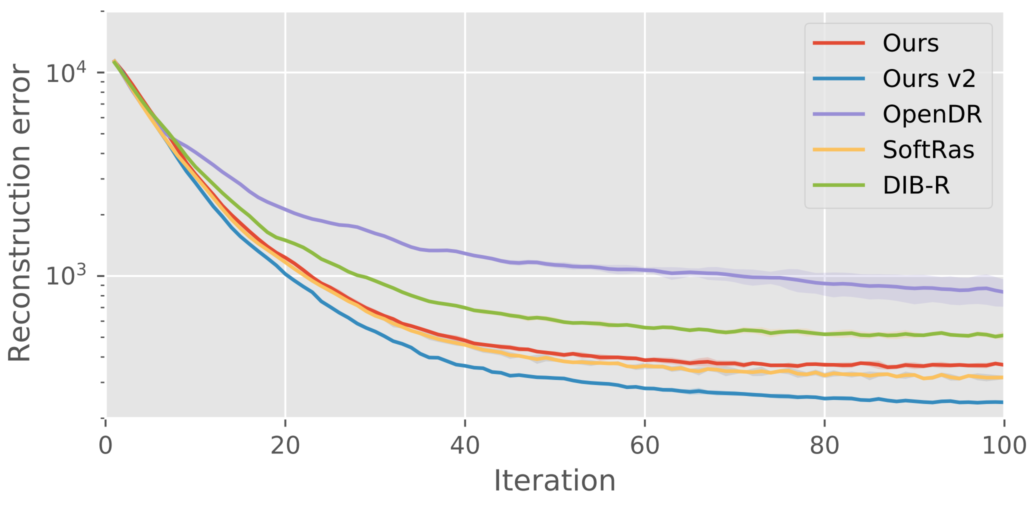


Figure 3.7 Optimized shapes by different differentiable rasterization methods. Gaussian pyramid, the number of downsampling of which is one, is used for a loss function. Two views are shown for each object.



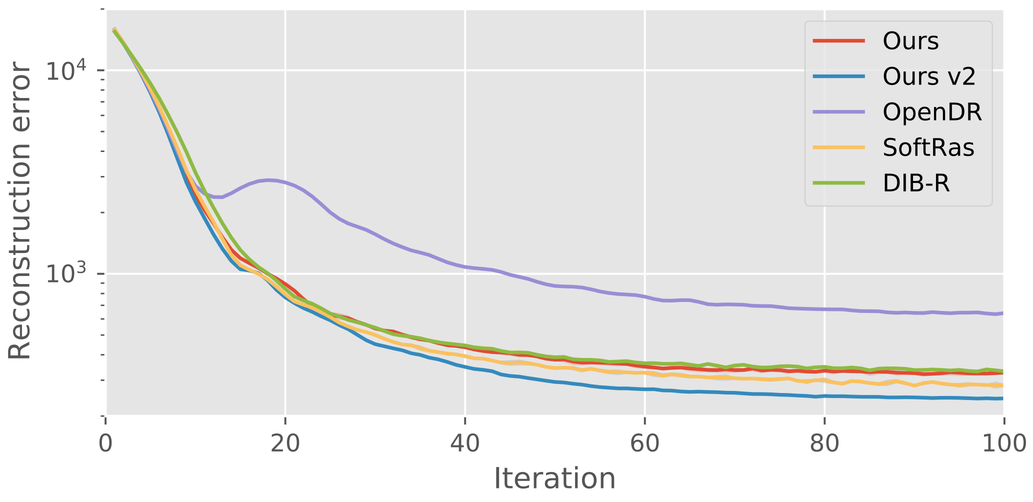
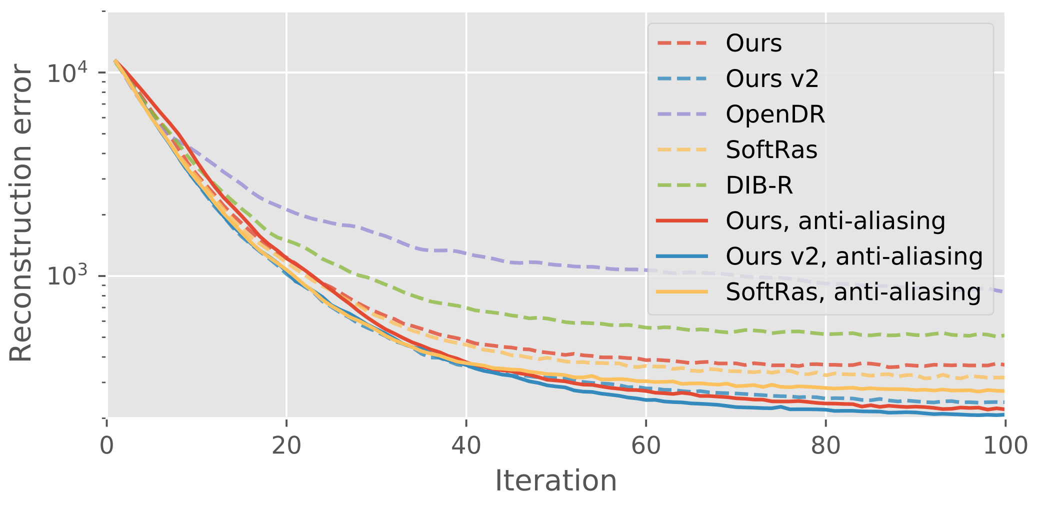


Figure 3.8 Image reconstruction error during optimization. Gaussian pyramid, the number of downsampling of which is one, is used for a loss function. The used object models are *bunny* (upper) and *teapot* (lower).



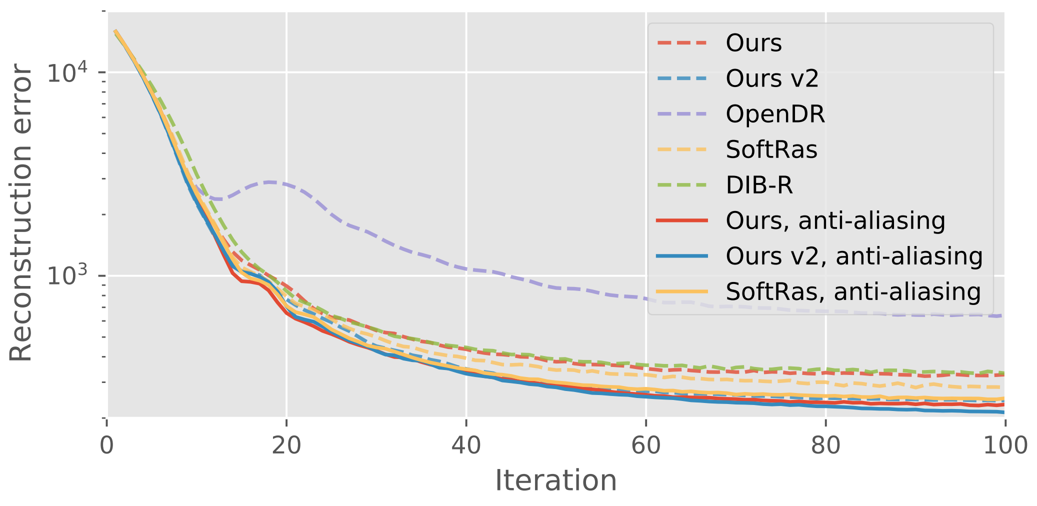
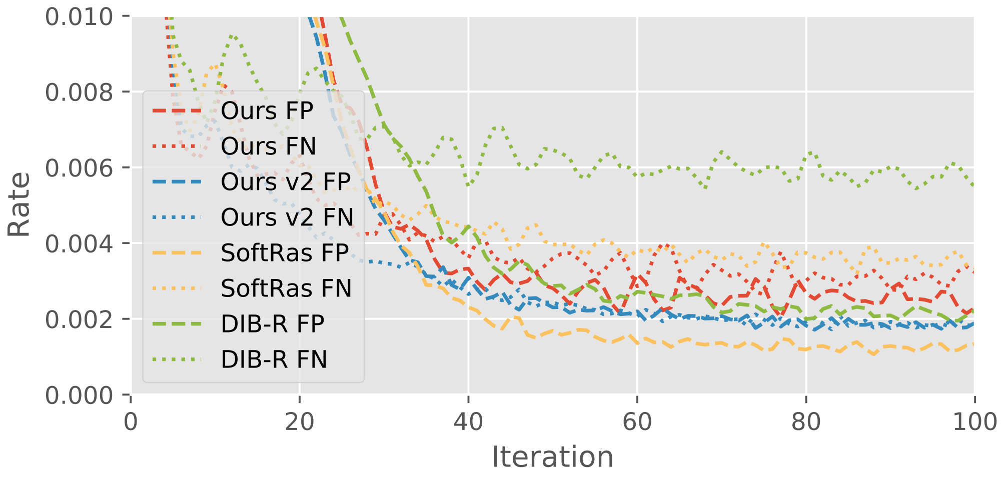


Figure 3.9 Image reconstruction error during optimization. Gaussian pyramid, the number of downsampling of which is one, is used for a loss function. Anti-aliasing of rasterization is used if supported. The used object models are *bunny* (upper) and *teapot* (lower).



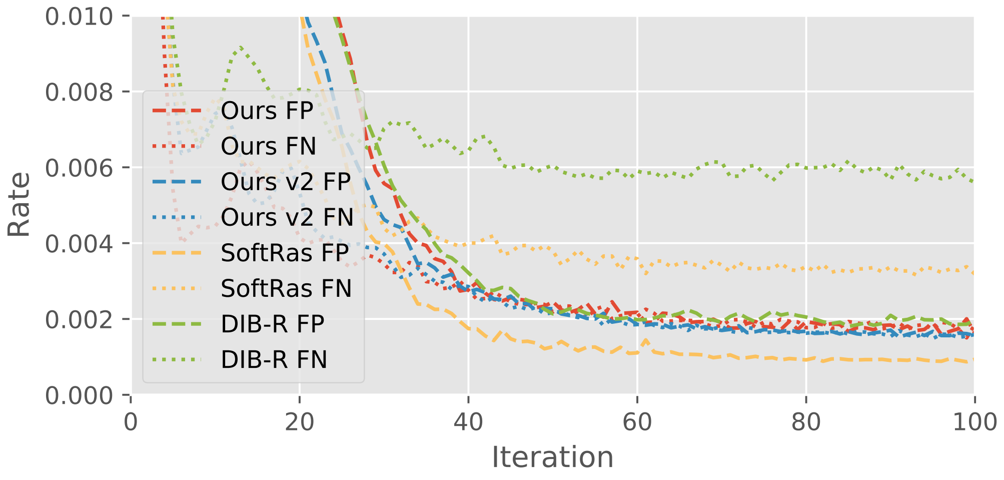


Figure 3.10 False positive (FP) and false negative (FN) rate during optimization. Gaussian pyramid, the number of downsampling of which is one, is used for a loss function. Anti-aliasing is employed in the lower figure. The used object model is *bunny*.

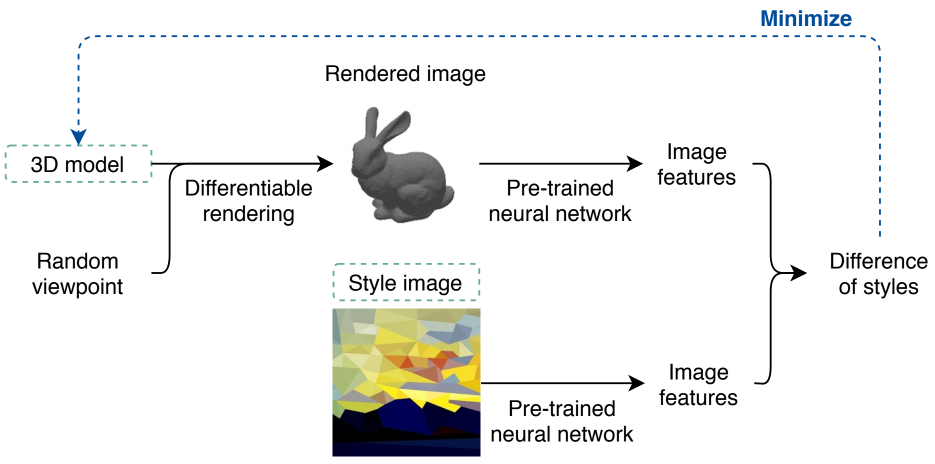


Figure 3.11 The optimization procedure of the proposed 2D-to-3D style transfer.

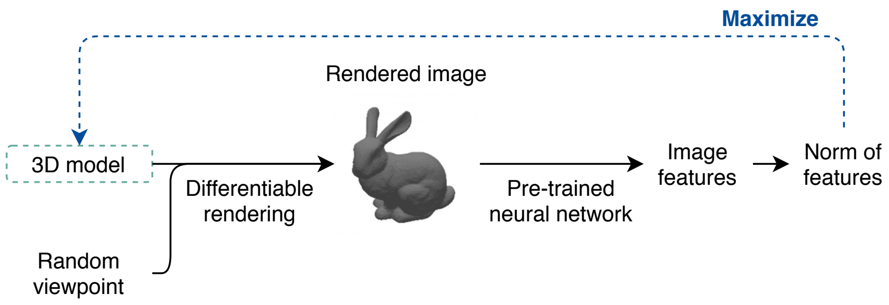


Figure 3.12 The optimization procedure of the proposed 3D DeepDream.

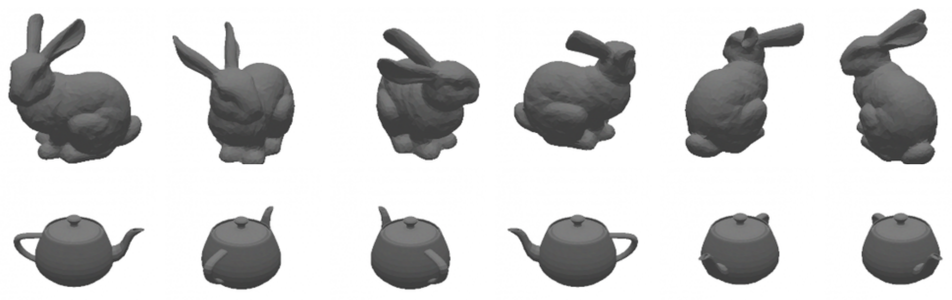


Figure 3.13 Initial state of meshes in style transfer and DeepDream. Rendered from six viewpoints.

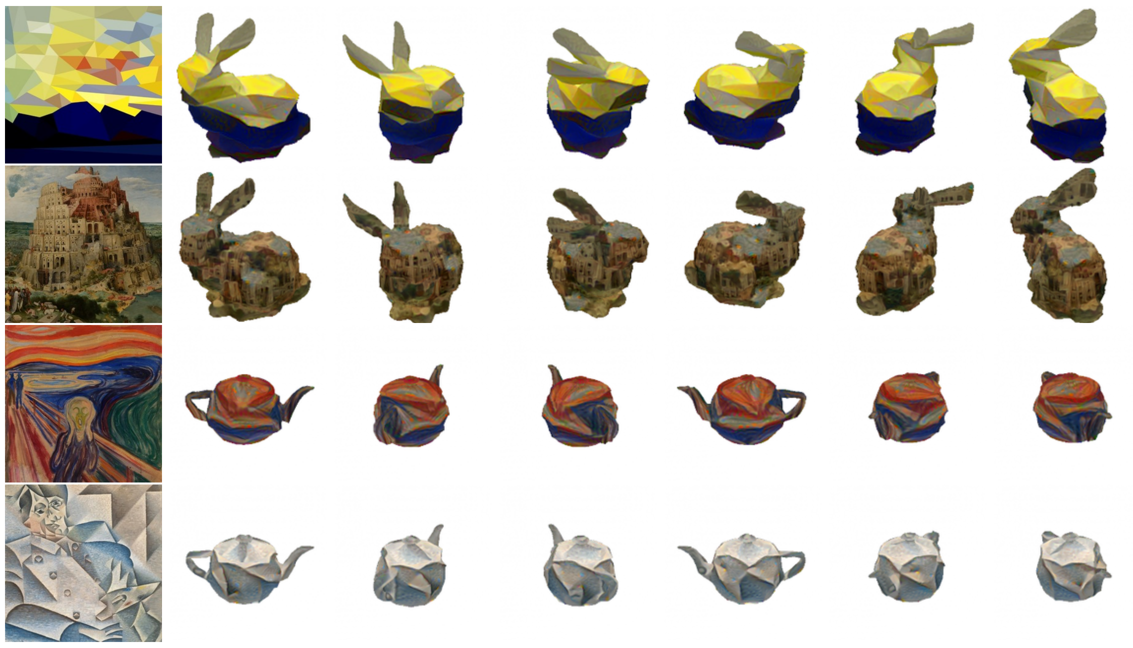


Figure 3.14 2D-to-3D style transfer. The leftmost images represent styles. The style images are *Thomson No. 5 (Yellow Sunset)* (D. Coupland, 2011), *The Tower of Babel* (P. Bruegel the Elder, 1563), *The Scream* (E. Munch, 1910), and *Portrait of Pablo Picasso* (J. Gris, 1912).

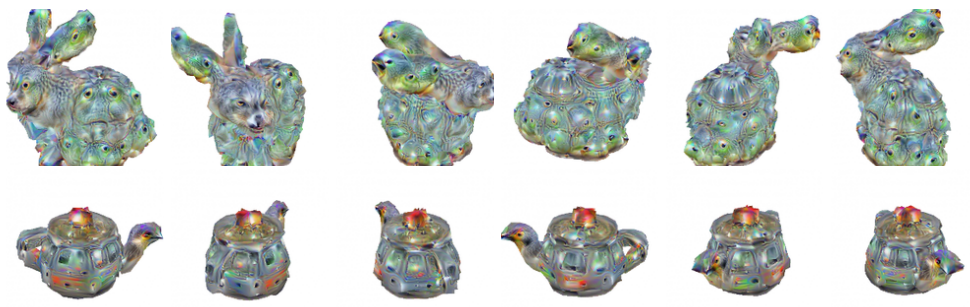


Figure 3.15 DeepDream of 3D mesh.

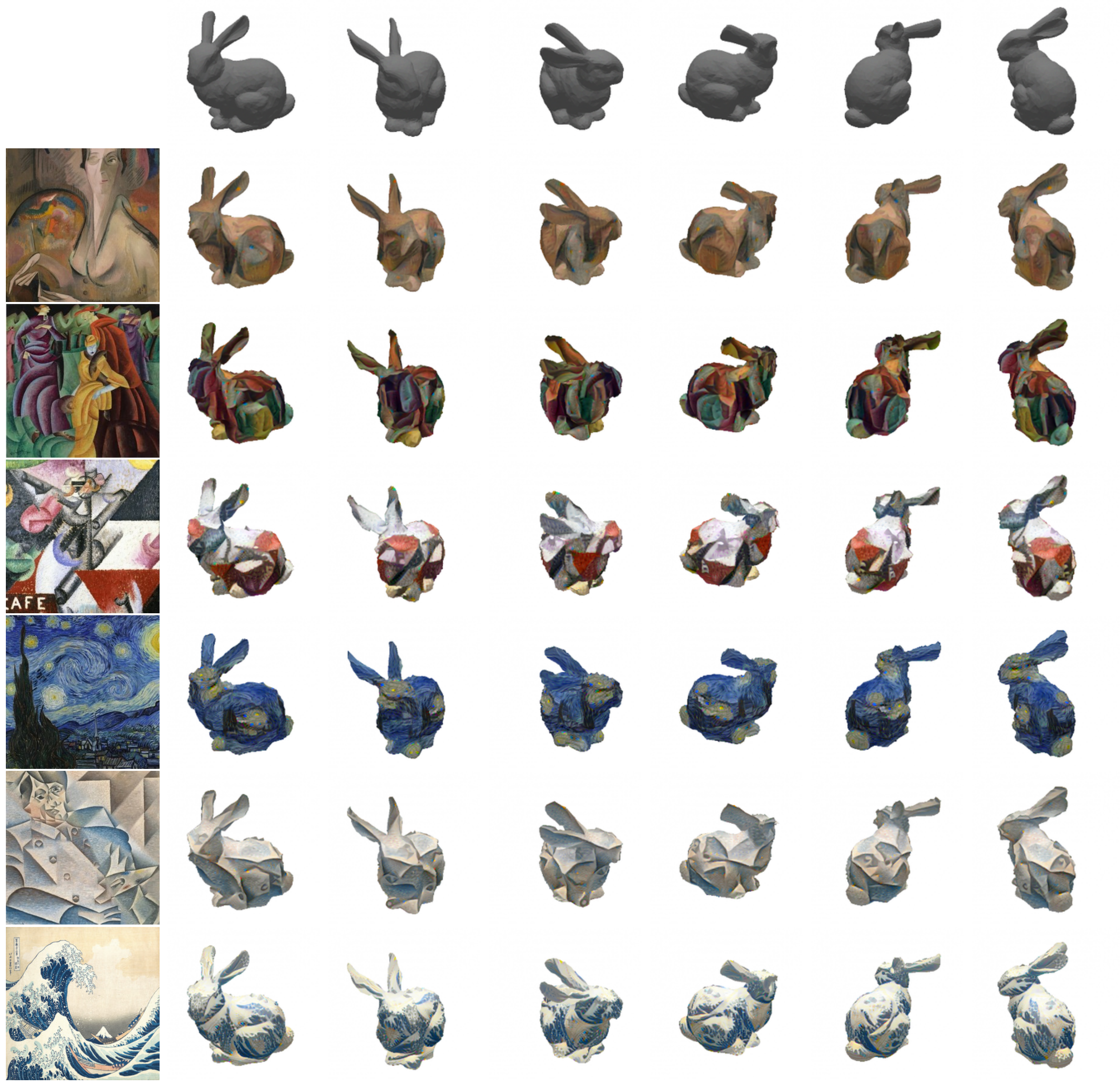


Figure 3.16 Additional results of style transfer. The style images are *Self-Portrait* (A. Bailly, 1917), *Jesuits III* (L. Feininger, 1915), *Ritmo plastico del 14 luglio* (S. Gino, 1913), and *The Starry Night* (V. van Gogh, 1889), *Portrait of Pablo Picasso* (J. Gris, 1912), and *The Great Wave off Kanagawa*, (Hokusai, 1829-1832).

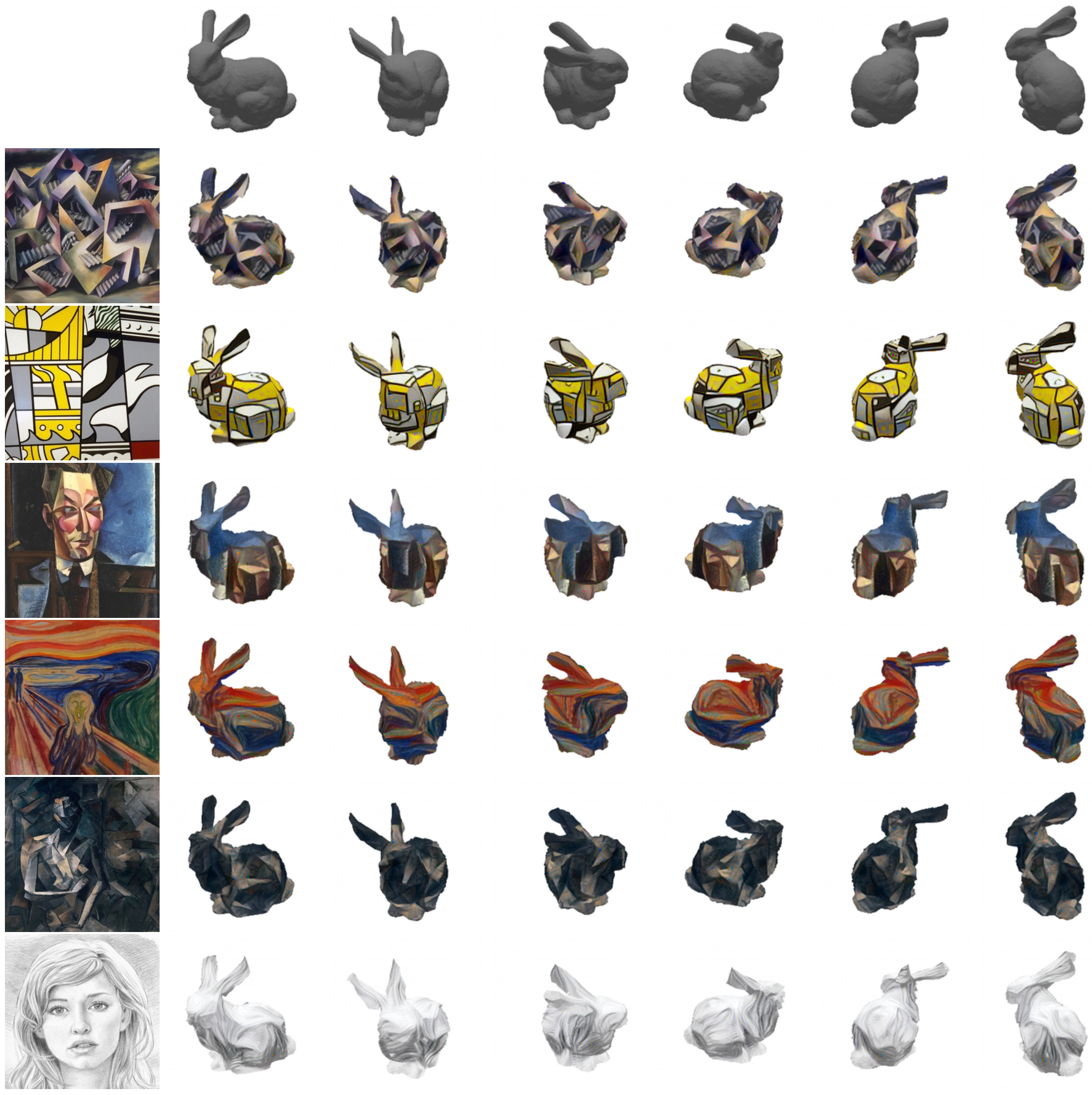


Figure 3.17 Additional results of style transfer. The style images are *The Trial* (W. Lettl, 1981), *Bicentennial Print* (R. Lichtenstein, 1975), *Portrait of a Friend* (M. H. Maxy, 1926), *The Scream* (E. Munch, 1910), *Femme nue assise* (P. Picasso, 1909), and *Sketch* [35].

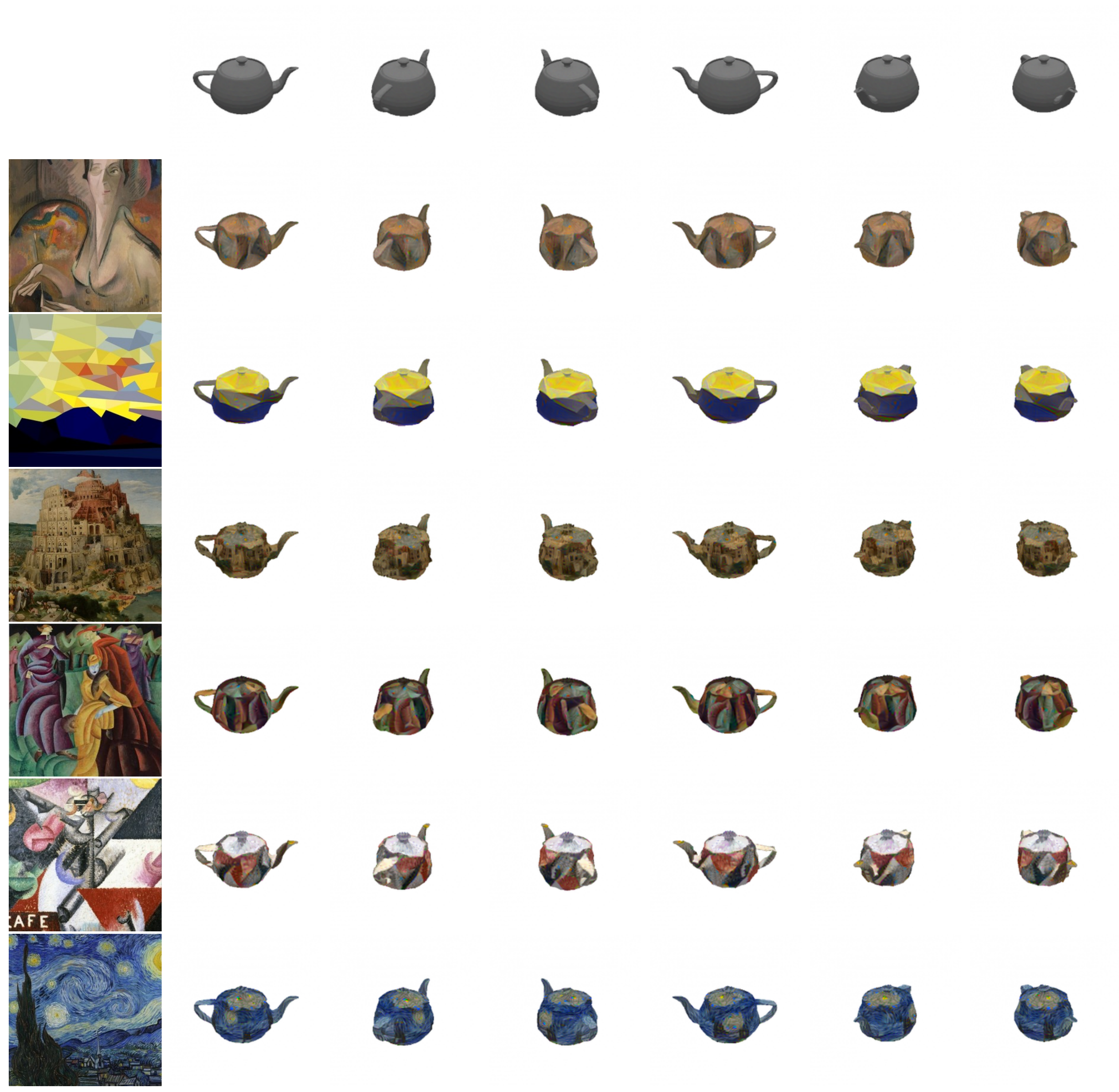


Figure 3.18 Additional results of style transfer. The style images are *Self-Portrait* (A. Bailly, 1917), *Thomson No. 5 (Yellow Sunset)* (D. Coupland, 2011), *The Tower of Babel* (P. Bruegel the Elder, 1563), *Jesuits III* (L. Feininger, 1915), *Ritmo plastico del 14 luglio* (S. Gino, 1913), and *The Starry Night* (V. van Gogh, 1889).

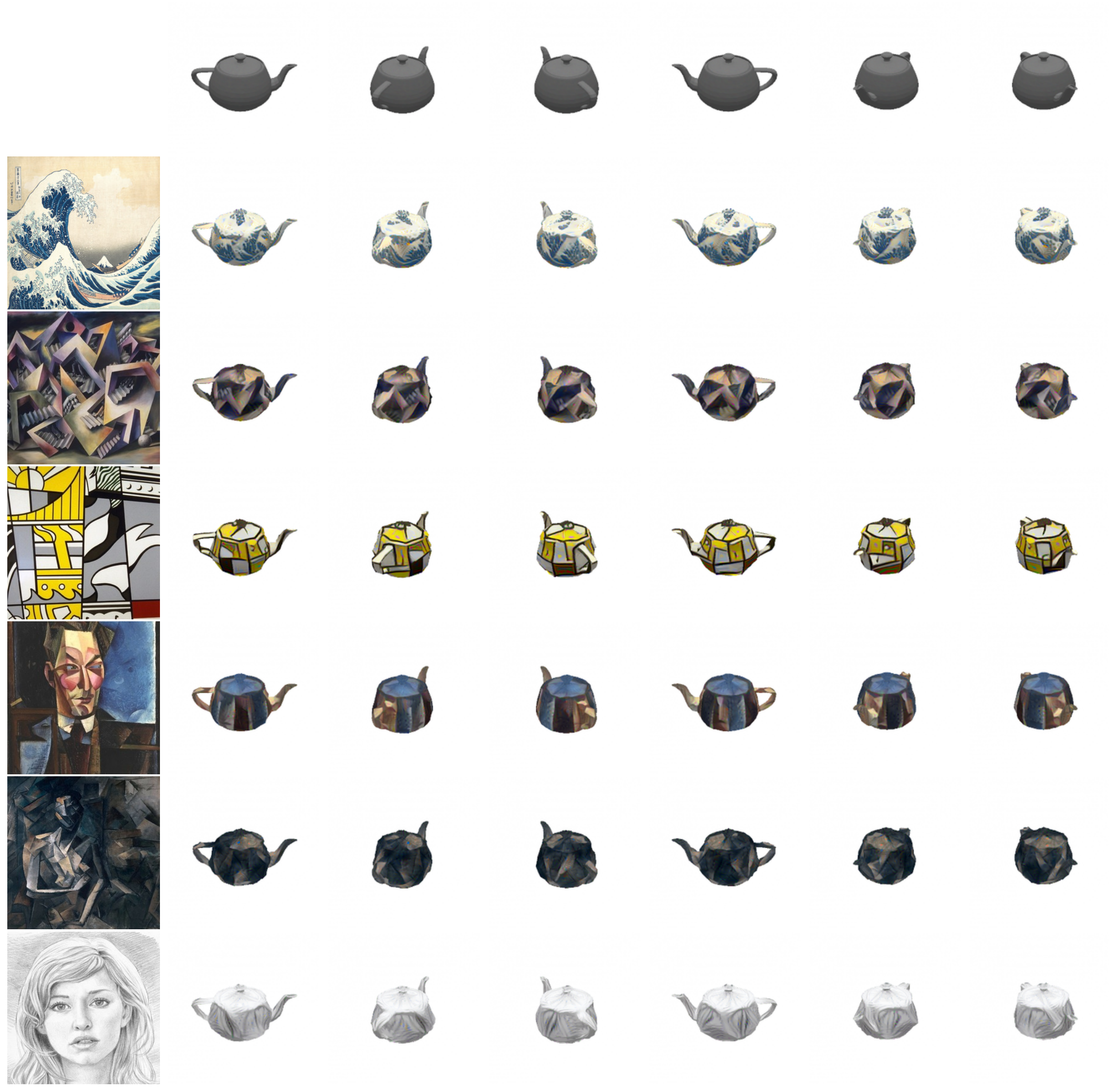


Figure 3.19 Additional results of style transfer. The style images are *The Great Wave off Kanagawa*, (Hokusai, 1829-1832), *The Trial* (W. Lettl, 1981), *Bicentennial Print* (R. Lichtenstein, 1975), *Portrait of a Friend* (M. H. Maxy, 1926), *Femme nue assise* (P. Picasso, 1909), and *Sketch* [35].

Chapter 4

Learning with multi-view images

4.1 Difficulty in learning 3D reconstruction from images

When learning 3D reconstruction from 2D images, we have to integrate a module that projects an estimated 3D shape to a 2D plane into the training pipeline. A few works have used voxel rendering methods to achieve this goal [107, 126], but none has used meshes that are more suited to learning 3D reconstruction by rendering[1]. We employ the rendering method proposed in the previous chapter for this task.

This chapter addresses the simplest and technically easiest case of learning 3D object reconstruction from images. We assume multiple images of a single object from various viewpoints are given for training. We also assume the extrinsic and intrinsic parameters of cameras and the foreground masks of the images are available. Collecting such data requires a 3D scanning system composed of calibrated cameras. Therefore, the cost of such a dataset is not very low. Nevertheless, the cost is relatively lower than creating 3D shape datasets.

4.2 Method

Yan *et al.* [126] demonstrated that single-image 3D reconstruction can be realized without 3D training data. In their setting, a 3D generation function $G(x)$ on an image x was trained such that silhouettes of a predicted 3D shape $\{\hat{s}_i = R(G(x), \phi_i)\}$ matches the ground truth silhouettes $\{s_i\}$, assuming that the viewpoints $\{\phi_i\}$ are known. This pipeline is illustrated in Figure 1.2. While Yan *et al.* [126] generated voxels, we generate a mesh.

Although voxels can be generated by extending existing 2D image generators [17, 83] to 3D, mesh generation is not so straightforward. In this work, instead of generating a mesh

[1]At the time of submitting our original paper [41].

from scratch, we deform a predefined mesh to generate a new mesh. Specifically, we use an isotropic sphere with 642 vertices and move each vertex $\boldsymbol{v}_i$ as $\boldsymbol{v}_i + \boldsymbol{b}_i + \boldsymbol{c}$ using a local bias vector $\boldsymbol{b}_i$ and global bias vector $\boldsymbol{c}$. Additionally, we restrict the movable range of each vertex within the same quadrant on the original sphere. The faces $\{\boldsymbol{f}_i\}$ are unchanged. Therefore, the intermediate outputs of $G(x)$ are $\boldsymbol{b} \in \mathbb{R}^{642\times 3}$ and $\boldsymbol{c} \in \mathbb{R}^{1\times 3}$. The mesh we use is specified by 642×3 parameters, which is far less than the typical voxel representation with a size of 32^3. This low-dimensionality is presumably beneficial for shape estimation.

The generation function $G(x)$ is trained using silhouette loss $\mathscr{L}_{\text{sl}}$ and smoothness loss $\mathscr{L}_{\text{sm}}$. Silhouette loss represents how much the reconstructed silhouettes $\{\hat{s}_i\}$ differ from the correct silhouettes $\{s_i\}$. Smoothness loss represents how smooth the surfaces of a mesh are and acts as a regularizer. The objective function is a weighted sum of these two loss functions $\mathscr{L} = \lambda_{\text{sl}}\mathscr{L}_{\text{sl}} + \lambda_{\text{sm}}\mathscr{L}_{\text{sm}}$.

Let $\{s_i\}$ and $\{\hat{s}_i\}$ be binary masks, θ_i be the angle between two faces including the i-th edge in $G(x)$, $\mathscr{E}$ be the set of all edges in $G(x)$, and $\odot$ be an element-wise product. We define the loss functions as:

$$\begin{aligned} \mathscr{L}_{\text{sl}}(x|\phi_i, s_i) &= -\frac{|\hat{s}_i \odot s_i|_1}{|\hat{s}_i + s_i - \hat{s}_i \odot s_i|_1}. && (4.1) \\ \mathscr{L}_{\text{sm}}(x) &= \sum_{\theta_i \in \mathscr{E}} (\cos\theta_i + 1)^2. && (4.2) \end{aligned}$$

$\mathscr{L}_{\text{sl}}$ corresponds to a negative intersection over union (IoU) between the true and reconstructed silhouettes. $\mathscr{L}_{\text{sm}}$ ensures that intersection angles of all faces are close to 180 degrees.

We assume that the object region in an image is segmented via preprocessing in common with the exiting works [13, 107, 126]. We input the mask of the object region into the generator as an additional channel of an RGB image.

4.3 Experiments

4.3.1 Experimental settings

To compare our mesh-based method with the voxel-based approach by Yan *et al.* [126], we used nearly the same dataset as they did[2]. We used 3D objects from thirteen categories in the ShapeNetCore [7] dataset. Images were rendered from 24 azimuth angles with a fixed elevation angle, under the same camera setup, and lighting setup using Blender. The render

[2]The dataset we used was not exactly the same as that used in [126]. The rendering parameters for the input images were slightly different. Additionally, while our silhouette images were rendered by Blender from the meshes in the ShapeNetCore dataset, theirs were rendered by their PTNs using voxelized data.

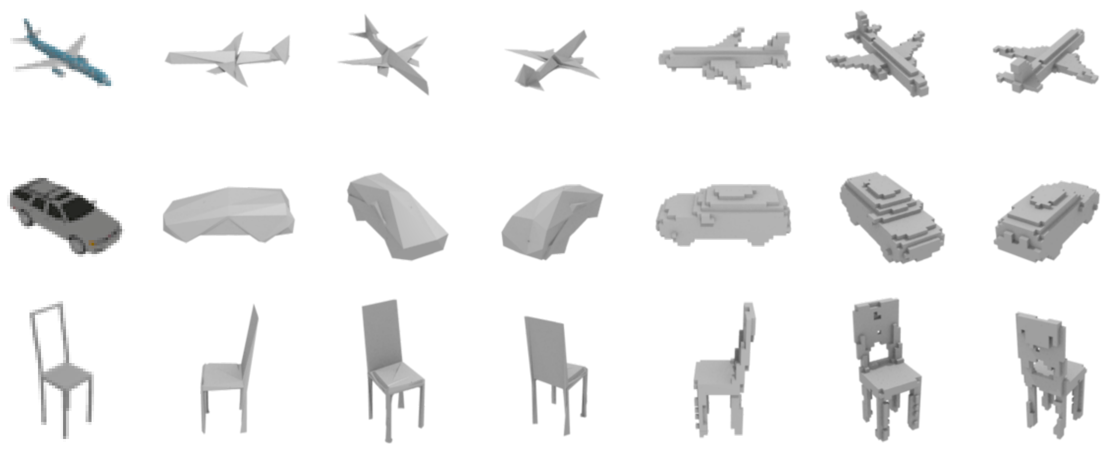


Figure 4.1 3D mesh reconstruction from a single image. Results are rendered from three viewpoints. First column: input images. Second through fourth columns: mesh reconstruction (proposed method). Fifth through seventh columns: voxel reconstruction [126].

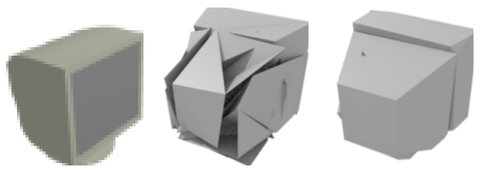


Figure 4.2 Generation of the back side of a CRT monitor with/without smoothness regularizer. Left: input image. Center: prediction without regularizer. Right: prediction with regularizer.

size was 64×64 pixels. We used the same training, validation, and test sets as those used in [126].

We compared reconstruction accuracy between the voxel-based and retrieval-based approaches [126]. In the voxel-based approach, $G(x)$ is composed of a convolutional encoder and deconvolutional decoder. While their encoder was pre-trained using the method in Yang *et al.* [128], our network works well without any pre-training. In the retrieval-based approach, the nearest training image is retrieved using the `fc6` feature of a pre-trained VGG network [97]. The corresponding voxels are regarded as a predicted shape. Note that the retrieval-based approach uses ground truth voxels for supervision. To evaluate the reconstruction performance quantitatively, we voxelized both the ground truth meshes and the generated meshes to compute the intersection over union (IoU) between the voxels. The size of voxels was set to 32^3. For each object in the test set, we performed 3D reconstruction using the images from 24 viewpoints, calculated the IoU scores, and reported the average score.

	Retrieval [126]	Voxel-based [126]	Mesh-based (ours)
`airplane`	0.5564	0.5556	**0.6172**
`bench`	0.4875	0.4924	**0.4998**
`dresser`	0.5713	0.6823	**0.7143**
`car`	0.6519	**0.7123**	0.7095
`chair`	0.3512	0.4494	**0.4990**
`display`	0.3958	0.5395	**0.5831**
`lamp`	0.2905	**0.4223**	0.4126
`loudspeaker`	0.4600	0.5868	**0.6536**
`rifle`	0.5133	0.5987	**0.6322**
`sofa`	0.5314	0.6221	**0.6735**
`table`	0.3097	**0.4938**	0.4829
`telephone`	0.6696	0.7504	**0.7777**
`vessel`	0.4078	0.5507	**0.5645**
`mean`	0.4766	0.5736	**0.6016**

Table 4.1 Reconstruction accuracy measured by voxel IoU. Higher is better. Our mesh-based approach outperforms the voxel-based approach [126] in ten out of thirteen categories.

We used an encoder-decoder architecture for the generator $G(x)$. Our encoder is nearly identical to that of [126], which encodes an input image into a 512D vector. Our decoder is composed of three fully-connected layers. The sizes of the hidden layer are 1024 and 2048. The render size of our renderer is set to 128×128 and downsampled them to 64×64. We rendered only the silhouettes of objects without using textures and lighting. We set $\lambda_{\text{sl}} = 1$ and $\lambda_{\text{sm}} = 0.001$ in Section 4.3.2, and $\lambda_{\text{sm}} = 0$ in Section 4.3.3. We trained our generator using the Adam optimizer [43] with $\alpha = 0.0001$, $\beta_1 = 0.9$, and $\beta_2 = 0.999$. The batch size was set to 64. In each minibatch, we included silhouettes from two viewpoints per input image.

4.3.2 Qualitative evaluation

We trained thirteen models with images from each class. Figure 4.1 presents a part of results from the test set by our mesh-based method and the voxel-based method [126][3]. These results demonstrate that a mesh can be correctly reconstructed from a single image using our method.

Compared to the voxel-based approach, the shapes reconstructed by our method are more visually appealing from the two points. One is that a mesh can represent small parts, such as airplane wings, with high resolution. The other is that there is no cubic artifacts in a mesh.

[3]We trained generators using the code from the authors and our dataset.

Although low resolutions and artifacts may not be a problem in tasks such as picking by robots, they are disadvantageous for computer graphics, computational photography, and data augmentation.

Without using the smoothness loss, our model sometimes produces very rough surfaces. That is because the smoothness of surfaces has little effect on silhouettes. With the smoothness regularizer, the surface becomes smoother and looks more natural. Figure 4.2 illustrates the effectiveness of the regularizer. However, if the regularizer is used, the voxel IoU for the entire dataset becomes slightly lower.

Figure 4.3 and Figure 4.4 show additional results of 3D reconstruction.

4.3.3 Quantitative evaluation

We trained a single model using images from all classes. The reconstruction accuracy is shown in Table 4.1. Our mesh-based approach outperforms the voxel-based approach [126] for ten out of thirteen categories. Our result is significantly better for the `airplane`, `chair`, `display`, `loudspeaker`, and `sofa` categories. The basic shapes of the `loudspeaker` and `display` categories are simple. However, the size and position vary depending on the objects. The fact that a meshes are suitable for scaling and translation presumably contributes to the performance improvements in these categories. The variations in shapes in the `airplane`, `chair` and `sofa` categories are also relatively small.

Our approach did not perform very well for the `car`, `lamp`, and `table` categories. The shapes of the objects in these categories are relatively complicated, and they are difficult to be reconstructed by deforming a sphere.

4.3.4 Limitation

Although our reconstruction method already surpasses the voxel-based method in terms of visual appeal and voxel IoU, it has a clear disadvantage in that it cannot generate objects with various topologies. In order to overcome this limitation, it is necessary to generate the faces-to-vertices relationship $\{\boldsymbol{f}_i\}$ dynamically. This is beyond the scope of this study, but it is an interesting direction for future research.

4.4 Summary

Using the renderer proposed in the previous chapter, we proposed a method to reconstruct a 3D mesh from a single image, the performance of which is superior to the existing voxel-based approach [126] in terms of visual appeal and the voxel IoU metric. Ours is the first work to learn 3D mesh reconstruction from 2D images. We found that, compared with voxel reconstruction, accuracy of mesh reconstruction is better in shapes that have thin structures such as `airplane` and `chair`, and shapes that are simple such as `display` and `loudspeaker`.

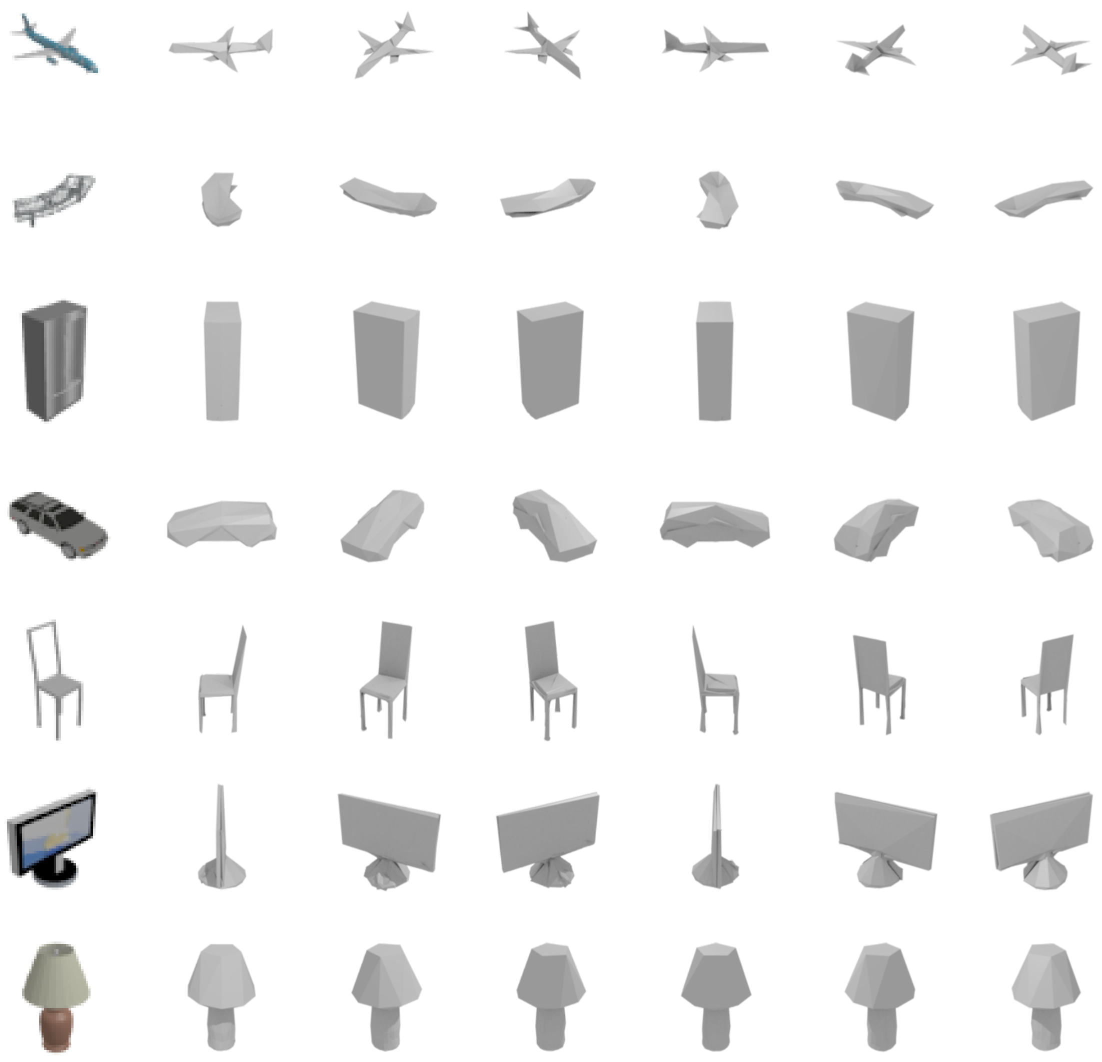


Figure 4.3 3D mesh reconstruction from a single image. The leftmost images are the inputs. Results are rendered from six viewpoints.

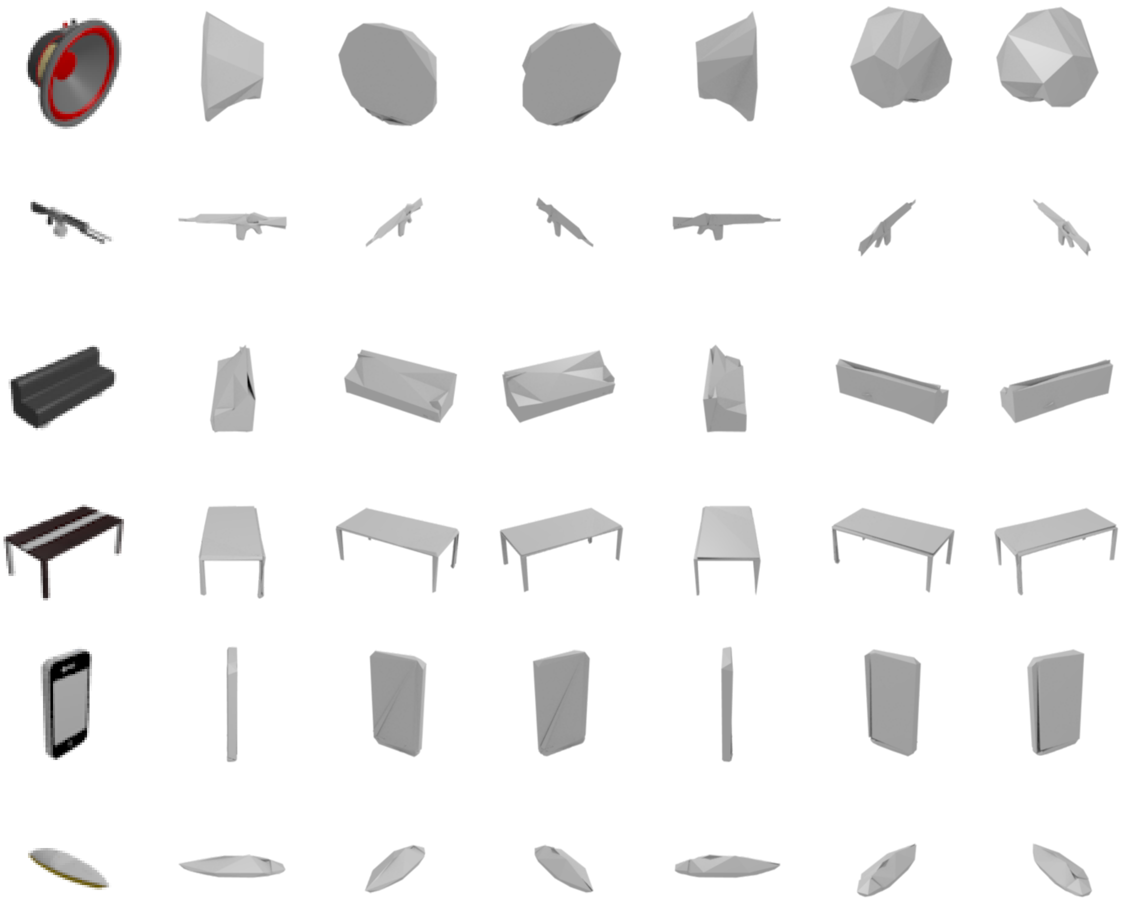


Figure 4.4 3D mesh reconstruction from a single image. The leftmost images are the inputs. Results are rendered from six viewpoints.

Chapter 5

Learning with annotated single-view images

5.1 Difficulty in learning from single-view images

Learning single-view 3D object reconstruction from single-view images is preferable because collecting them is less expensive than collecting multi-view images. Multi-view image collection requires specialized equipment, while single-view image collection involves just taking a photograph. Another way is to collect images that are on the Internet.

The problem in learning 3D reconstruction with a single image per object is the high ambiguity of the 3D shape in a single image. If a 3D shape is given as supervision, there is no ambiguity of shape at all. In the case of learning from multi-view images as in the previous chapter, there is some ambiguity. However, if the number of given views is large, it is not a severe problem. However, in the case of a single image per object, as shown in Figure 1.5, we cannot obtain good results only by minimizing image reconstruction error. This is because views of reconstructed objects from other viewpoints are not taken into account.

In this chapter, we propose a way to learn the validity of a shape's appearance from data. This criterion is used in conjunction with image reconstruction during learning and significantly improves 3D object reconstruction performance. This chapter was published as [40].

5.2 Method

In this section, we introduce a simple view-based method in the previous chapter. Then, we describe our main technique, called *view prior learning (VPL)*. We also explain a technique

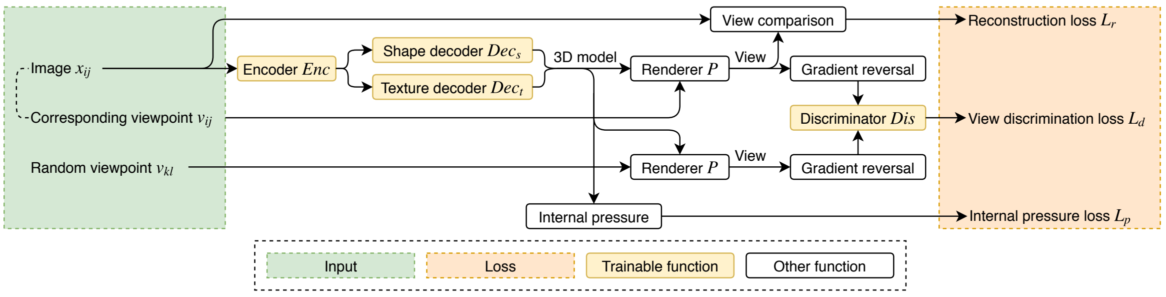


Figure 5.1 Architecture of the proposed method. The main point of our method is the use of discrimination loss to learn priors of views. While the discriminator aims to minimize discrimination loss, the encoder and decoders try to maximize it using a gradient reversal layer.

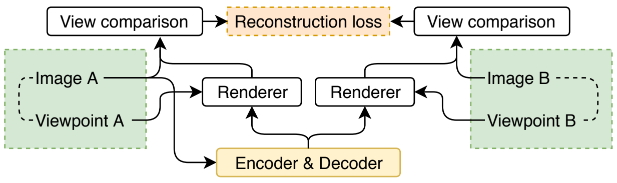


Figure 5.2 Reconstruction loss in multi-view training. Images A and B are views of the same object. Although only the loss with respect to a view reconstructed from image A is shown in this figure, the loss with respect to image B is also computed.

to further improve reconstruction accuracy by applying internal pressure to shapes. Figure 5.1 shows the architecture of our method.

For training, our method requires a dataset that contains single or multiple views of objects, and their silhouette and viewpoint annotations, similar to previous studies [36, 107, 126]. Additionally, ours can also use class labels of views if they are available. After training, reconstruction is performed without silhouette, viewpoint, and class label annotations.

5.2.1 View-based training for 3D reconstruction

In this section, we describe our baseline method for 3D reconstruction. We extend a method which uses silhouettes in the previous chapter to handle textures using a texture decoder and perceptual loss [35].

Overview. The common approach to view-based training of 3D reconstructors is to minimize the difference between views of a reconstructed shape and views of a ground truth shape. Let x_{ij} be the view of an object o_i from a viewpoint v_{ij}, N_o be the number of objects in the training dataset, N_v be the number of viewpoints per object, $R(\cdot)$ be a reconstructor that takes an image and outputs a 3D model, $P(\cdot,\cdot)$ be a renderer that takes a 3D model and a viewpoint and outputs the view of the given model from the given viewpoint, and $\mathscr{L}_v(\cdot,\cdot)$ be a function that measures the difference between two views. Then, our reconstruction loss is defined as

$$\mathscr{L}_r(x,v) = \sum_{i=1}^{N_o} \sum_{j=1}^{N_v} \sum_{k=1}^{N_v} \mathscr{L}_v(P(R(x_{ij}),v_{ik}),x_{ik}). \tag{5.1}$$

We call the case where $N_v = 1$ *single-view training*. In this case, the reconstruction loss is simplified to $\mathscr{L}_r(x,v) = \sum_{i=1}^{N_o} \mathscr{L}_v(P(R(x_{i1}),v_{i1}),x_{i1})$. We call the case where $2 \leq N_v$ *multi-view training*.

3D representation and renderer. Some works use voxels as a 3D representation in view-based training [107, 126]. However, voxels are not well suited to view-based training because using high-resolution views of voxels is difficult as voxels are memory inefficient. Therefore, we use a mesh and a differentiable renderer[1].

Reconstructor. In this work, a 3D model is represented by a pair of a shape and a texture image. Our reconstructor $R(\cdot)$ uses an encoder-decoder architecture. An encoder $Enc(\cdot)$ encodes an input image, and a shape decoder $Dec_s(\cdot)$ and texture decoder $Dec_t(\cdot)$ generate a 3D mesh and a texture image, respectively. We generate a shape by moving the vertices of a pre-defined mesh as in the previous chapter. The details of the encoder and the decoders are described in the experiment section.

View comparison function. Color images (RGB channels) and silhouettes (alpha channels) are processed separately in $\mathscr{L}_v(\cdot,\cdot)$. Let x and $\hat{x} = P(R(x),v)$ be a ground truth view and an estimated view, x_c, $\hat{x}_c$ be the RGB channels of x, $\hat{x}$, and x_s,$\hat{x}_s$ be the alpha channels of x, $\hat{x}$. The silhouette at the i-th pixel x_{s_i} is set to one if an object exists at the pixel and to zero if the pixel is part of the background. x_s can take a value between zero and one owing to anti-aliasing of the renderer. To compare color images x_c,$\hat{x}_c$, we use perceptual loss $\mathscr{L}_p$ [35] with additional feature normalization. Let $F_m(\cdot)$ be the m-th feature map of N_f maps in a pre-trained CNN for image classification. In addition, let C_m, H_m, W_m be the channel size, height, and width of $F_m(\cdot)$, respectively. Specifically, we use the five feature maps after convolution layers of AlexNet [47] for $F_m(\cdot)$. Then, using $D_m = C_m H_m W_m$, the perceptual

[1]We modified the approximate differentiation of the renderer in the previous chapter. Details are described later.

loss is defined as

$$\mathscr{L}_p(\hat{x_c}, x_c) = \sum_{m=1}^{N_f} \frac{1}{D_m} \left| \frac{F_m(\hat{x_c})}{|F_m(\hat{x_c})|} - \frac{F_m(x_c)}{|F_m(x_c)|} \right|^2. \tag{5.2}$$

For silhouettes $x_s, \hat{x}_s$, we use their multi-scale cosine distance. Let x_s^i be an image obtained by down-sampling x_s 2^{i-1} times, and N_s be the number of scales. We define the loss function as

$$\mathscr{L}_s(x_s, \hat{x}_s) = \sum_{i=1}^{N_s} \left(1 - \frac{x_s^i \cdot \hat{x}_s^i}{|x_s^i||\hat{x}_s^i|} \right). \tag{5.3}$$

We also use negative intersection over union (IoU) of silhouettes, as was used in the previous chapter. Let $\odot$ be an elementwise product. This loss is defined as

$$\mathscr{L}_s(x_s, \hat{x}_s) = 1 - \frac{|x_s \odot \hat{x}_s|_1}{|x_s + \hat{x}_x - x_s \odot \hat{x}_s|_1}. \tag{5.4}$$

The total reconstruction loss is $\mathscr{L}_v = \mathscr{L}_s + \lambda_c \mathscr{L}_c$. λ_c is a hyper-parameter.

Training. We optimize $R(\cdot)$ using mini-batch gradient descent. Figure 5.1 shows the architecture of single-view training. In multi-view training, we randomly take two views of an object in one minibatch. The architecture for computing $\mathscr{L}_r$ in this case is shown in Figure 5.2.

5.2.2 View prior learning

As described in Chapter 1, in view-based training, a reconstructor can generate a shape that looks unrealistic from unobserved viewpoints. In order to reconstruct a shape that is viewed as realistic from any viewpoint, it is necessary to (1) learn the difference between correct views and incorrect views, and (2) tell the reconstructor how to modify incorrect views. In view-based training, reconstructed views from observed viewpoints converge to the real views in a training dataset by minimizing the reconstruction loss, and views from unobserved viewpoints do not always become correct. Therefore, the former can be regarded as correct and realistic views, and the latter can be regarded as incorrect and unrealistic views. Based on this assumption, we propose to train a discriminator that distinguishes estimated views at observed viewpoints from estimated views at unobserved viewpoints to learn the correctness of views. The discriminator can pass this knowledge to the reconstructor by back-propagating the gradient of the discrimination loss into the reconstructor via estimated views and shapes as with adversarial training in image generation [17] and domain adaptation [15].

Concretely, let $Dis(\cdot,\cdot)$ be a trainable discriminator that takes a view and its viewpoint and outputs the probability that the view is correct, and $\mathscr{V}$ be the set of all viewpoints in the training dataset. Using cross-entropy, we define *view disrcimination loss* as

$$\begin{aligned}\mathscr{L}_d(x_{ij}, v_{ij}) = &-\log(Dis(P(R(x_{ij}), v_{ij}), v_{ij})) \\ &- \sum_{v_u \in \mathscr{V}, v_u \neq v_{ij}} \frac{\log(1 - (Dis(P(R(x_{ij}), v_u), v_u)))}{|\mathscr{V}| - 1}. \end{aligned} \quad (5.5)$$

In minibatch training, we sample one random view for each reconstructed object to compute $\mathscr{L}_d$.

Stability of training. Although adversarial training is generally not stable, training of our proposed method is stable. It is known that training of GANs fails when a discriminator is too strong to be fooled by a generator. This problem is explained from the distinction of the supports of *real* and *fake* samples [2]. However, in our case, it is very difficult to distinguish views correctly in an earlier training stage because view reconstruction is not accurate and views are incorrect from any viewpoint. Even in a later stage, the reconstructor can easily fool the discriminator by slightly breaking correct views. Therefore, the discriminator cannot be dominant in our method.

Optimization of the reconstructor. The original procedure of adversarial training requires optimizing a discriminator and a generator iteratively [17]. Subsequently, Ganin *et al.* [15] proposed to train a generator using the reversed gradient of discrimination loss. The proposed gradient reversal layer does nothing in the forward pass, although it reverses the sign of gradients and scales them λ_d times in the backward pass. This layer is posed on the right before a discriminator. Because this optimization procedure is not iterative, the training time is shorter than iterative optimization. Furthermore, we experimentally found that the performances of the gradient reversal and iterative optimization are nearly the same in our problem. Therefore, we use the gradient reversal layer for training the reconstructor.

Image type for the discriminator. The discriminator can take both RGBA images and silhouette images. We give it RGBA images when texture prediction is conducted, otherwise we give it silhouettes.

Class conditioning. In addition, a discriminator can be conditioned on class labels using the conditional GAN [64] framework. When class labels are known, view discrimination

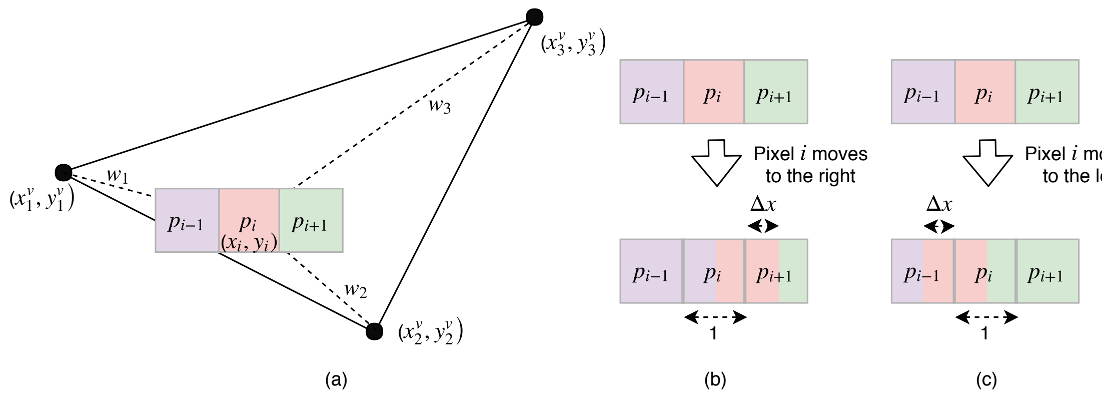


Figure 5.3 Our assumptions on the differentiation of a renderer.

becomes easier and the discriminator becomes more reliable. We use the projection discriminator [65] for class conditioning. Note that the test phase does not require class labels.

Another possible discriminator. We propose to train a discriminator on views of reconstructed shapes at observed and unobserved viewpoints. Another possible approach is to distinguish reconstructed views from real views in a training dataset. In fact, this discriminator does not work well because generating a view that is difficult to distinguish from real views is very difficult. This is caused by the limitation of the representation ability of the reconstructor and renderer. Table 2.3 shows a summary of the discriminators we have explained thus far.

5.2.3 Internal pressure

One of the most popular methods in multi-view 3D reconstruction is space carving [50]. In space carving, a point inside all silhouettes is assumed to be inside the object. In other words, in terms of shape ambiguity, space carving produces the shape with the largest volume. Following this policy, we inflate the volume of estimated shapes by giving them internal pressure in order to maximize their volume. Concretely, we add a gradient along the normal of the face for each vertex of a triangle face. Let p_i be one of the vertices of a triangle face, and n be the normal of the face. We add a loss term $\mathscr{L}_p$ that satisfies $\frac{\partial \mathscr{L}_p(p_i)}{\partial p_i} = -n$.

5.2.4 Modification of neural mesh renderer

In our implementation, we compute the differentiation of a renderer in a different way from the previous chapter because we found that it is not stable when δ_i^x is very small. Furthermore, the computation time is significant because a very large number of pixels is involved in computing the gradient with respect to one pixel. The approximate differentiation described in this section solves both problems.

Suppose three pixels are aligned horizontally, as shown in Figure 5.3 (a). Their coordinates are (x_{i-1}, y_{i-1}), (x_i, y_i), and (x_{i+1}, y_{i+1}), and their colors are p_{i-1}, p_i, and p_{i+1}, respectively. Pixel i is located on a polygon, and its three vertices projected onto a 2D plane are (x_1^v, y_1^v), (x_2^v, y_2^v), and (x_3^v, y_3^v). Then (x_i, y_i) can be represented by their weighted sum $(x_i, y_i) = w_1(x_1^v, y_1^v) + w_2(x_2^v, y_2^v) + w_3(x_3^v, y_3^v)$. Let $\mathscr{L}$ be the loss function of the network. When the gradient with respect to a pixel $(\frac{\partial \mathscr{L}}{\partial x_i}, \frac{\partial \mathscr{L}}{\partial y_i})$ located at (x_i, y_i), is obtained, the gradient with respect to the vertices of the polygon $(\frac{\partial \mathscr{L}}{\partial x_1^v}, \frac{\partial \mathscr{L}}{\partial y_1^v})$, $(\frac{\partial \mathscr{L}}{\partial x_2^v}, \frac{\partial \mathscr{L}}{\partial y_2^v})$, $(\frac{\partial \mathscr{L}}{\partial x_3^v}, \frac{\partial \mathscr{L}}{\partial y_3^v})$ can be computed using w_1, w_2, w_3, and the chain rule.

We assume that when a pixel i moves to the right by Δx_i, the pixel color changes, as shown in Figure 5.3 (b). Concretely, the color of pixel i changes to $p_i + (p_{i-1} - p_i)\Delta x$ and the color of pixel $i+1$ changes to $p_{i+1} + (p_i - p_{i+1})\Delta x$. Then, $\frac{\partial p_i}{\partial x_i} = p_{i-1} - p_i$ and $\frac{\partial p_{i+1}}{\partial x_i} = p_i - p_{i+1}$. Let g_i be the gradient of the loss function back-propagated to pixel i, $g_i = \frac{\partial \mathscr{L}}{\partial p_i}$. Then, the gradient of x_i is

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial x_i} &= \frac{\partial \mathscr{L}}{\partial p_i}\frac{\partial p_i}{\partial x_i} + \frac{\partial \mathscr{L}}{\partial p_{i+1}}\frac{\partial p_{i+1}}{\partial x_i} \\ &= g_i(p_{i-1} - p_i) + g_{i+1}(p_i - p_{i+1}) \\ &= (g_i^p)^{\text{right}}. \end{aligned} \tag{5.6}$$

In the case where pixel i moves to the left, we can compute the gradient in a similar manner. Thus,

$$\begin{aligned}\frac{\partial \mathscr{L}}{\partial x_i} &= \frac{\partial \mathscr{L}}{\partial p_i}\frac{\partial p_i}{\partial x_i} + \frac{\partial \mathscr{L}}{\partial p_{i-1}}\frac{\partial p_{i-1}}{\partial x_i} \\ &= g_i(p_i - p_{i+1}) + g_{i-1}(p_{i-1} - p_i) \\ &= (g_i^p)^{\text{left}}. \end{aligned} \tag{5.7}$$

The problem is whether to use $(g_i^p)^{\text{right}}$ or $(g_i^p)^{\text{left}}$. When x_i moves to the right, the decrease in $\mathscr{L}$ is proportional to $(d)^{\text{right}} = -(g_i^p)^{\text{right}}$. When x_i moves to the left, the decrease in $\mathscr{L}$ is proportional to $(d)^{\text{left}} = (g_i^p)^{\text{left}}$. We define the gradient differently according to the following three cases.

- When $\max((d)^{\text{right}}, (d)^{\text{left}}) < 0$, the loss increases by moving the pixel i. Therefore, in this case, we define $\frac{\partial \mathscr{L}}{\partial x_i} = 0$.

- When $0 \leq \max((d)^{\text{right}}, (d)^{\text{left}})$ and $(d)^{\text{left}} < (d)^{\text{right}}$, the loss decreases more by moving pixel i to the right. In this case, we define $\frac{\partial \mathscr{L}}{\partial x_i} = (g_i^p)^{\text{right}}$.
- When $0 \leq \max((d)^{\text{right}}, (d)^{\text{left}})$ and $(d)^{\text{right}} < (d)^{\text{left}}$, it is better to move pixel i to the left. In this case, we define $\frac{\partial \mathscr{L}}{\partial x_i} = (g_i^p)^{\text{left}}$.

The gradient with respect to y_i is defined in a similar way.

5.2.5 Summary

In addition to using reconstruction loss $\mathscr{L}_r = \mathscr{L}_s + \lambda_c \mathscr{L}_c$, we propose to use view discrimination loss $\mathscr{L}_d$ to reconstruct realistic views and internal pressure loss $\mathscr{L}_p$ to inflate reconstructed shapes. The total loss is $\mathscr{L} = \mathscr{L}_s + \lambda_c \mathscr{L}_c + \mathscr{L}_d + \lambda_p \mathscr{L}_p$. The hyperparameters of loss weighting are λ_c, λ_p, and λ_d. Because λ_d is used in the gradient reversal layer, it does not appear in $\mathscr{L}$. The entire architecture is shown in Figure 5.1.

5.3 Experiments

We tested our proposed view prior learning (VPL) on synthetic and natural image datasets. We conducted an extensive evaluation of our proposed method using a synthetic dataset because it consists of a large number of objects with accurate silhouette and viewpoint annotations.

As a metric of the reconstruction accuracy, we used intersection over union (IoU) of a predicted shape and a ground truth that was used in many previous publications [9, 13, 36, 39, 85, 101, 107, 126]. To fairly compare our results with those in the literature, we computed IoU after converting a mesh into a volume of 32^3 voxels[2].

5.3.1 Experimental settings

Optimizer

We used the Adam optimizer [43] in all experiments. In our ShapeNet experiments, the Adam parameters were set to $\alpha = 4e-4, \beta_1 = 0.5, \beta_2 = 0.999$. In the PASCAL experiments, the parameters were set to $\alpha = 2e-5, \beta_1 = 0.5, \beta_2 = 0.999$. The batch size is set to 64 in our ShapeNet experiments, and set to 16 in our PASCAL experiments.

[2]Another popular metric is the chamfer distance of point clouds. However, this metric is not suitable for use in view-based learning. Because it commonly assumes that points are distributed on surfaces, it is influenced by invisible structures inside shapes, which are impossible to learn in view-based training. This problem does not arise when using IoU because it commonly assumes that the interior of a shape is filled.

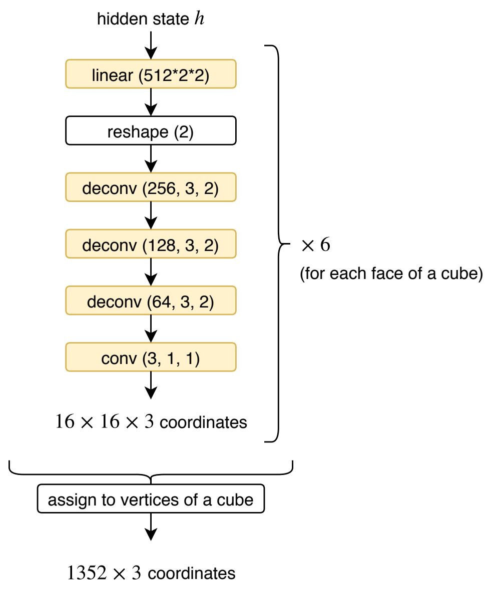


Figure 5.4 Architecture of the shape decoder used in the ShapeNet experiments. The 16×16 vertices on each face of the cube are separately generated, and they are merged into 1352 vertices. The dimension of the input vector is 512. All linear and deconvolution layers except the last one are followed by ReLU nonlinearity.

Encoder, decoder and discriminator

We used the ResNet-18 architecture [24] for the encoders in all experiments. The weights of the encoder were randomly initialized in the ShapeNet experiments. The weights were initialized using the weights of the pre-trained model from [24] in the PASCAL experiments.

We generated a 3D shape and texture image by deforming a pre-defined cube. The number of vertices on each face of the cube is 16×16, and the vertices on the edge of the cube are shared within two faces. The total number of vertices is 1352. The size of a texture image on each face is 64×64 pixels. The shape decoder outputs the coordinates of the vertices of this cube, and the texture decoder outputs six texture images. Figures 5.4 and 5.5 show the architecture of the shape decoders used in the ShapeNet and PASCAL experiments. Figure 5.6 shows the architecture of the texture decoder used in all experiments. Figures 5.7 and 5.8 show the architectures of the discriminators in the ShapeNet and PASCAL experiments. The layers used in the architecture figures are as follows:

- linear(a) is an affine transformation layer. a is the number of feature maps.
- conv(a,b,c) is a 2D convolution layer. The number of feature maps is a, the kernel size is $b \times b$, and the stride size is $c \times c$.

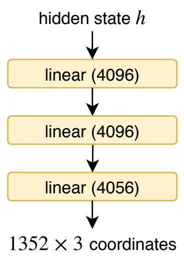


Figure 5.5 Architecture of the shape decoder used in the PASCAL experiments. The dimension of the input vector is 512. All linear layers except the last one are followed by ReLU nonlinearity.

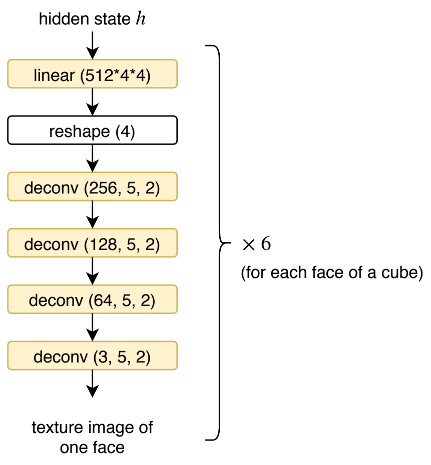


Figure 5.6 Architecture of the texture decoder used in all experiments. A texture image of size 64×64 is generated separately for each face of a cube. The input vector has 512 dimensions. All linear and deconvolution layers except the last one are followed by Batch Normalization [32] and ReLU nonlinearity.

- $\text{deconv}(a, b, c)$ is a 2D deconvolution layer. The number of feature maps is a, the kernel size is $b \times b$, and the stride size is $c \times c$.
- $\text{reshape}(a)$ reshapes a vector into feature maps of size $a \times a$.
- $\text{tile}(a)$ tiles a vector into feature maps of size $a \times a$.
- $\text{concat}(\cdot)$ stacks two feature maps.

Other hyperparameters

Table 5.1 and Table 5.2 show the number of training iteration and the weights of loss terms in ShapeNet and PASCAL experiments.

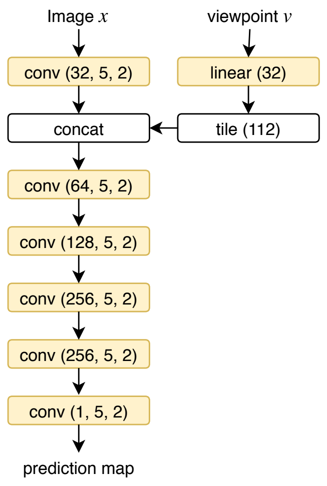


Figure 5.7 The architecture of the discriminator used in the ShapeNet experiments. The size of the input image is 224×224. A viewpoint is represented by a three-dimensional vector of the elevation, azimuth, and distance to the object. Spectral Normalization [66] is applied to all convolution and linear layers. All convolution layers except the last one are followed by LeakyReLU nonlinearity.

5.3.2 Synthetic dataset

As a synthetic dataset, we used ShapeNet [7], a large-scale dataset of 3D CAD models. We use $43,784$ objects in thirteen categories from ShapeNet. By using ShapeNet and a renderer, a dataset of views, silhouettes, viewpoints, and ground truth 3D shapes can be synthetically created. We used ground truth 3D shapes only for validation and testing. We used rendered views and train/val/test splits provided by Kar *et al.* [39]. In this dataset, each 3D model is rendered from twenty random viewpoints. Each image has a resolution of 224×224. We augmented the training images by random color channel flipping and horizontal flipping, as was used in [39, 85][3]. We use all or a subset of views for training, and all views were used for testing.

We used Batch Normalization [32] and Spectral Normalization [66] in the discriminator. The parameters were optimized with the Adam optimizer [43]. The hyperparameters were tuned using the validation set. We used Equation 5.3 as the view comparison function for silhouettes.

[3]When flipping images, we also flip the corresponding viewpoints.

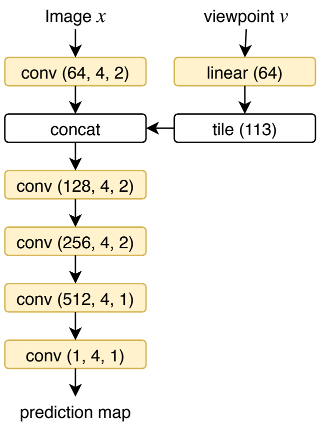


Figure 5.8 Architecture of the discriminator used in the PASCAL experiments. The size of the input image is 224×224. A viewpoint is represented by a 3×3 rotation matrix. All convolution layers except the last one are followed by LeakyReLU nonlinearity.

Training type	N_v	TP	VPL	#training iteration	λ_c	λ_d	λ_p
single-view	1			50000	-	-	0.0001
single-view	1	✓		50000	0.5	-	0.0001
single-view	1		✓	100000	-	0.2	0.0001
single-view	1	✓	✓	100000	0.5	2	0.0001
multi-view	$2,3,5,10,20$			$25000N_v$	-	-	0.0001
multi-view	$2,20$	✓		$25000N_v$	0.1	-	0.0001
multi-view	$2,3,5,10,20$		✓	$50000N_v$	-	0.03	0.0001
multi-view	$2,20$	✓	✓	$\min(50000N_v, 500000)$	0.1	0.3	0.0001

Table 5.1 Hyperparameters used in the ShapeNet experiments.

Single-view training

At first, we trained reconstructors in single-view training described in Section 5.2.1. Namely, we used only one randomly selected view out of twenty views for each object in training.

Figure 5.9 shows examples of reconstructed shapes with and without VPL. When viewed from the original viewpoints (b), the estimated shapes appear valid in all cases. However, without VPL, the shapes appear incorrect when viewed from other viewpoints (c–e). For example, the backrest of the chair is too thick, the car is completely broken, and the airplane has a strange prominence in the center. When VPL is used, the shapes look reasonable from any viewpoint. These results clearly indicate that the discriminator informed the reconstructor regarding knowledge of feasible views.

Learning with annotated single-view images

Training type	TP	VPL	#training iteration	λ_c	λ_d	λ_p
category-agnostic			15000	-	-	0.00003
category-agnostic	✓		15000	0.01	-	0.00003
category-agnostic		✓	50000	-	2	0.00003
category-agnostic	✓	✓	250000	0.01	0.5	0.00003
category-specific			5000	-	-	0.00003
category-specific	✓		5000	0.01	-	0.00003
category-specific		✓	40000	-	2	0.00003
category-specific	✓	✓	80000	0.01	0.5	0.00003

Table 5.2 Hyperparameters used in the PASCAL experiments.

Table 5.3 shows a quantitative evaluation of single-view training. VPL provides significantly improved reconstruction performance. This improvement is further boosted when the discriminator is class conditioned. We can tell that conducting texture prediction also helps training accurate reconstructors.

VPL is particularly effective with the *phone*, *display*, *bench*, and *sofa* categories. In contrast, VPL is not effective with the *lamp* category. In the case of *phone* and *display* categories, because the silhouettes are very simple, the shapes are ambiguous and various shapes can fit into one view. Although integrating texture prediction reduces the ambiguity, VPL is much more effective. In the case of *bench* and *sofa* categories, learning their long shapes is difficult without considering several views. Because the shapes in the *lamp* category are diverse and the training dataset is relatively small, the discriminator cannot learn meaningful priors.

Multi-view training

Second, we trained reconstructors using multi-view training as described in Section 5.2.1. Namely, we used two or more views out of twenty views for each object in training.

Table 5.4 shows the relationship between the reconstruction accuracy and the number of views per object N_v used for training. Texture prediction was not conducted in this experiment, and the difference between the proposed method and the baseline is the use of VPL with class conditioning. Our proposed method outperforms the baseline in all cases, which indicates that VPL is also effective in multi-view training. The effect of VPL increases as N_v decreases, as expected. Figure 5.10 shows reconstructed shapes with texture prediction when $N_v = 2$. When VPL is used, the shape details become more accurate.

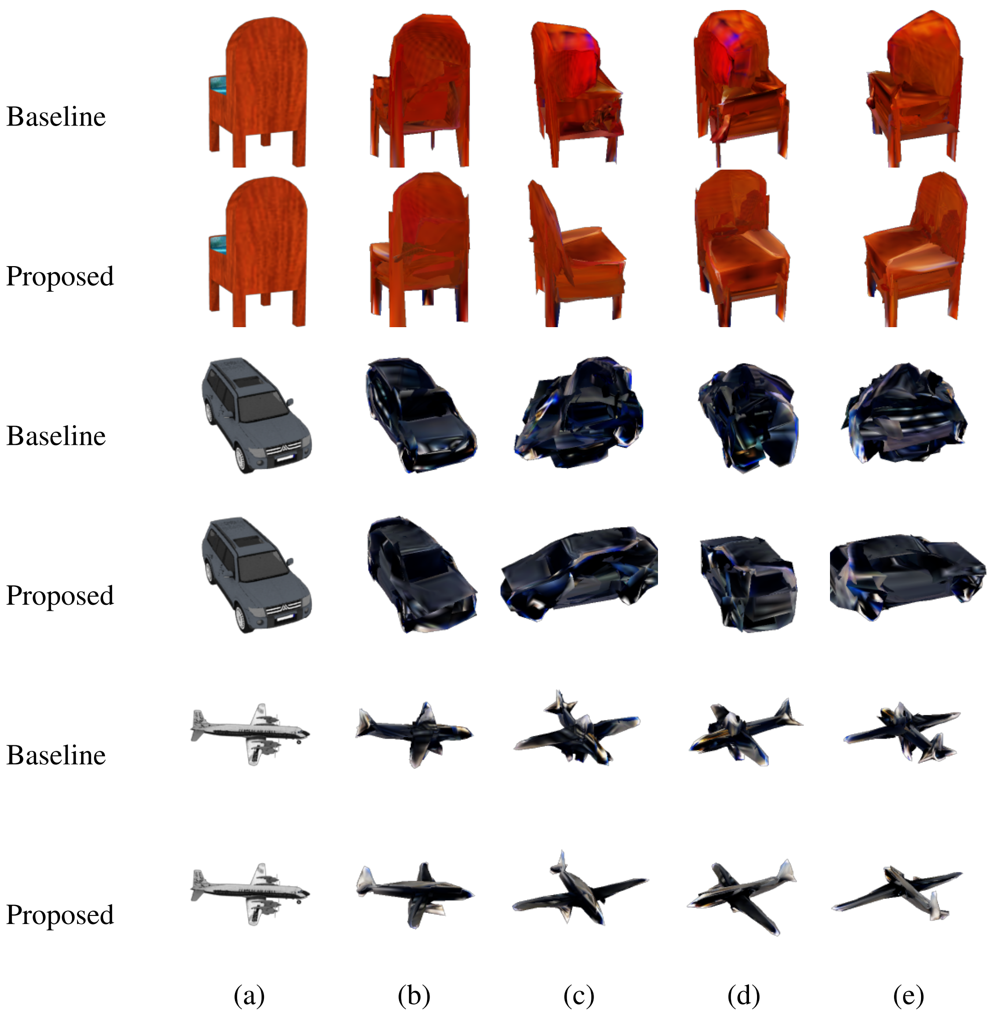


Figure 5.9 Examples of single-view training on the ShapeNet dataset. (a) Input images. (b) Reconstructed shapes viewed from the original viewpoints. (c–e) Reconstructed shapes viewed from other viewpoints.

Discriminator and optimization

We discussed two types of discriminators in the last paragraph of Section 5.2.2 and emphasized the importance of discriminating between estimated views rather than estimated views and real views. We validated this statement with an experiment. We ran experiments in single-view training using the discriminator of Table 2.3 (d). We also tested the iterative optimization used in GAN [17] instead of using a gradient reversal layer [15]. However, in both cases, we were unable to observe any meaningful improvements from the baseline by tuning λ_d. This fact indicates that the discriminator in Figure 2.3 (d) does not work well in practice, and discriminating estimated views is key to effective training.

VPL	CC	TP	airplane	bench	dresser	car	chair	display	lamp
			.479	.266	.466	.550	.367	.265	**.454**
✓			.500	.347	.583	.673	.413	.399	.443
✓	✓		.513	.376	**.591**	.701	.444	**.425**	.422
		✓	.483	.284	.544	.535	.356	.372	.443
✓		✓	.524	.378	.581	**.705**	.442	.422	.441
✓	✓	✓	**.531**	**.385**	**.591**	.701	**.454**	.423	.441

VPL	CC	TP	speaker	rifle	sofa	table	phone	vessel	all
			.524	.382	.367	.342	.337	.439	.403
✓			.578	.481	.464	.423	.583	.486	.490
✓	✓		**.596**	.479	.500	.436	.595	.485	.505
		✓	.534	.386	.370	.361	.529	.448	.434
✓		✓	.561	.510	.475	.443	**.625**	.490	.508
✓	✓	✓	.570	**.521**	**.508**	**.444**	.601	**.498**	**.513**

Table 5.3 IoU of single-view training on the ShapeNet dataset. VPL: proposed view prior learning. CC: class conditioning in the discriminator. TP: texture prediction.

Comparison with manually-designed priors

Our proposed internal pressure (IP) loss and some regularizers and constraints used in [36, 111] were designed using human knowledge regarding shapes. Table 5.5 shows a comparison with VPL. This experiment was conducted in single-view training without texture prediction.

This result shows that IP loss improves performance. The symmetricity constraint also improves the performance, however, some objects in ShapeNet are actually not symmetric. By regularizing the graph Laplacian and the edge length of meshes, although the visual quality of the generated meshes became better, improvement of IoU was not observed.

VPL cannot be compared with the learning-based 3D shape priors detailed by Gwak *et al.* [21] and Wu *et al.* [118] because these methods require additional 3D models for training, and their methods are applicable to voxels rather than meshes.

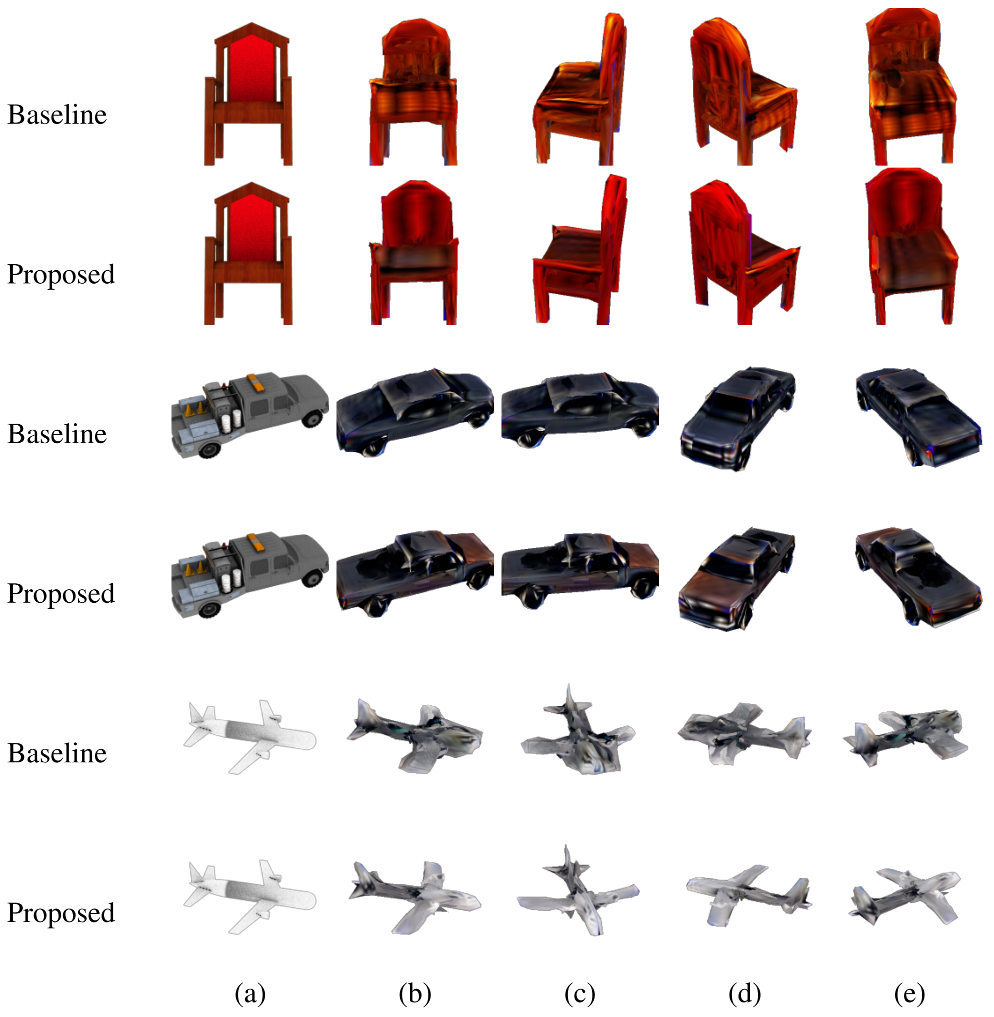


Figure 5.10 Examples of multi-view training on ShapeNet ($N_v = 2$). Panels (a–e) are the same as in Figure 5.9.

Comparison with state-of-the-arts

Our work also shows the effectiveness of view-based training. Table 5.6 shows the reconstruction accuracy (IoU) on the ShapeNet dataset by our method and recent papers[4].

Our method outperforms existing view-based training methods [126]. The main differences between our baseline and ours in the previous chapter 4 are the internal pressure and the training dataset. Because the resolution of our training images (224×224) is larger than

[4]The most commonly used dataset of ShapeNet for 3D reconstruction was provided by Choy *et al.* [9]. However, we found that this dataset is not suitable for view-based training because there are large occluded regions in the views owing to the narrow range of elevation in the viewpoints. Therefore, we used a dataset by Kar *et al.* [39], in which images were rendered from a variety of viewpoints. A comparison of the results from both datasets is not so unfair because the performance of 3D-R2N2 [9] is close in both datasets.

N_v	2	3	5	10	20
Baseline	.575	.596	.620	.641	.652
Proposed	**.583**	**.600**	**.624**	**.644**	**.655**

Table 5.4 The relation between the number of views per object N_v and the reconstruction accuracy (IoU) in multi-view training.

Prior	IoU
None	.387
Internal pressure (IP, ours)	.403
IP & Symmetricity [36]	.420
IP & Regularizing graph Laplacian [36, 111]	*.403**
IP & Regularizing edge length [111]	*.403**
IP & View prior learning (ours)	**.505**

Table 5.5 Comparison of our learning-based prior with manually-designed shape regularizers and constraints. *No meaningful improvement was observed.

theirs (64×64) and the elevation range in the viewpoints ($[-20°, 30°]$) is wider than that of theirs ($30°$ only), more accurate and detailed 3D shapes can be learned in our experiments.

It may be surprising that our view-based method outperforms reconstructors trained using 3D models. Although view-based training is currently less popular than 3D-based training, one can say that view-based training has much room for further study.

5.3.3 Natural image dataset

If a 3D model is available, we can synthetically create multiple views with accurate silhouette and viewpoint annotations. However, in practical applications, it is not always possible to obtain many 3D models, and datasets must be created using natural images. In this case, generally, multi-view training is not possible, and silhouette and viewpoint annotations are noisy. Therefore, to measure the practicality of a method, it is important to evaluate it in such a case.

Thus, we used the PASCAL dataset preprocessed by Tulsiani *et al.* [107]. This dataset is composed of images in PASCAL VOC [12], annotations of 3D models, silhouettes, and viewpoints in PASCAL 3D+ [123], and additional images in ImageNet [89] with silhouette and viewpoint annotations automatically created using [52]. We conducted single-view training because there is only one view per object. Because this dataset is not large, the variance in the training results is not negligible. Therefore, we report the mean accuracy from

	N_v	IoU
Single-view training		
Our best model$^\sharp$	1	**.513**
Multi-view training		
PTN [126]	24	.574
Ours (Chapter 4)	24	.602
Our best model$^\sharp$	20	**.655**
3D supervision		
3D-R2N2$^\sharp$ [39]	20	.551
3D-R2N2$^\flat$ [9]	24	.560
OGN$^\flat$ [101]	24	.596
LSM$^\sharp$ [39]	20	.615
Matryoshka$^\flat$ [85]	24	.635
PSGN$^\flat$ [13]	24	.640
VTN$^\flat$ [85]	24	**.641**

Table 5.6 Comparison of our method and state-of-the-art methods. Although supervision is weaker, our proposed method outperforms the other models trained using 3D models. $^{\sharp\flat}$Models denoted with the same symbol use the same rendered images.

five runs with different random seeds. We used the pre-trained ResNet-18 model [24] as the encoder as with [36, 107]. The parameters were optimized with the Adam optimizer [43]. We constrained estimated shapes to be symmetric, as with a previous study [36]. We used Equation 5.4 as the view comparison function for silhouettes.

Table 5.7 shows the reconstruction accuracy on the PASCAL dataset. Our proposed method consistently outperforms the baseline and provides state-of-the-art performance for this dataset, which validates the effectiveness of our proposed method. Category-specific models outperform category-agnostic models because the object shapes in these three categories are not very similar and multitask learning is not beneficial. The performance difference when texture prediction is used is primarily caused by the relative weight of the internal pressure loss.

Figure 5.11 shows typical improvements that can be gained using our method. Improvements are prominent on the wings of the airplane, the tires of the car, and the front legs of the chair when viewed from unobserved viewpoints.

In this experiment, internal pressure loss plays an important role because observed viewpoints are not diverse. Figure 5.12 shows a reconstructed shape without internal pressure. The trunk of the car is hollowed, and this hollow cannot be filled by VPL because there are few images taken from viewpoints such as (c–e) in the dataset.

	airplane	car	chair	mean
Category-agnostic models				
DRC [107]	.415	.666	.247	.443
Baseline (s)	.448	.652	.272	.458
Proposed (s)	.450	**.672**	.292	.471
Baseline	.440	.640	.280	.454
Proposed	**.460**	.662	**.296**	**.473**
Category-specific models				
CSDM [38]	.398	.600	.291	.429
CMR [36]	.46	.64	n/a	n/a
Baseline (s)	.449	.679	.289	.472
Proposed (s)	.472	**.689**	.303	**.488**
Baseline	.450	.669	.293	.470
Proposed	**.475**	.679	**.304**	.486

Table 5.7 IoU of single-view 3D reconstruction on the PASCAL dataset. The difference between the proposed method and the baseline is the use of view prior learning. (s) indicates silhouette only training without texture prediction ($\lambda_c = 0$).

5.3.4 Additional examples of single-view training

Figures 5.13, 5.14, and 5.15 show some reconstruction examples of *phone*, *display*, *bench*, *sofa*, and *lamp* categories on the ShapeNet dataset using single-view training. These figures correspond to the description in Section 5.3.2. The difference among categories and the use of texture prediction can be examined from these figures.

5.3.5 Performance of each category by multi-view training

In Section 5.3.2, only the average performance in all categories on the ShapeNet dataset has been reported. Table 5.8 shows reconstruction accuracy of each category.

5.3.6 Additional examples of multi-view training

Figures 5.16, 5.17, 5.18, and 5.19 show results from our best performing models using multi-view training ($N_v = 20$) for those interested in the state-of-the-art performance on the ShapeNet dataset. As can be seen in the figure, high-quality 3D models with textures can be reconstructed without using 3D models for training. The mean IoU of our method without texture prediction is 65.5, and the mean IoU of our method with texture prediction is 65.0. In contrast to single-view training, texture prediction does not improve the performance in multi-view training for large N_v.

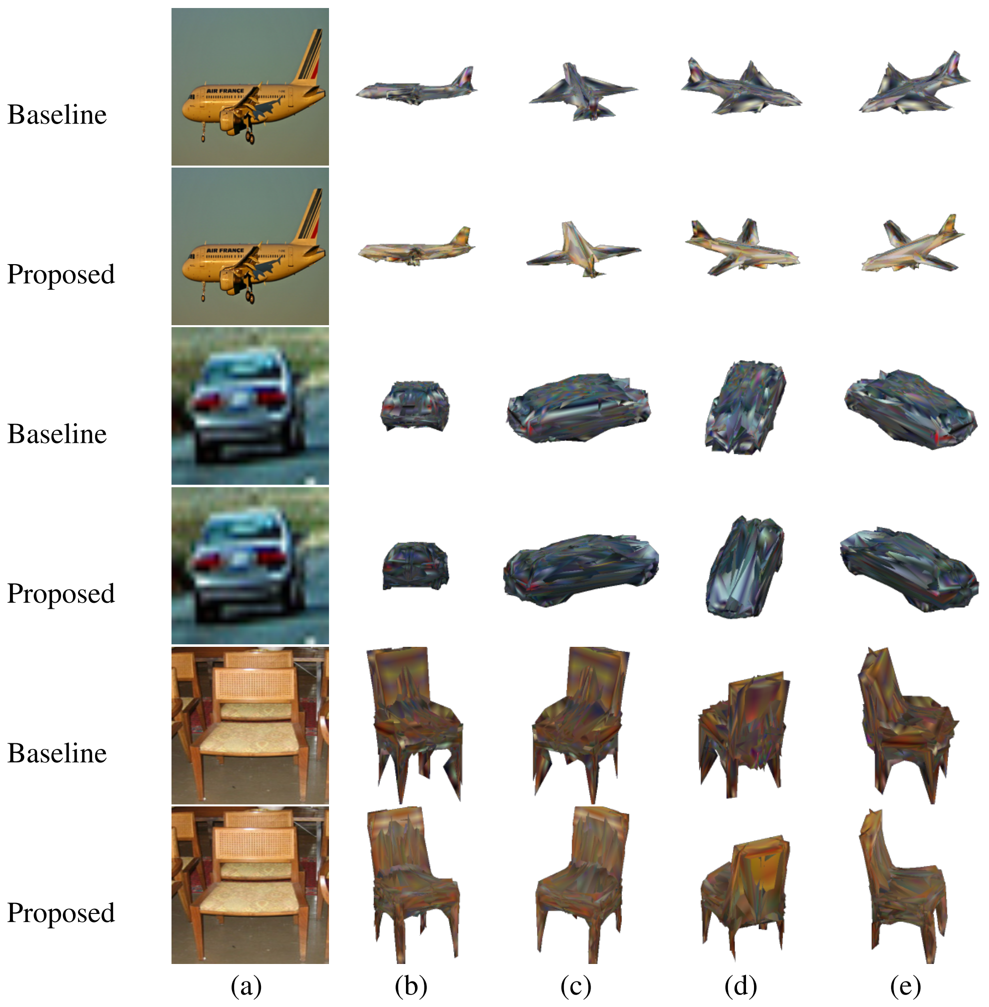


Figure 5.11 Examples on the PASCAL dataset. Panels (a–e) are the same as in Figure 5.9.

5.3.7 Discriminators and optimization

Table 5.9 shows the performances of the discriminators in Table 2.3 (c–d) in single-view training. The discriminator in Table 2.3 (d) does not work well in all cases. This table corresponds to the description in Section 5.3.2.

5.3.8 Internal pressure in multi-view training

In Section 5.3.2, we validated the effect of the internal pressure loss in single-view training. Table 5.10 shows that this loss is also effective in multi-view training. This experiment was conducted without texture prediction and view prior learning.

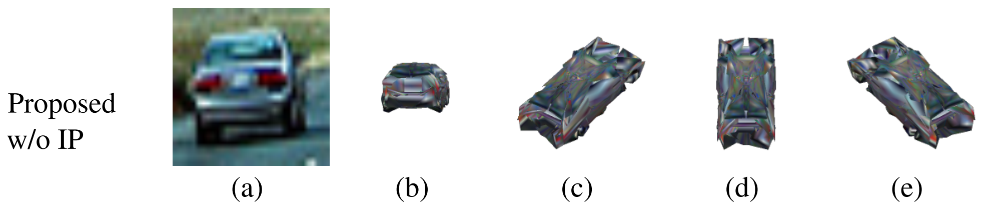


Figure 5.12 An example of reconstruction without internal pressure (IP). Panels (a–e) are the same as in Figure 5.9.

5.4 Summary

This chapter proposed a method to learn prior knowledge of views for view-based training of 3D object reconstruction. We verified our approach in single-view training on both synthetic and natural image datasets. We also found that our method is effective, even when multiple views are available for training. The key to our success involves using a discriminator with two estimated views from observed and unobserved viewpoints. Our data-driven method works better than existing manually-designed shape regularizers. We also showed that view-based training works as well as methods that use 3D models for training. The experimental results validate these statements.

Our proposed method, *view prior learning* (VPL), is particularly effective in cases where there is a high degree of ambiguity in the 3D shape due to the simplicity of the shape projected in 2D, such as *phone* and *display*. Our method is also effective for *bench* and *sofa* because the appearances of their long shapes vary with different viewpoints. These observations confirm our proposed method of increasing the validity of shapes from multiple viewpoints in addition to reducing image reconstruction error. We also found that the performance is improved by reconstructing textures as well as shapes, and that our method is effective not only for synthetic images but also for natural images.

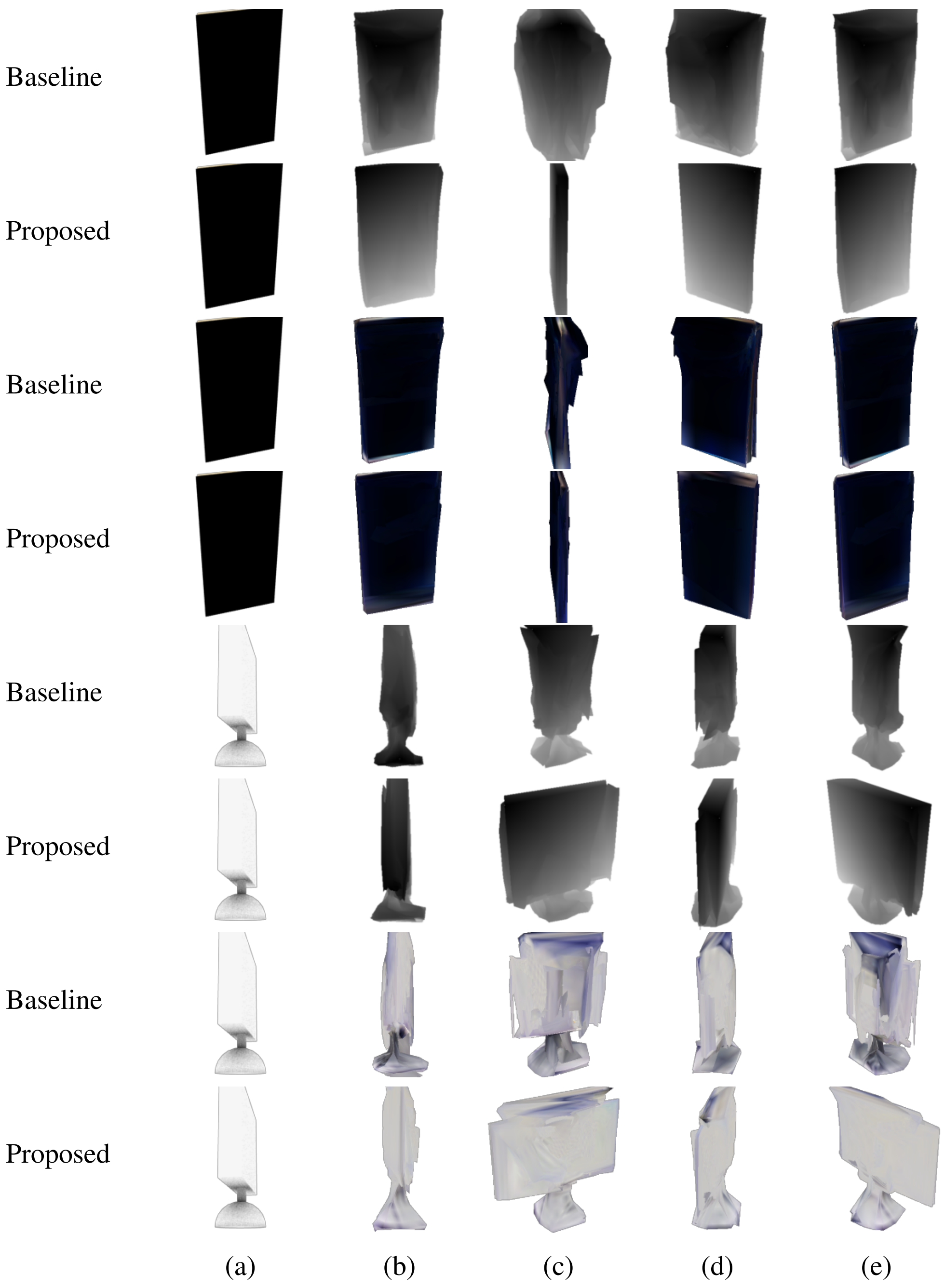


Figure 5.13 Examples on the ShapeNet dataset in single-view training. Panels (a–e) are the same as in Figure 5.9. This figure corresponds to Section 5.3.2.

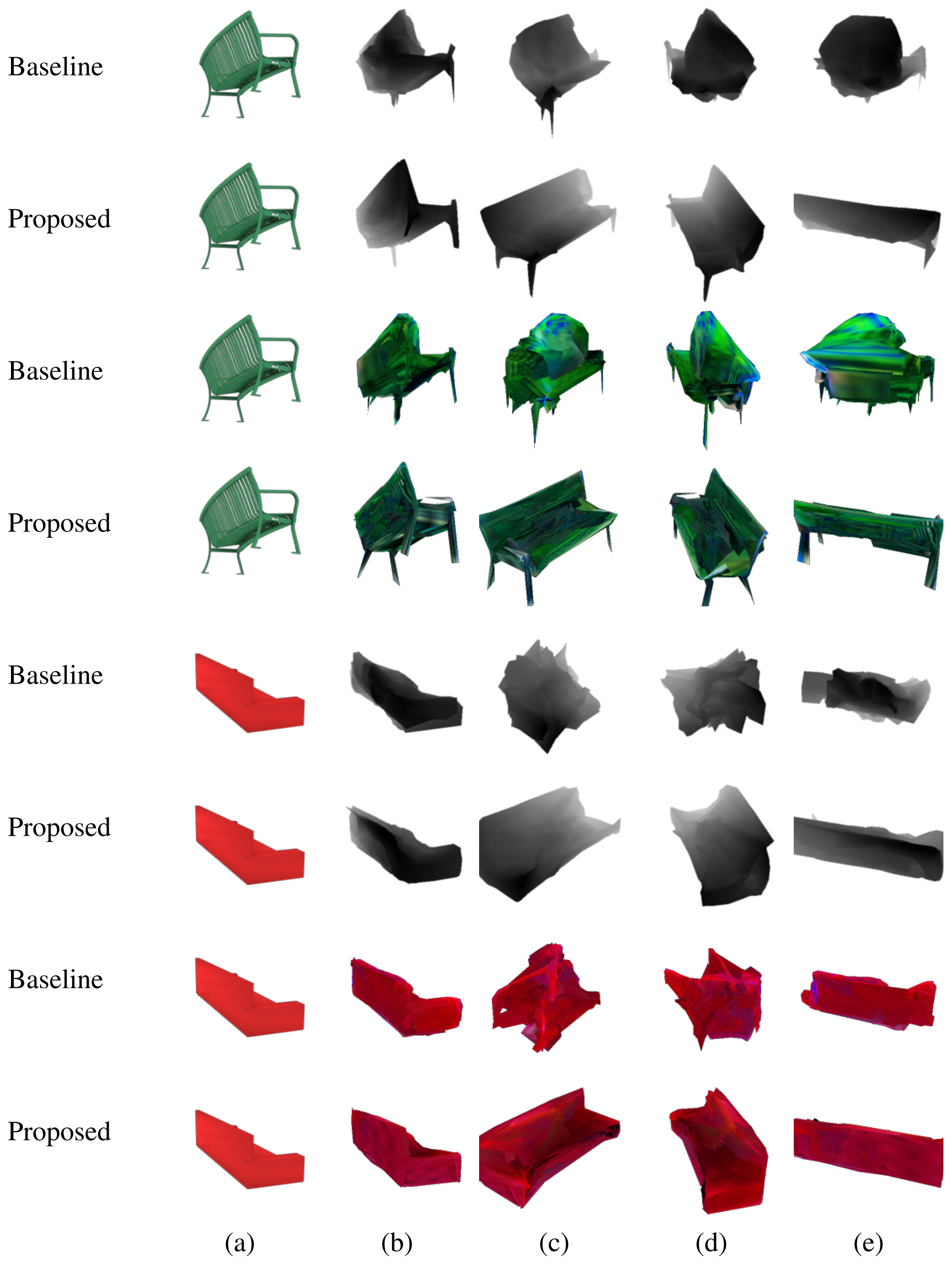


Figure 5.14 Examples on the ShapeNet dataset in single-view training. Notation of (a–e) is the same as in Figure 5.9. This figure corresponds to Section 5.3.2.

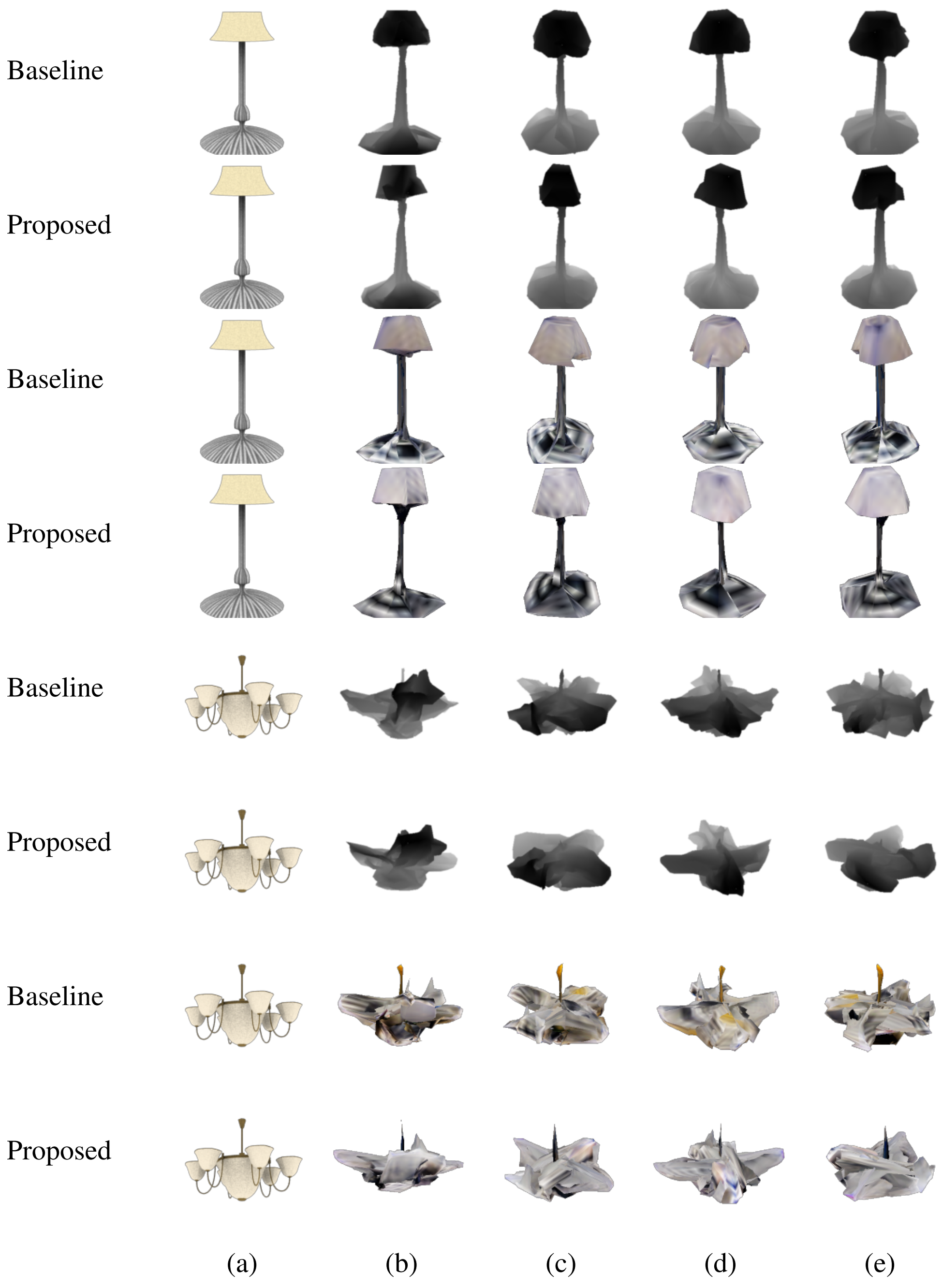


Figure 5.15 Examples on the ShapeNet dataset in single-view training. Notation of (a–e) is the same as in Figure 5.9. This figure corresponds to Section 5.3.2.

N_v	VPL	TP	airplane	bench	dresser	car	chair	display	lamp
2			.615	.467	.658	.767	.512	.465	.476
2	✓		.619	.476	.667	.770	.518	.467	.477
3			.631	.495	.673	.775	.537	.499	.490
3	✓		.638	.507	.674	.779	.543	.497	.491
5			.654	.530	.696	.787	.554	.539	.502
5	✓		.662	.542	.699	.792	.565	.533	.502
10			.673	.570	.712	.795	.582	.572	.510
10	✓		.681	.577	.718	.798	.584	.570	.508
20			.688	.593	.722	.799	.597	.599	.512
20	✓		.691	.598	.724	.802	.601	.597	.505
2		✓	.614	.469	.663	.768	.511	.475	.477
2	✓	✓	.618	.481	.666	.772	.522	.476	.480
20		✓	.685	.588	.723	.799	.597	.589	.508
20	✓	✓	.686	.589	.723	.803	.598	.592	.508

N_v	VPL	TP	speaker	rifle	sofa	table	phone	vessel	all
2			.631	.603	.588	.517	.622	.557	.575
2	✓		.631	.598	.590	.529	.687	.556	.583
3			.639	.624	.599	.535	.672	.573	.596
3	✓		.638	.625	.607	.552	.680	.574	.600
5			.657	.642	.623	.564	.721	.589	.620
5	✓		.656	.647	.629	.571	.725	.593	.624
10			.671	.660	.640	.585	.750	.606	.641
10	✓		.673	.661	.648	.593	.761	.606	.644
20			.678	.665	.651	.595	.766	.615	.652
20	✓		.680	.664	.656	.607	.775	.613	.655
2		✓	.624	.605	.582	.523	.649	.560	.579
2	✓	✓	.631	.607	.589	.529	.678	.557	.585
20		✓	.674	.663	.648	.603	.762	.615	.650
20	✓	✓	.676	.661	.651	.597	.759	.613	.650

Table 5.8 IoU of multi-view training on the ShapeNet dataset dataset. This table corresponds Section 5.3.2. VPL: proposed view prior learning. TP: texture prediction.

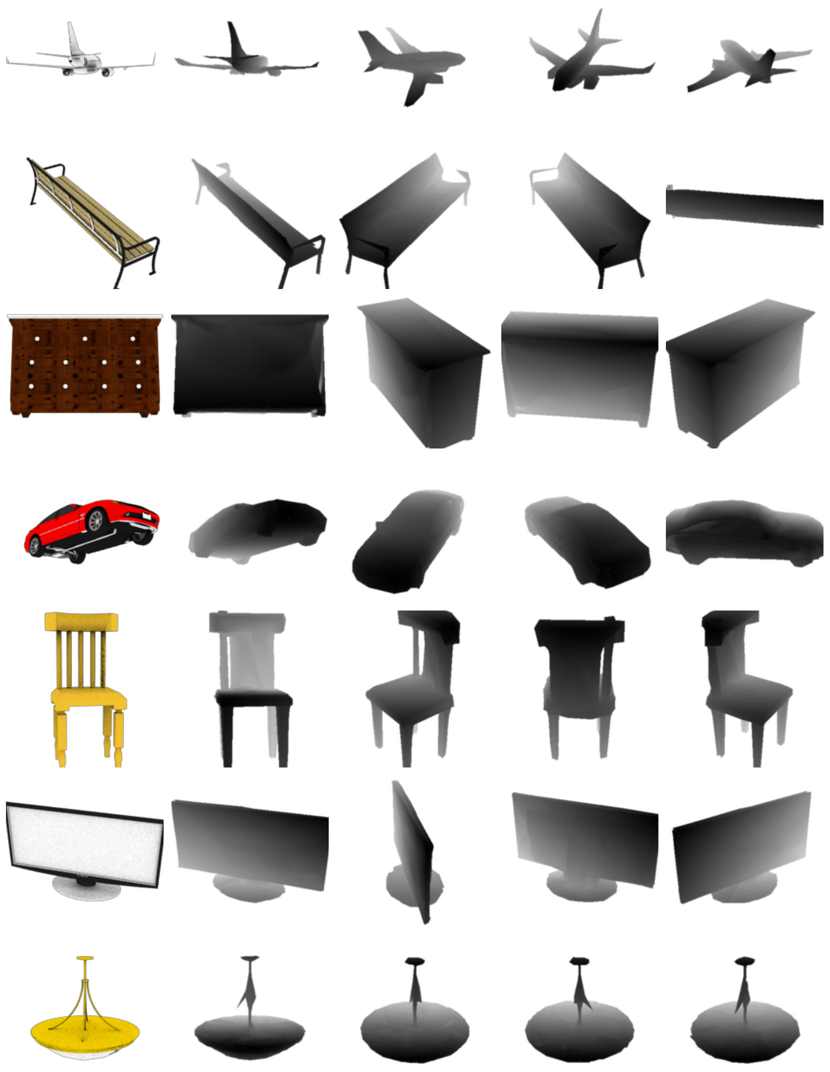


Figure 5.16 Examples of thirteen categories on the ShapeNet dataset by multi-view training ($N_v = 20$) without texture prediction. Panels (a–e) are the same as in Figure 5.9.

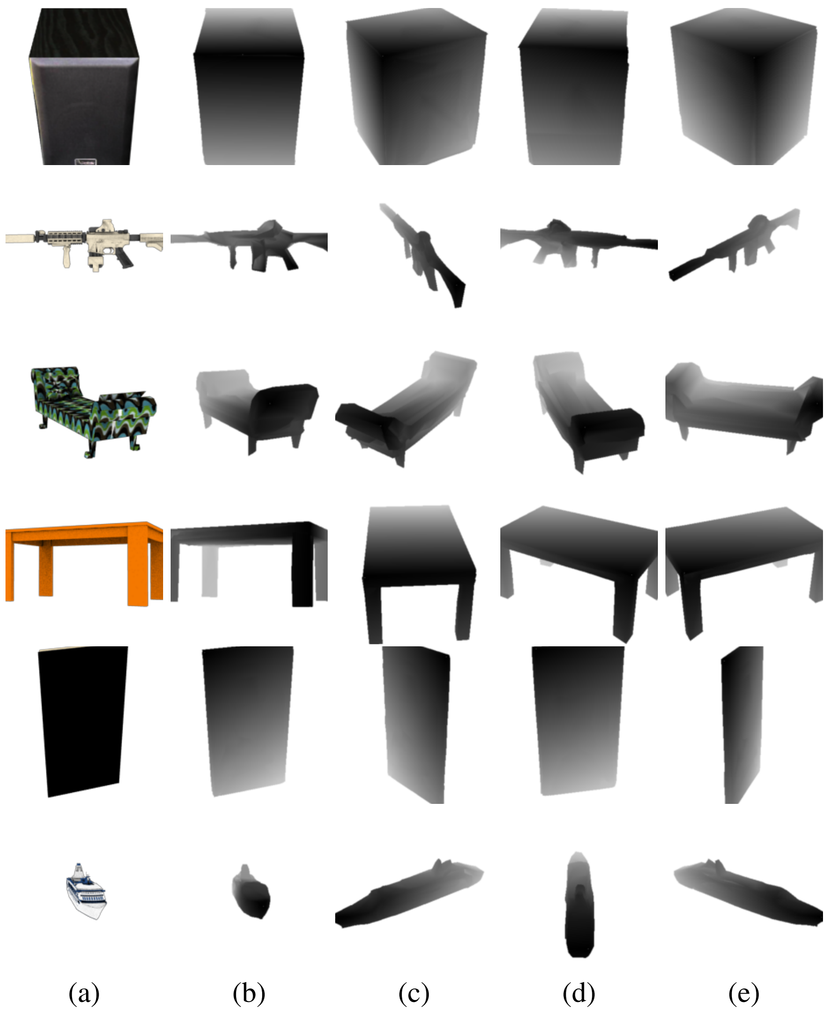


Figure 5.17 Examples of thirteen categories on the ShapeNet dataset by multi-view training (N_v = 20) without texture prediction. Panels (a–e) are the same as in Figure 5.9.

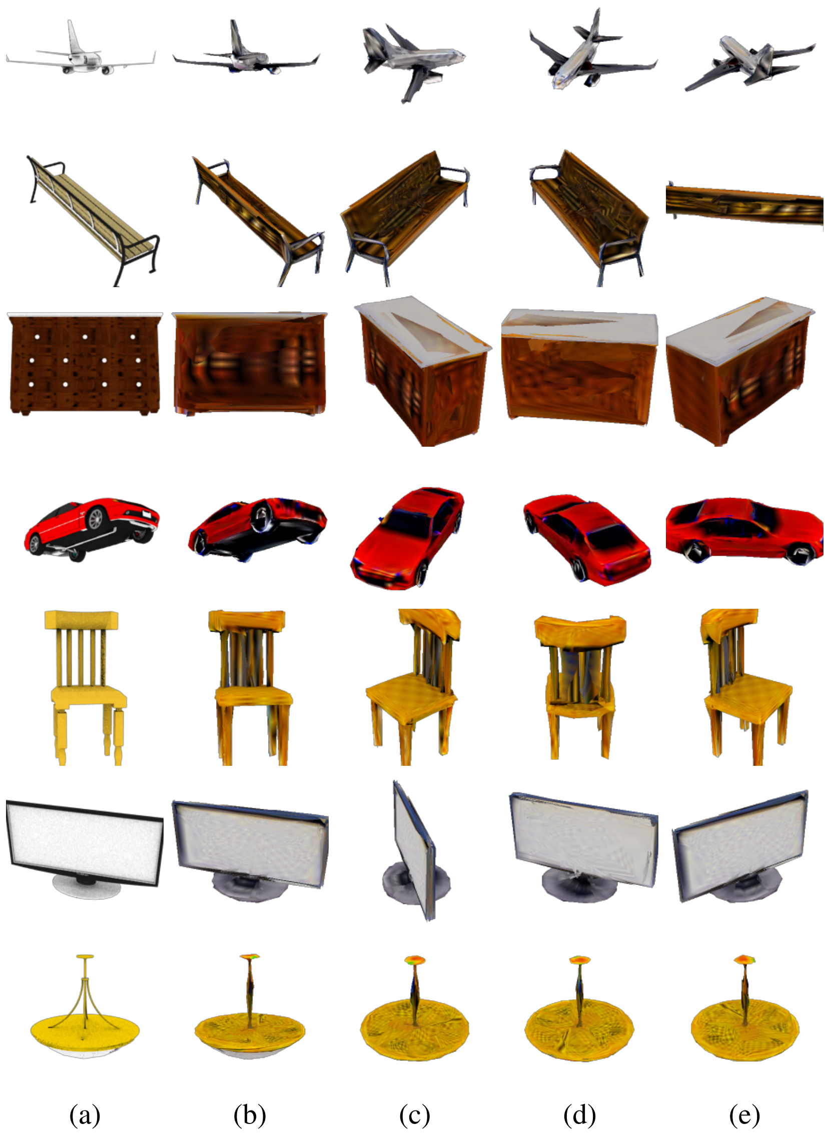


Figure 5.18 Examples of thirteen categories on the ShapeNet dataset by multi-view training (N_v = 20) with texture prediction. Panels (a–e) are the same as in Figure 5.9.

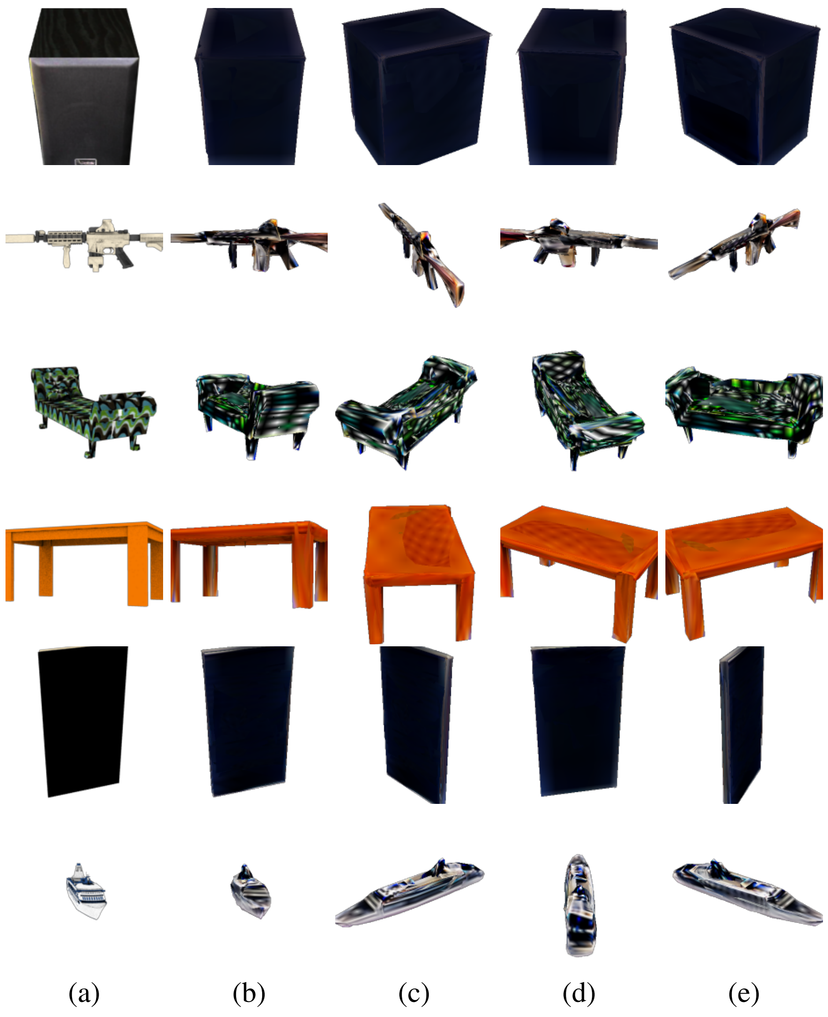


Figure 5.19 Examples of thirteen categories on the ShapeNet dataset by multi-view training ($N_v = 20$) with texture prediction. Panels (a–e) are the same as in Figure 5.9.

Discriminator	Optimization	Texture	IoU
None	-		.403
Table 2.3 (c)	Gradient reversal		.505
Table 2.3 (c)	Iterative		.514
Table 2.3 (d)	Gradient reversal		*.403**
Table 2.3 (d)	Iterative		*.403**
None	-	✓	.434
Table 2.3 (c)	Gradient reversal	✓	.513
Table 2.3 (c)	Iterative	✓	.510
Table 2.3 (d)	Gradient reversal	✓	*.434**
Table 2.3 (d)	Iterative	✓	*.434**

Table 5.9 Evaluation of the discriminators in Table 2.3 (c–d). *No meaningful improvement was observed by tuning λ_d.

Supervision	N_v	Internal pressure	IoU
Single-view	1		.387
Single-view	1	✓	.403
Multi-view	20		.648
Multi-view	20	✓	.652

Table 5.10 Effect of internal pressure loss.

Chapter 6

Learning with unannotated single-view images

6.1 Difficulty in learning from unannotated images

Learning 3D object reconstruction from unannotated images is quite inexpensive. The previous chapters assume that object silhouettes and camera parameters are given, which is relatively costly. This is because annotating segmentation is a time-consuming task, and annotating accurate camera parameters requires special skills. It is preferable to avoid these expensive tasks in learning single-view 3D object reconstruction of a large number of object categories.

In learning from images without any annotations, it is difficult to reconstruct images while separating objects from backgrounds. If the training is based only on image reconstruction, it results in reconstruction that does not distinguish between an object and background, as shown in Figure 1.7. In the previous chapters, such a problem does not arise because the silhouette of an object is assumed to be given as supervision.

In this chapter, to prevent background and texture predictors from simply copying an input image, we propose a method to first learn category-specific template shapes, and then to learn 3D object reconstruction using them. In the first step, we use a similar idea to the previous chapter. We optimize a template shape so that images of it from random viewpoints look like images in its category. In the second step, we restrict shapes to be reconstructed to the slight deformations of a template shape. Therefore, image reconstruction cannot be achieved by simply copying an input image. This method allows us to achieve 3D reconstruction that separates objects from backgrounds.

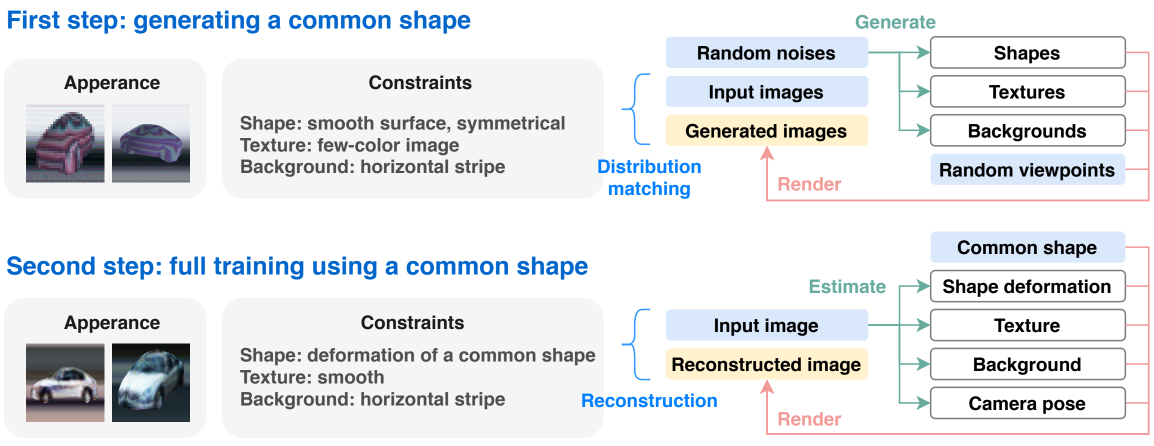


Figure 6.1 Training steps and constraints of our proposed method.

6.2 Method

We propose a method to train single-view 3D object reconstruction in Figure 1.6 with self-supervision while avoiding unrealistic solutions like these shown in Figure 1.7. The cause of the failures in Figure 1.7 is that copying an input image into texture or background is the easiest way to minimize the reconstruction error being unaware of shapes. Therefore, we propose to focus on reconstruction of shapes at first by limiting the representation capacity of textures and backgrounds to prevent from copying an input. Because image reconstruction is infeasible with this limited model, instead of training image reconstruction for each image, we aim to obtain a single category-specific common shape that is used in the following step. We also constrain a shape to be symmetric and make the surfaces smooth to avoid complicated shapes. In the second step, we train the reconstruction model. We limit reconstructed shapes to deformation of the common shape obtained in the first step not to copy an input into a texture image. Also, we limit the representation capacity of the background generator as with the first step. Figure 6.1 shows a summary of the training steps and Table 6.1 shows the introduced constrains. We describe the details of each step in the following sections.

6.2.1 Learning a category-specific common shape

The objective of this step is to obtain a single common shape in a category. To avoid poor solutions that do not separate geometry and appearance, we must introduce an additional constraint. How to design such a constraint has been discussed in intrinsic image decomposition [6]. Making reflectance constant where the spatial gradient of pixel intensity is low is one of the widely known assumptions [18, 29, 51]. However, some recent work suggests that

Category	Sub category	Prior	Step I	Step II
Geometry	Local	Shapes must be locally smooth	✓	
Geometry	Global	Shapes must be symmetric	✓	
Appearance	Object surface	Textures only contain a few colors	✓	
Appearance	Background	Background is horizontal stripes	✓	✓
Geometry	-	Shapes are deformations of a common shape		✓

Table 6.1 Designed priors used in this chapter.

restricting the number of colors is a promising assumption [5, 87, 95] by leveraging the fact that typical images contain only a few colors [74]. We also follow this direction. Besides, we limit background images to horizontal stripes not to copy an input. Horizontal stripes can express rough scene structures, such as the sky and grasses. However, they cannot express objects[1].

Reconstructing an image that looks close to an input image is infeasible by this model because of these strong limitations. Therefore, instead of reconstructing each image, we propose to train a generative model of images in a category by rendering generated 3D objects from various viewpoints. However, training a discriminator like GANs [17] is not feasible because a discriminator can use the introduced limitations to recognize generated images. Therefore, we use feature matching [91] and the Chamfer distance to match the distributions of training images and generated images.

Because the representation capacity of textures and backgrounds is limited, shapes try to become excessively complicated to represent edges in images. Therefore, we introduce local and global priors for shapes. The local prior is that shapes are locally smooth. This prior is first introduced in shape-from-shading [30] and has been demonstrated to be useful for stereo matching [114] and single-view 3D object reconstruction [5, 36, 111]. The global prior that shapes are symmetric, which is known to be useful for estimating occluded surfaces [105] and single-view reconstruction [36, 119]. Such a regularization is also popular in single-view 3D object reconstruction [36, 111].

The top-right part in Figure 6.1 shows the architecture of this step. The objective function is composed of the distribution matching loss $\mathscr{L}_m$ and a regularizer for shape smoothness $\mathscr{L}_s$. Though inputs are random noises, generated shapes converge to a single shape after training, which is similar to mode collapse in GANs. The details of each component in this step are as follows.

[1]We also tried total variation regularization and Gaussian blurring of textures and backgrounds in our preliminary experiment. However, we found that these limitations are not enough to prevent from copying.

Shape generation. We generate a shape by deforming vertices of a pre-defined sphere using a neural network that takes a random vector as input. Deforming sphere is a commonly used approach in single-view 3D object reconstruction [36, 111]. After a shape is generated, it is scaled to fit into a unit cube. Additionally, we manually set the initial dimensions of the sphere in each category. For the global prior, we make shapes symmetric by generating the left half of the shape by the neural network and creating the right half by flipping it. For the local geometry prior, we introduce a regularizer $\mathscr{L}_s$ that has several terms. One is to minimize graph Laplacian of a mesh, which represents approximated mean curvature at each vertex [103]. The other is angles between two neighboring triangle polygons, which implies smoothness at each edge. The details of these regularizers are described later.

Texture generation. We assume that the UV-mapping of a texture image and surfaces is pre-defined and fixed during training. We generate a texture image using a neural network that has DCGAN [83]-like architecture with residual connections [24]. Instead of generating a three-channel RGB image, we generate a N_c-channel image by a neural network and normalize each pixel, so that the sum of the channel values is one. Additionally, a color palette of N_c colors is generated by another neural network. Then, an RGB image is generated by mixing the N_c colors according to the N_c-channel image. Though the representation capability of this reparameterization is the same as the original network when $N_\mathrm{c} \geq 3$, it generates few-color images in practice.

Pose generation. We assume that and only azimuth and elevation of viewpoints can be changed, assuming that the camera is always directed to the center of the object and the distance to the object is fixed. Additionally, We assume the upward direction of the camera is also fixed. We randomly sample viewpoints from a manually-designed distribution without training. The distributions used in experiments are described later.

Background generation. Background images are constrained to horizontal stripes, as described above. We use a multi-layer neural network that takes a noise vector and outputs H colors to generate an image of height H.

Rendering. We render an image using a generated shape, pose, texture, and background with random directional lights. Using lights is essential to express shapes with few-color textures because shading is an essential cue for understanding shapes. We use a differentiable renderer proposed in the previous chapter to back-propagate the gradient from the loss function into generators.

Feature matching and Chamfer distance. We employ two metrics to make close the distribution of images in a dataset and generated images. The one is feature matching [91], and the other is Chamfer distance. Let N_b be the number of samples in a minibatch, and f_i^t and f_i^g be the image features of i-th training image and i-th randomly generated image in a minibatch, respectively. The feature matching loss represents the distance between the mean of both feature sets. It is defined as $\mathscr{L}_{\text{fm}} = \left| \frac{\sum_i f_i^t}{N_b} - \frac{\sum_i f_i^g}{N_b} \right|_1$. Regarding f_i^t and f_i^g as point sets, Chamfer distance between these sets are defined as $\mathscr{L}_{\text{c}} = \frac{1}{N_b} \left(\sum_i \min_j \left| f_i^t - f_j^g \right| + \sum_i \min_j \left| f_i^g - f_j^t \right| \right)$. We use a hyperparameter λ_m to balance them. The loss for distribution matching is $\mathscr{L}_m = (1 - \lambda_m)\mathscr{L}_{\text{fm}} + \lambda_m \mathscr{L}_{\text{c}}$.

Post-processing. The variance of polygon sizes should be smaller for proper texture mapping. However, a generated common shape may not satisfy it. Therefore, we generate another mesh using a generated mesh as supervision. The training objective is composed of the difference between silhouettes, the variance of polygon sizes, and the sum area of the surfaces. This post-processing significantly reduces the variance of polygon sizes while maintaining the whole shape.

6.2.2 Training of a full model using a common shape

In the second step, we train the model in Figure 1.6 while limiting generated shapes to deformations of a category-specific common shape. We use encoder-decoder architectures for shape, pose, texture, and background estimation. We do not employ the few-color constraint for textures because using common shapes avoid the fault in the left of Figure 1.7. Instead, we regularize the total variation [88] of textures to reduce noise. We use the same background generator in the previous step to prevent the fault in the right of Figure 1.7. We render images without using directional lights because textures can represent shadings in this step.

The bottom-right part in Figure 6.1 shows the architecture of this step. In this step, as described later, we explore better shapes and poses for each image by random perturbation and record them at each training iteration to avoid local minima. In addition to the components shown in the figure, we employ view prior learning (VPL) proposed in the previous chapter to stabilize training. Summarily, the loss function is composed of the following four terms. (1) Reconstruction loss. We compare input images with reconstructed images using the recorded shapes, estimated textures, the recorded poses, and estimated backgrounds. We also use feature matching to improve the reality of reconstructed images. (2) Mean absolute error between estimated shapes/poses and the recoded shapes/poses during training. (3) Total variation of estimated texture images. (4) VPL loss. To facilitate an early phase of training, at the *i*-th iteration, we select training samples randomly from first to *i*-th image in the dataset. This trick makes the model see the same image frequently in an early stage, which simplifies finding better poses and improve the accuracy of pose estimation.

The details of each component in this step are as follows.

Shape prediction. We deform a category-specific base shape using free-form deformation [94] similar to several other object reconstruction works [49, 132]. We use a spatial grid of $4 \times 4 \times 4$ vertices, and regress $4 \times 4 \times 4 \times 3$ variables that represent the difference between the original grid and a deformed grid by a neural network. Besides, we use another neural network to regress the relative height, width, and length of shapes. After deformation, we scale the estimated shapes to fit a unit cube.

We found in our preliminary experiments that the variation of generated shapes tends to be very small because exploring various shapes using only a differentiable renderer and gradient descent is difficult due to local minima. Therefore, we explore and record better shapes for each input image at each training iteration by random perturbation. Concretely, we render images using an estimated shape, a recorded shape, a recorded shape with random perturbation, and random shapes. Then, we compute the reconstruction loss of these images to find the best shape, and record it as the best shape for the image.

Pose prediction. In this step, we parameterize a 6DoF object/camera pose by the azimuth and the elevation of a viewpoint, the in-plane rotation of an object, the coordinates of the center point of the object in the image, and a scale of the object. We regress these six parameters by a neural network. We adopt the multiple regression method used in [31].

Similarly to shape estimation, we also need to explore better poses by random perturbations to avoid local minima. For each training iteration, for each image, we render images using an estimated pose, a recorded pose, a recorded pose with random perturbations, and random poses. We compute reconstruction loss and record a pose that gives minimum reconstruction error.

Photometric per-instance fine-tuning. In inference, we slightly adjust the outputs by neural networks by minimizing the reconstruction loss. Zuffi *et al.* [140] also employed a similar technique. This is possible because we can compute the loss without silhouette annotations. We successively optimize the estimated background, pose, and shape. We do not optimize texture to avoid copying the input.

Dataset	Category	Elevation	Azimuth
CIFAR-10, PASCAL	*car*	Uniform (0, 30)	$\text{Uniform}(0,360)$
CIFAR-10	*horse*	Uniform (-10, 10)	$\text{Beta}(1.5,1.5)*180$
PASCAL	*aeroplane*	Uniform (-60, 60)	$\text{Beta}(1.5,1.5)*180$
PASCAL	*chair*	Uniform (0, 30)	$\text{Uniform}(0,360)$

Table 6.2 Predefined distribution of viewpoints. When azimuth is zero, the camera is located in front of the object.

Dataset	Category	Width	Height	Length
CIFAR-10, PASCAL	*car*	0.5	0.5	1.0
CIFAR-10	*horse*	1.0	1.0	1.0
PASCAL	*aeroplane*	1.0	0.5	1.0
PASCAL	*chair*	1.0	1.0	1.0

Table 6.3 Initial dimensions in common shape learning.

6.3 Implementation details

6.3.1 Learning a category-specific common shape

We explain the details of the top part of Figure 6.1 in this section. The inputs are training images and the distributions of random noises and random viewpoints. We train neural networks for shape, texture, and background generation. The loss function has two terms. The one is to match distributions of training and generated images. The other is to regularize the surfaces of shapes.

Input image. In both CIFAR-10 and PASCAL experiments, we extract images that belong to the target category from the datasets for training. The range of pixels is normalized to $\{-1,1\}$. In the PASCAL experiments, we crop object regions using the provided bounding boxes so that the height and width of a bonding box is 80% of the image and resize them to 224×224. In the CIFAR-10 experiments, we take random crops from images padded by two pixels on each side for data augmentation.

Distributions of random noises and viewpoints. The random noises have 128 dimensions and follow the standard Gaussian distribution. Table 6.2 shows the distributions of viewpoints in each object category.

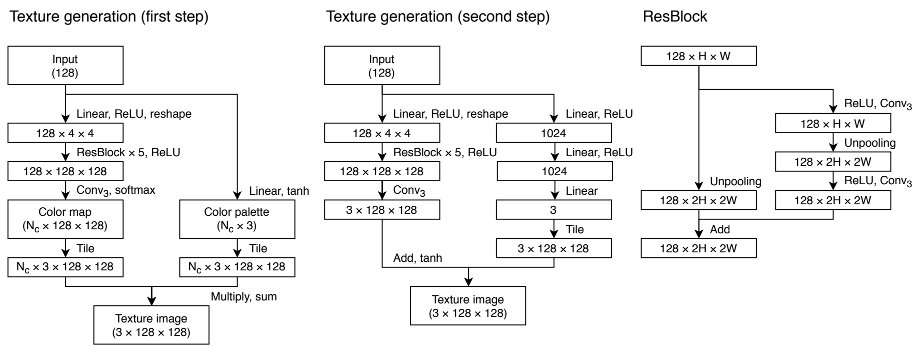


Figure 6.2 The architectures of our texture generation / estimation networks.

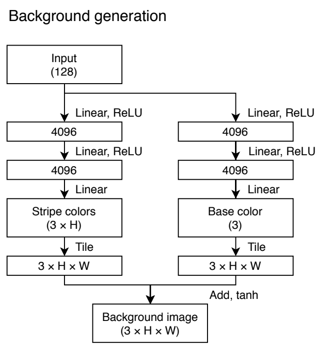


Figure 6.3 The architecture of our background generation / estimation network.

Shape generation. We move the vertices of an icosphere with 642 vertices to generate a shape. We manually give a rough aspect ratio of an object by setting the initial dimensions of the sphere. Table 6.3 shows the parameters. The shape generation network has three fully-connected layers that takes 128-dimensional vector and outputs 642×3 variables. We set the numbers of neurons in two hidden layers to 4096 and use ReLU activation function.

Texture generation. The left of Figure 6.2 illustrates the network architecture to generate a texture image that has N_c colors. We set the size of a texture image to 128×128.

Background generation. Figure 6.3 illustrates the network architecture of a background generator.

Distribution matching loss. The distribution matching loss $\mathscr{L}_m$ is composed of feature matching loss $\mathscr{L}_{\text{fm}}$ [91] and the Chamfer distance $\mathscr{L}_{\text{c}}$. Though we described $\mathscr{L}_{\text{fm}}$ and $\mathscr{L}_{\text{c}}$ in the case that image features f_i^t and f_i^g are vectors in Section 6.2.1, we actually use multi-scale feature maps in the CIFAR-10 and PASCAL experiments. We assume that the i-th image in a minibatch has L feature maps and the size of l-th feature map f_{il} is $H_l \times W_l$. With these notations, we define the distance of two feature maps f_i, f_j as

$$\mathscr{D}(f_i, f_j) = \sum_{l=1}^{L} \frac{1}{H_l W_l} \sum_{y=1}^{H_l} \sum_{x=1}^{W_l} \left| f_{ilxy} - f_{jlxy} \right|_1 . \tag{6.1}$$

Using this distance, we define the feature matching loss and Chamfer distance as

$$\mathscr{L}_{\text{fm}}(f^t, f^g) = \mathscr{D}\left(\sum_{i=1}^{N_b} \frac{f_i^t}{N_b}, \sum_{i=1}^{N_b} \frac{f_i^g}{N_b}\right) \tag{6.2}$$

$$\mathscr{L}_{\text{c}}(f^t, f^g) = \frac{1}{N_b}\left(\sum_{i=1}^{N_b} \min_i \mathscr{D}(f_i^t, f_i^g) + \sum_{i=1}^{N_b} \min_j \mathscr{D}(f_i^g, f_i^t)\right). \tag{6.3}$$

We introduce a hyper parameter λ_{m} to balance these two functions.

$$\mathscr{L}_{\text{m}} = \lambda_{\text{m}} \mathscr{L}_{\text{c}} + (1 - \lambda_{\text{m}}) \mathscr{L}_{\text{fm}}. \tag{6.4}$$

We set λ_{m} to $\{0.9, 0.3, 0.3, 0.9, 0.8\}$ for {CIFAR-10 *car*, CIFAR-10 *horse*, PASCAL *aeroplane*, PASCAL *car*, and PASCAL *chair*}. The feature maps need to have features that capture shapes of a target object to make the distribution matching loss meaningful. Therefore, the networks for feature extraction have to be trained to be aware of the target category. As feature maps, we use layers right before three sub-sampling operations of WRN-16-4 pre-trained using CIFAR-10 dataset in the CIFAR-10 experiments. In the PASCAL experiments, we use layers after the five convolution operations of AlexNet pre-trained using ImageNet[2].

Regularization of surfaces. We introduce a regularization term $\mathscr{L}_s$ to make smooth surfaces. Specifically, we minimize the graph Laplacian of a mesh, which represents approximated mean curvature at each vertex [103], and angles between two neighboring triangle polygons, which implies smoothness at each edge. For a mesh of N_v vertices, let $L \in \mathbb{R}^{N_v \times N_v}$ be a Laplacian matrix of vertices $v \in \mathbb{R}^{N_v \times 3}$ and let $t_l \in \mathbb{R}^+$ be a hyper parameter that controls tolerance. We minimize

$$\mathscr{L}_{g_1} = \max\left(\left(\sum_i \|L_i v\|_2^1\right) - t_l, 0\right)^2, \tag{6.5}$$

$$\mathscr{L}_{g_2} = \sum_i \|L_i v\|_2^2. \tag{6.6}$$

[2]ImageNet contains several object categories that relate to airplanes, cars, and chairs.

Similarly, let $\theta_{i,j}$ be the angle between normal vectors of two neighboring triangle polygons f_i and f_j and let $t_a \in \mathbb{R}^+$ be a hyper parameter, we minimize

$$\mathcal{L}_{a_1} = \max(\|\sum_{i,j} \theta_{i,j}\| - t_a, 0)^2. \tag{6.7}$$

$$\mathcal{L}_{a_2} = \sum_{i,j} \theta_{i,j}^2. \tag{6.8}$$

The regularization term $\mathcal{L}_s$ is a weighted sum of these terms using hyperparameters $\lambda_{g_1}, \lambda_{g_2}, \lambda_{a_1}, \lambda_{a_2}$.

$$\mathcal{L}_s = \lambda_{g_1}\mathcal{L}_{g_1} + \lambda_{g_2}\mathcal{L}_{g_2} + \lambda_{a_1}\mathcal{L}_{a_1} + \lambda_{a_2}\mathcal{L}_{a_2}. \tag{6.9}$$

The hyperparameters (λ_{g_1}, λ_{g_2}, λ_{a_1}, λ_{a_2}, t_l, t_a) are set to {(0, 0.3, 0.1, 0, 0, 2.0), (0.02, 0.2, 0.5, 0, 2.0, 5.0), (0, 0.3, 0, 0, 0, 0), (0, 0.3, 0.1, 0.1, 0.1, 2.0), (1.0, 0, 0.1, 0, 4.0, 3.0)} for {CIFAR-10 *car*, CIFAR-10 *horse*, PASCAL *aeroplane*, PASCAL *car*, PASCAL *chair*}.

Optimization. The loss function is the sum of $\mathcal{L}_m$ and $\mathcal{L}_s$. We used Adam optimizer with $\alpha = 0.0001$, $\beta_1 = 0.5$, and $\beta_2 = 0.99$ for optimization in all experiments. The batch size is set to 64, and the number of training iterations is set to $\{1200, 300, 200, 800, 200\}$ for {CIFAR-10 *car*, CIFAR-10 *horse*, PASCAL *aeroplane*, PASCAL *car*, PASCAL *chair*}, respectively.

6.3.2 Training of a full model using a common shape

The bottom-right part of Figure 6.1 illustrates the system in this step. The inputs are training images and a common shape of the category obtained in the previous step. We train neural networks for shape, texture, background, and camera pose prediction. We use encoder-decoder architectures for these networks. As encoders, we use WRN-16-4 in the CIFAR-10 experiments and ResNet-18 in the PASCAL experiments without pre-training. We set the dimension of embedding to 128.

Input image. We use the same images used in the previous step except for data augmentation. We do not use data augmentation in the CIFAR-10 experiments. In the PASCAL experiments, we apply small random translations of images when we input them to the neural networks as with [107].

Shape estimation. To predict a shape from an image, we deform a common shape using free-form deformation [94]. At first, we define a $4 \times 4 \times 4$ grid that envelops the common shape. Then, we regress the displacements of $4 \times 4 \times 4$ grid points by neural networks. Let NN^{g} be a three-layer neural network that takes a latent vector and outputs $4 \times 4 \times 4 \times 3$ variables that represent the displacements of each grid point. Besides, we use another network NN^{s} that outputs three variables that represent the scaling along the x-axis, y-axis, and z-axis. With these notations, we define the displacement of each grid point as

$$g_{xyz} = \mathrm{NN}^{\mathrm{g}}(h)_{xyz}\mathrm{NN}^{\mathrm{s}}(h). \tag{6.10}$$

The difference between two shapes can be measured by

$$\mathscr{L}_{\mathrm{shape}}(g, \hat{g}) = \sum_{x,y,z} \left| g_{xyz} - \hat{g}_{xyz} \right|_1. \tag{6.11}$$

Texture estimation. The middle part of Figure 6.2 illustrates the network architecture for texture estimation. We do not use the few-color constraint for textures in this step.

Background estimation. We use the same background decoder used in the previous step.

Camera pose estimation. A camera/object pose has seven parameters in our implementation: an elevation and azimuth angle of a viewpoint, an in-plane rotation angle, the size and center point of the 2D bounding box of an object in an image. We use a three-layer neural network that outputs these parameters.

Loss functions. For i-th training image x_i^{in} in a minibatch, let $s_i^p, s_i^r, t_i, b_i, p_i^p, p_i^r$ be a shape predicted by neural networks, a shape recorded during training, a predicted texture image, a predicted background image, a predicted pose, and a recorded pose, respectively. We render the following four images.

$$x_i^{\mathrm{pred}} = \mathrm{render}(s_i^r, t_i, b_i, p_i^r). \tag{6.12}$$

$$x_i^{\mathrm{rec}} = \mathrm{render}(s_i^p, t_i, b_i, p_i^p). \tag{6.13}$$

$$x_i^{\mathrm{real}} = \mathrm{render}(s_i^r, t_i, b_i, p_i^r). \tag{6.14}$$

$$x_i^{\mathrm{fake}} = \mathrm{render}(s_i^r, t_j, b_i, p_i^r). \tag{6.15}$$

Also, we extract features from three images.

$$f_i^{\text{in}} = \text{extract}(x_i^{\text{in}}). \tag{6.16}$$

$$f_i^{\text{pred}} = \text{extract}(x_i^{\text{pred}}). \tag{6.17}$$

$$f_i^{\text{rec}} = \text{extract}(x_i^{\text{rec}}). \tag{6.18}$$

To measure the accuracy of image reconstruction, we define the reconstruction loss $\mathscr{L}_{\text{rec}}$ and feature matching loss $\mathscr{L}_{\text{fm}}$ as follows.

$$\mathscr{L}_{\text{rec}} = \frac{1}{N_b}\left(\sum_i \mathscr{D}(f_i^{\text{in}}, f_i^{\text{rec}}) + \mathscr{D}(f_i^{\text{in}}, f_i^{\text{pred}})\right). \tag{6.19}$$

$$\mathscr{L}_{\text{fm}} = \mathscr{D}(\sum_{i=1}^{N_b} \frac{f_i^{\text{in}}}{N_b}, \sum_{i=1}^{N_b} \frac{f_i^{\text{rec}}}{N_b}). \tag{6.20}$$

We use the total variation loss $\mathscr{L}_{\text{tv}}$ for denoising texture images. Let t_{iuv} be the color at uv coordinates of i-th texture image. We define the loss as

$$\begin{aligned}\mathscr{L}_{\text{tv}} = \frac{1}{N_b}\sum_i \Big(&\frac{1}{(H_t-1)W_t}\sum_{u=1}^{H_t-1}\sum_{v=1}^{W_t} \left|t_{iuv} - t_{i(u+1)v}\right| + \\ &\frac{1}{H_t(W_t-1)}\sum_{u=1}^{H_t}\sum_{v=1}^{W_t-1} \left|t_{iuv} - t_{iu(v+1)}\right|\Big).\end{aligned} \tag{6.21}$$

In addition, to make the shape and pose prediction close to the shapes and poses explored and recorded by random exploration during training iterations, we use the following loss functions.

$$\mathscr{L}_{\text{r}} = \frac{1}{N_b}\sum_i \left(\mathscr{L}_{\text{shape}}(s_i^p, s_i^r) + \mathscr{L}_{\text{pose}}(p_i^p, p_i^r)\right). \tag{6.22}$$

Also, we use a loss function of view prior learning proposed in the previous chapter. Using a discriminator, the loss is defined as

$$\mathscr{L}_{\text{v}} = \frac{1}{N_b}\sum_i -\log(\text{dis}(x_i^r)) - \log(1-\text{dis}(x_i^f)). \tag{6.23}$$

The loss function to be minimized is

$$\mathscr{L} = \mathscr{L}_{\text{rec}} + \lambda_{\text{fm}}\mathscr{L}_{\text{fm}} + \lambda_{\text{tv}}\mathscr{L}_{\text{tv}} + \mathscr{L}_{\text{r}} + \lambda_{\text{v}}\mathscr{L}_{\text{v}}. \tag{6.24}$$

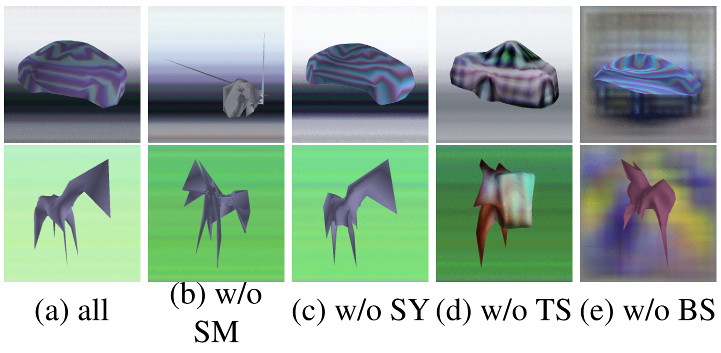


Figure 6.4 Category-specific common shapes on CIFAR-10 dataset generated (a) with all proposed constrains, (b) without the smoothness regularization, (c) without the symmetricity constraint (d) without the texture simplicity constraint, and (e) without the background simplicity constraint. These images are rendered in 256×256 resolution with upsampled background images.

We set $\{\lambda_{\text{fm}}, \lambda_{\text{tv}}, \lambda_{\text{v}}\}$ to {(0.4, 0.2, 0.01), (0.4, 1, 1), (0.4, 0.1, 1), (0.4, 0.1, 1), (0.4, 0.1, 1)} for {CIFAR-10 *car*, CIFAR-10 *horse*, PASCAL *aeroplane*, PASCAL *car*, PASCAL *chair*}.

Optimization. We use Adam optimizer with the same parameter as the previous step for optimization. The number of iterations is set to 10000 in all categories.

6.4 Experiments

6.4.1 CIFAR-10

We mainly tested our method on the CIFAR-10 dataset [46] because it is composed of natural images and contains thousands of images per object category. We focused on *car* and *horse* classes among ten object categories, because *car* is an artificial and rigid object and one of the most commonly used categories on the synthetic ShapeNet dataset [7] and *horse* is a deformable natural object and it is not contained in ShapeNet. We trained WRN-16-4 [133] on the CIFAR-10 training set for feature extraction. We used three layers right before sub-sampling as feature maps.

Common shape learning

At first, we evaluate the first step described in Section 6.2.1. We set the number of colors of *car* texture to four, and that of *horse* to one. We trained 10 models for each category using different random seed and selected the best-looking one. Figure 6.4 (a) shows generated

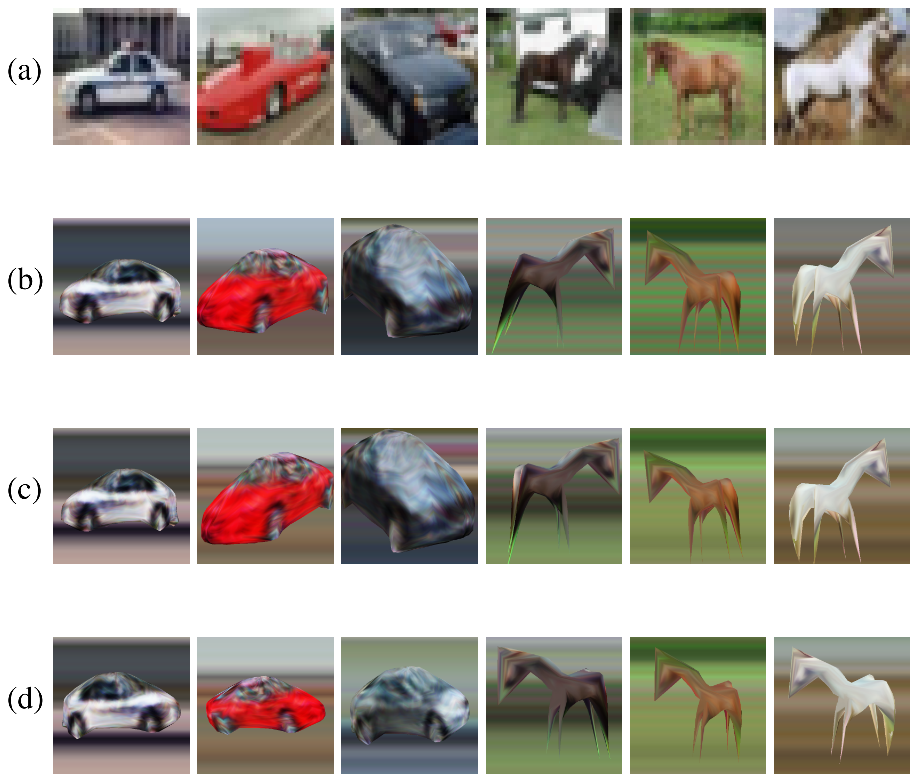


Figure 6.5 Representative results of 3D shape, pose, texture, and background estimation on CIFAR-10 test set. To understand the shapes and textures better, images are rendered at a higher resolution with upsampled backgrounds. Since the input images (a) are explicitly disentangled into 3D object elements, objects can be rendered from another viewpoint (d). Randomly selected results are in the appendix.

common shapes by our proposed method using different random noise. The generated shapes, textures, and backgrounds look plausible. Particularly, the *horse* correctly has four legs, and the *car* has four tires on the texture. The bright and dark regions in the background of *car* represent the sky and roads, and the background of *horse* shows grasses. This result indicates that the generators work properly.

Figure 6.4 (b–e) shows ablation study. (b) When the regularization of shape smoothness is removed, thin lines are generated to represent edges, which results in unrealistic shapes. (c) The shape symmetricity constraint seems unimportant for *car*, but it helps to generate legs on *horse* regularly. (d) Even when the texture simplicity constraint is removed, the texture does not represent the whole scene as in the left of Figure 1.7 because of constraints on shapes. However, the texture of *horse* contains the colors of horses and grasses, that results in the incorrect shape. (e) When the background simplicity constraint is not used, the

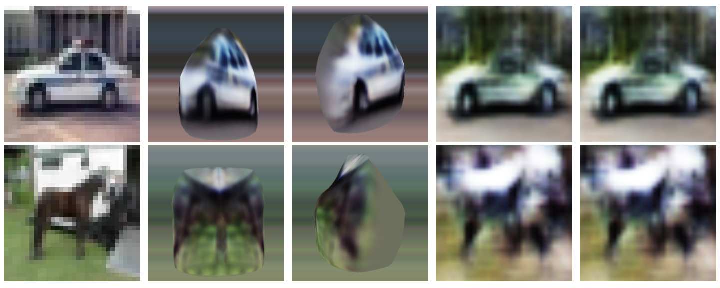


Figure 6.6 Training without our proposed two-stage training with and without proposed constraints (C). (b, d) is rendered using estimated viewpoints, and (c, e) is rendered using other viewpoints. (b–c) and (d–e) correspond to the left and right of Figure 1.7 respectively. These results confirm the importance of training shapes explicitly.

background generator tries to represent shapes, especially in *horse* . These results indicate introducing our prior knowledge about 3D scenes into a model is essential in self-supervised shape learning, and all of the proposed constraints and regularization are indispensable.

Full training using common shapes

Secondly, we evaluate the second step using the common shapes obtained in the previous step. Figure 6.5 shows representative results on the test set. (a–b) The result of reconstruction demonstrates that ours can reconstruct images that look similar to input images. (c) Estimated shapes, poses, and backgrounds can be further improved by photometric reconstruction loss and gradient descent. (d) Rendered images from other viewpoints show that these objects have correct 3D shapes, and the shapes are different according to the input images.

Effectiveness of two-stage training

One of the most important techniques of our method is to separate training into two stages. To validate its effectiveness, we trained our models in a single stage. Figure 6.6 shows the reconstruction results by the learned models. When all the constraints except for the texture simplicity are used, the textures represent edges of shapes, which results in incorrect shapes (b–c). When these constraints are not used, the reconstructed images look the same from any viewpoint because the background estimator copies input images (d–e). Apparently, neither model understands these 3D scenes correctly. These results correspond to the left and right of Figure 1.7 respectively.

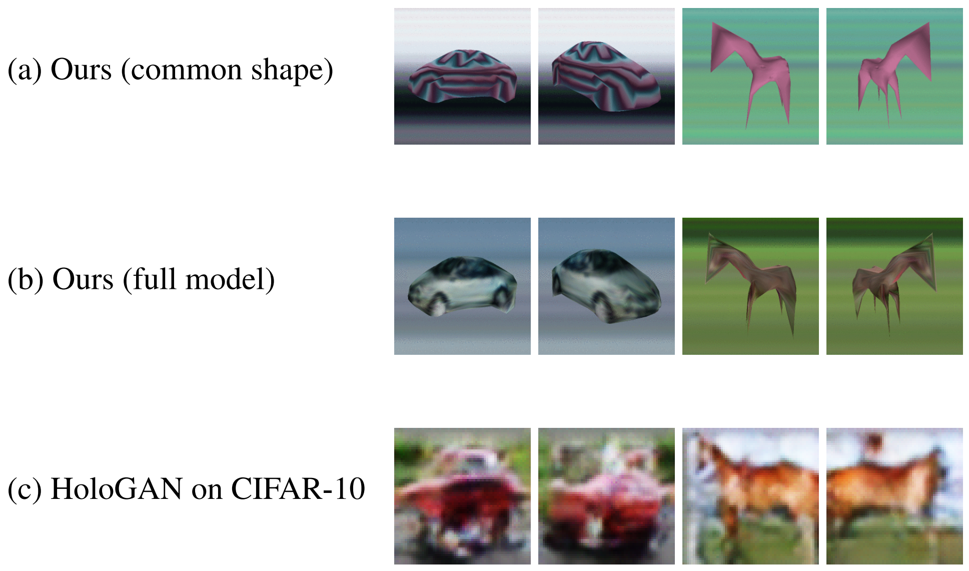


Figure 6.7 Comparison of ours with HoloGAN on CIFAR-10. Images at the same column are rendered by roughly the same viewpoints.

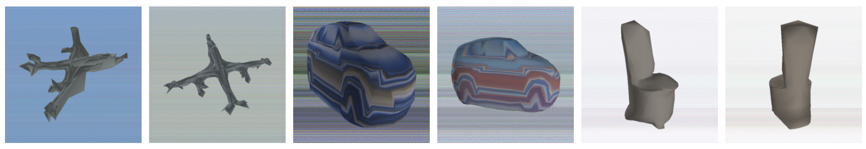


Figure 6.8 Generated common shapes of *aeroplane*, *car*, and *chair* on PASCAL dataset.

Comparison with existing works

To the best of our knowledge, this is the first work to learn single-view 3D object reconstruction from natural image collections without supervision. Therefore, we cannot conduct fair comparison between our work and existing works. One related approach would be structure-from-motion. Therefore, we tested COLMAP [92, 93] on CIFAR-10, however, it failed to reconstruct a shape because it cannot find initial corresponding image pairs. HoloGAN [71] learns a generative model of images with implicit but manipulable 3D representation from natural images though it is slightly different from 3D reconstruction. Figure 6.7 shows comparison of ours with HoloGAN trained on CIFAR-10. While the shapes produced by our method are consistent from multiple viewpoints, the shapes produced by HoloGAN are not. This result indicates the importance of having explicit 3D representations.

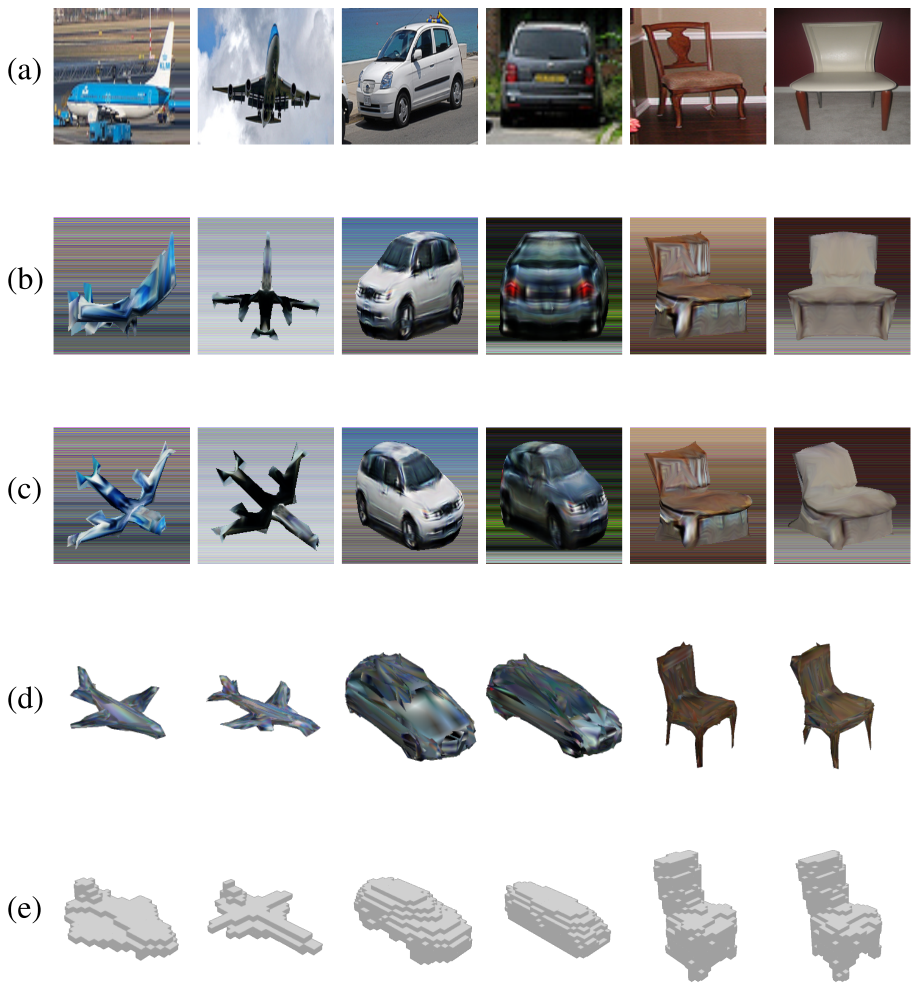


Figure 6.9 Representative results of 3D shape, pose, texture, and background estimation on PASCAL validation set, and comparison of them with results by other methods. (a) Input images. (b) Reconstructed images by our method. Rendered using estimated viewpoints. (c) Reconstructed images by our method using another viewpoint. (d) Shape and texture estimation in the previous Chapter. (e) Shape estimation by Tulsiani et al. [107]. Randomly sampled results of ours are in the appendix.

6.4.2 PASCAL

We also evaluated our method on PASCAL dataset preprocessed by Tulsiani *et al.* [107]. This dataset contains three object categories, *aeroplane*, *car*, and *chair*. It is composed of images from PASCAL VOC [12] and ImageNet [10]. Shape, silhouette, and pose annotations are provided for PASCAL images by PASCAL 3D+ [123]. Object regions were cropped using bounding boxes and resized to the same size. We do not use these annotations except

#training stages	Regularization	Shape			Viewpoint		
		aero	*car*	*chair*	*aero*	*car*	*chair*
single	w/o all	.245	.326	.180	.152	.167	.172
single	all	.114	.434	.150	.180	.487	.302
two	w/o smoothness	.271	.429	.223	.192	.646	.767
two	w/o symmetricity	.268	.442	.219	.200	.764	.535
two	w/o texture simplicity	.307	.534	.238	.231	.681	.786
two	w/o background simplicity	.297	.380	.217	.181	.502	.279
two	all	.302	.604	.235	.242	.781	.654

Table 6.4 Accuracy of shape and viewpoint estimation on PASCAL 3D+ dataset. These numbers indicate that (1) our proposed two-stage training significantly improves the accuracy of both tasks, and (2) all of the proposed four regularizations/constrains is effective on at least one object category.

Method	Silhouettes & viewpoints	*aero*	*car*	*chair*
CSDM [38]	annotated	.398	.600	.291
DRC [107]	annotated	.415	.666	.247
CMR [36]	annotated	.46	.64	n/a
Chapter 5	annotated	.472	.689	.303
Chapter 5	estimated by ours	.302	.604	.235

Table 6.5 Comparison of shape reconstruction accuracy with existing methods that require additional silhouette and viewpoint annotations for training. Though the accuracy of ours is lower than other recent works because of the difficulty of the task, the performance drop is not quite significant for *car* and *chairs*.

for evaluation. We used ResNet-18 architecture as encoders, and the same decoders as the CIFAR-10 experiments. We used five layers after the convolution operation of pre-trained AlexNet [47] as image features. Though this feature extractor requires additional supervision by ImageNet, this would be replaceable by any self-supervised representation learning methods. Since input images were cropped and resized using bounding boxes, we also cropped and resized rendered images. This reduces the degree of freedom of poses from six to three.

Figure 6.8 shows obtained category-specific common shapes by the first training step. Though they are not very beautiful, we can see that our method produces reasonable shapes on this dataset. Figure 6.9 shows several results of single-view reconstruction on the validation set by second training step. These results demonstrate that our proposed method works properly on PASCAL dataset. Though our shapes are rough and simple, the quality of ours is competitive with other recent methods because learning shapes from natural images is a very challenging task. Our estimated textures are more accurate than these in Chapter 5, and our estimated shapes are not very different from these by Tulsiani *et al*. [107]. Though the variation of our shapes is smaller than others, the variety of shapes by other methods does not always correctly reflect the input images (e.g. *aeroplane* in row (e)).

Because the images we used are resized to square using bounding boxes, our method cannot estimate an aspect ratio of an object correctly. Therefore, we cannot use annotated shapes for evaluation as is. Instead, we use the reconstruction accuracy of VPL with silhouettes and poses estimated by our method to evaluate the effectiveness of our proposed method quantitatively. We used foreground masks of reconstructed images as silhouette images. Additionally, we made the boundary of objects sharper using CRF [45]. Table 6.4 shows pose estimation accuracy on the training set and shape estimation accuracy on the validation set. Shape reconstruction accuracy is measured by intersection over union between ground truth and estimated shapes, and viewpoint estimation accuracy is measured by the $\frac{\pi}{6}$ score used in [106]. The significant difference in accuracy between single-stage training and our two-stage training indicates that our proposal to focus on shapes first is essential for this task. Introducing regularization in single-step training improves viewpoint estimation because it prevents shapes from shrinking. However, this does not help shape estimation. When smoothness regularization is not used in two-stage training, both shape estimation and viewpoint estimation accuracy decrease except for viewpoints of *chair*. This means that understanding poses is sometimes easier from sharp objects, however, it hurts the accuracy of silhouette and shape estimation. The simplicity of textures is effective only for *car* because separating foregrounds are relatively easy in other categories. The symmetricity constraint of shapes and simplicity constraint of backgrounds is confirmed to be effective in all cases. Table 6.5 provides quantitative comparison of our self-supervised shape reconstruction method with state-of-the-art methods on PASCAL dataset that use silhouette and viewpoint supervision. Though ours are not better as is expected, the performance gap between ours and other methods are not so huge except for *aeroplane*.

6.4.3 Discussion

Though we believe that this work is an important step toward 3D understanding with self-supervision, the accuracy of our method is not very high as the task is very challenging. In this section, we list our observations in the experiments and possible solutions. Please see the appendix for more qualitative results.

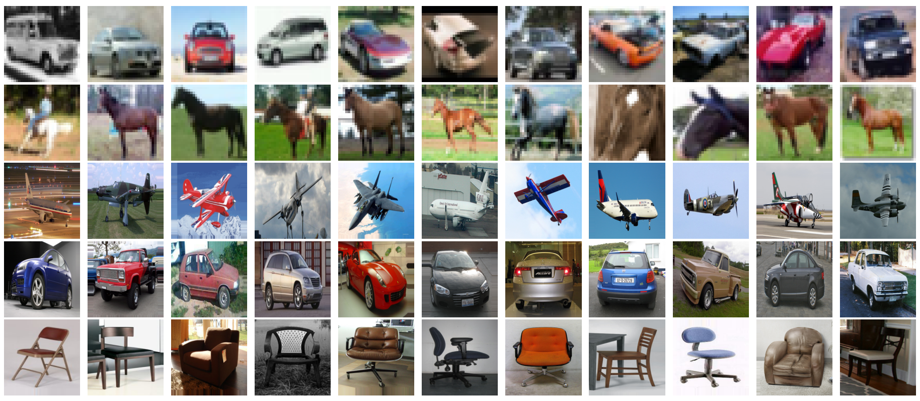


Figure 6.10 Randomly selected images in our training dataset. From top to bottom: CIFAR-10 *car*, CIFAR-10 *horse*, PASCAL *aeroplane*, PASCAL *car*, and PASCAL *chair*.

- Our method is not very good at reconstruction of small details (cf. legs of *horses*) and estimation of poses in ambiguity (cf. *aeroplane*). Also, the intra-class variance of generated shapes is small. One reason is that the reconstruction loss using pre-trained image features does not always capture category-specific fine-grained features. A reliable measure of image reconstruction is quite essential in self-supervised learning based on render-and-compare loss because reconstruction loss is the only supervision. Incorporating unsupervised learning of keypoints [99] would alleviate this problem.
- We introduced three assumptions in the introduction section. Though they hold in *car*, *horse*, *chair*, and *aeroplane* categories, our proposed method does not work well in *cat* and *dog* on CIFAR-10 because the deformation of shapes is relatively large. One possible way to learn large deformation would be using videos for training.
- We used manually-designed pose distributions in the common shape learning. However, designing them is not straightforward in some categories. For example, the viewpoint distribution of *horse* images are far from uniform because there are many photos of zoom up of heads, but fewer photos of tails. How to deal with biased distributions would be an important and interesting problem.

Training data

Figure 6.10 shows randomly selected training samples from our experiments. Different from commonly used datasets such as ShapeNet, foregrounds and backgrounds are not separated, viewpoints are unknown, objects have various shapes and sizes, and they locate and rotate freely. Though learning about 3D from these datasets is very challenging due to

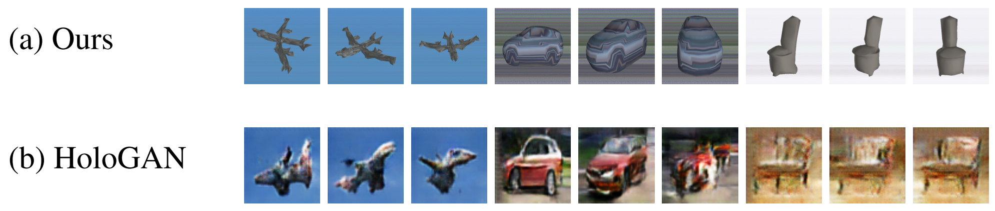


Figure 6.11 Comparison of ours with HoloGAN on PASCAL dataset. Images are rendered at 50 degree intervals.

these characteristics, we think that trying this is a necessary step toward fundamental 3D understanding in machines.

Comparison with HoloGAN on PASCAL

We showed comparison between ours and HoloGAN on CIFAR-10 in Figure 6.7. For this comparison, we used codes provided by the authors. Specifically, we used default settings for CelebA dataset except for elevation and azimuth, which are set to the values for the cars included in the thesis. In addition, we show comparison on PASCAL dataset in Figure 6.11. Though images generated by HoloGAN are improved, they still lack consistency from multiple viewpoints. Especially, this method seems not good at generating front images of cars.

Randomly selected results

In Figure 6.5 and Figure 6.9, selected results on CIFAR-10 and PASCAL datasets are shown. For further qualitative evaluation, randomly selected results are shown in Figure 6.12 to Figure 6.21. Input images are shown in the top rows, reconstructed images using estimated shapes, poses, textures, and backgrounds are shown in the middle rows, and reconstructed images using a fixed viewpoint are shown in the bottom rows. For training set, the best shapes and viewpoints found during training are used, and for validation set, predicted shapes and viewpoints by the shape estimator and viewpoint estimator are used. Photometric fine-tuning are used for only validation set.

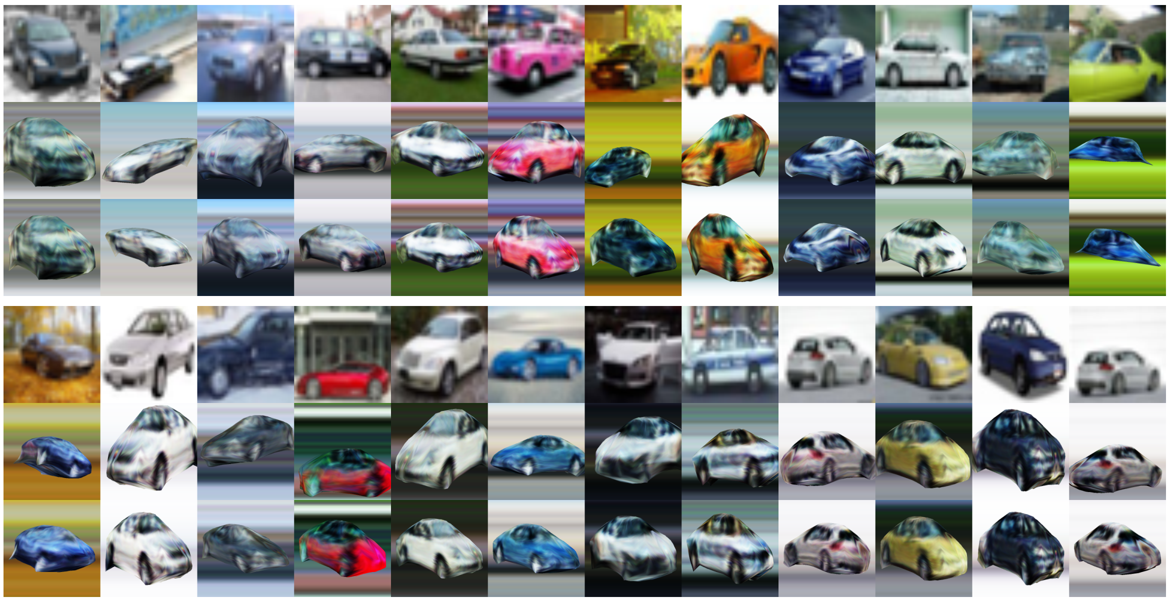


Figure 6.12 Randomly selected results on CIFAR-10 *car* training set.

6.5 Summary

This chapter presented a method to learn single-view reconstruction of the 3D shape, pose, and texture of objects from categorized natural images in a self-supervised manner. This work is the first to achieve learning 3D object reconstruction from unannotated images. The two main techniques were two-stage training to focus on shapes, and inducting strong regularization and constraints to the surface of shapes and background images. Results of experiments on CIFAR-10 and PASCAL confirm the importance of our proposed techniques. Besides, we summarized observations and possible research directions.

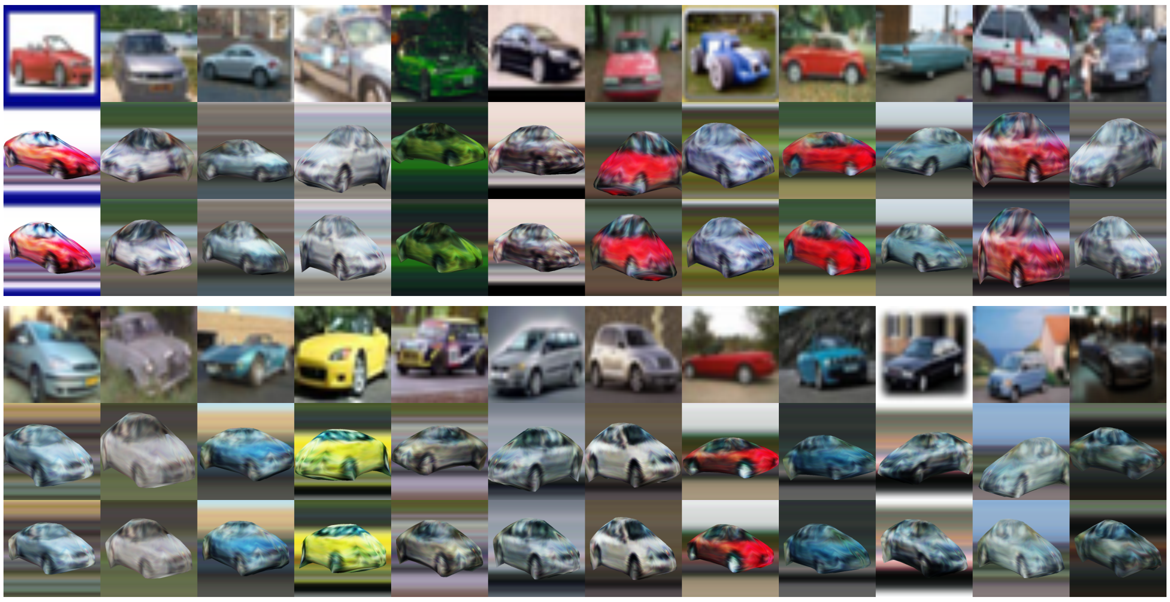


Figure 6.13 Randomly selected results on CIFAR-10 *car* validation set.

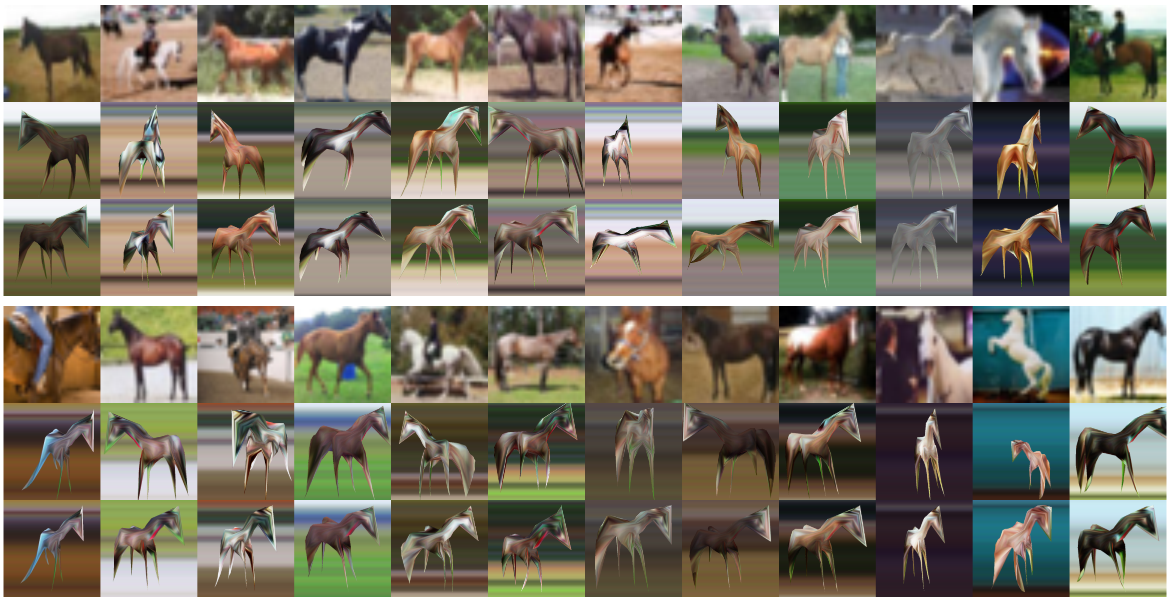


Figure 6.14 Randomly selected results on CIFAR-10 *horse* training set.

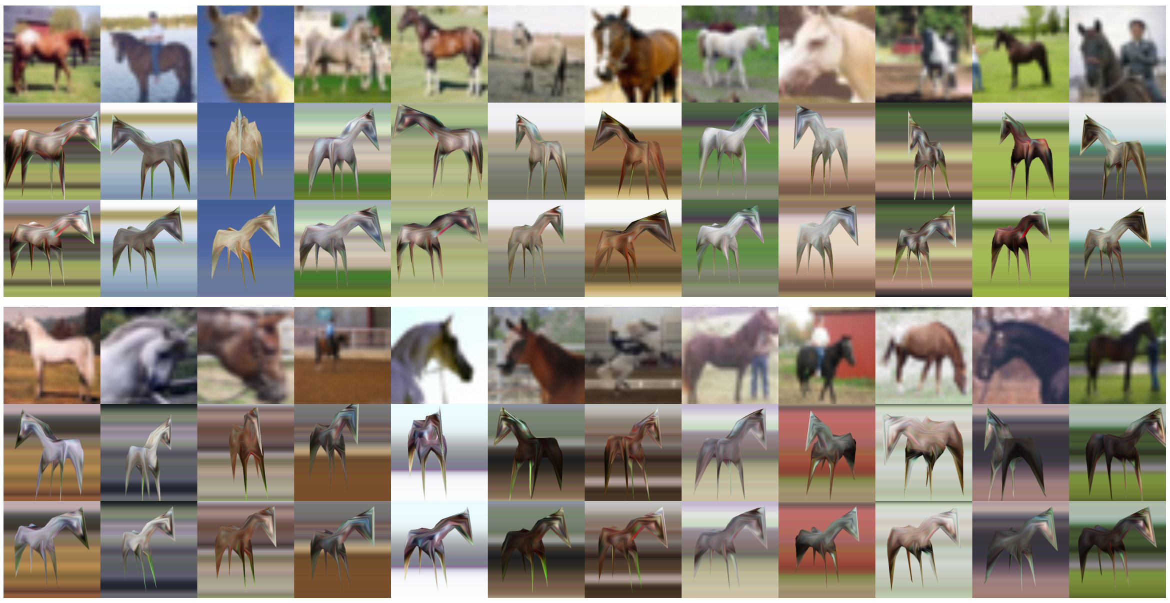


Figure 6.15 Randomly selected results on CIFAR-10 *horse* validation set.

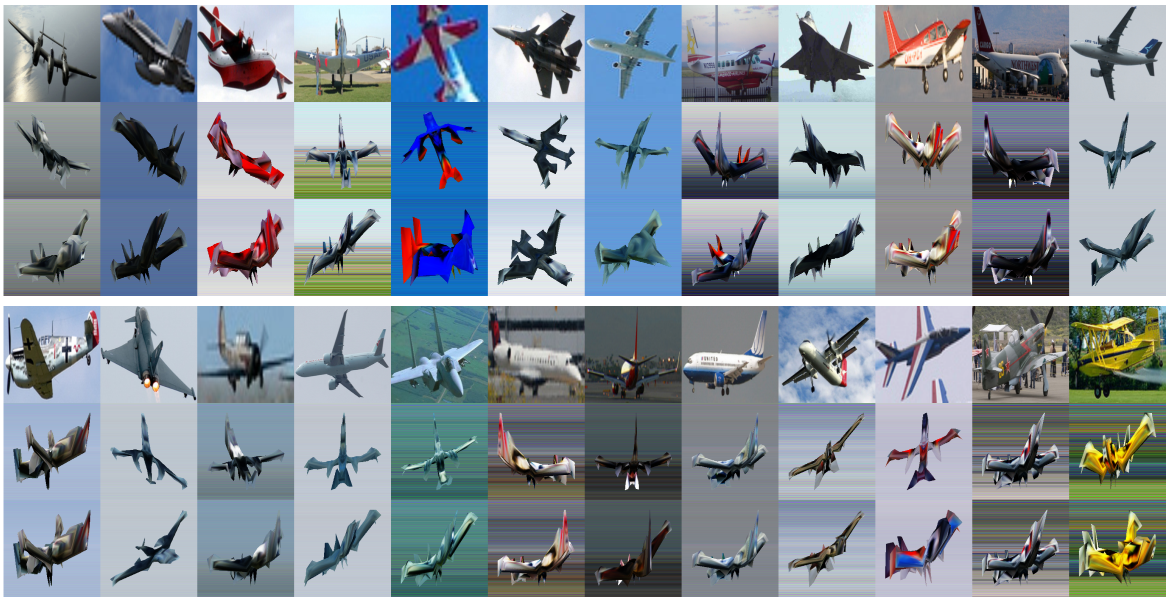


Figure 6.16 Randomly selected results on PASCAL *aeroplane* training set.

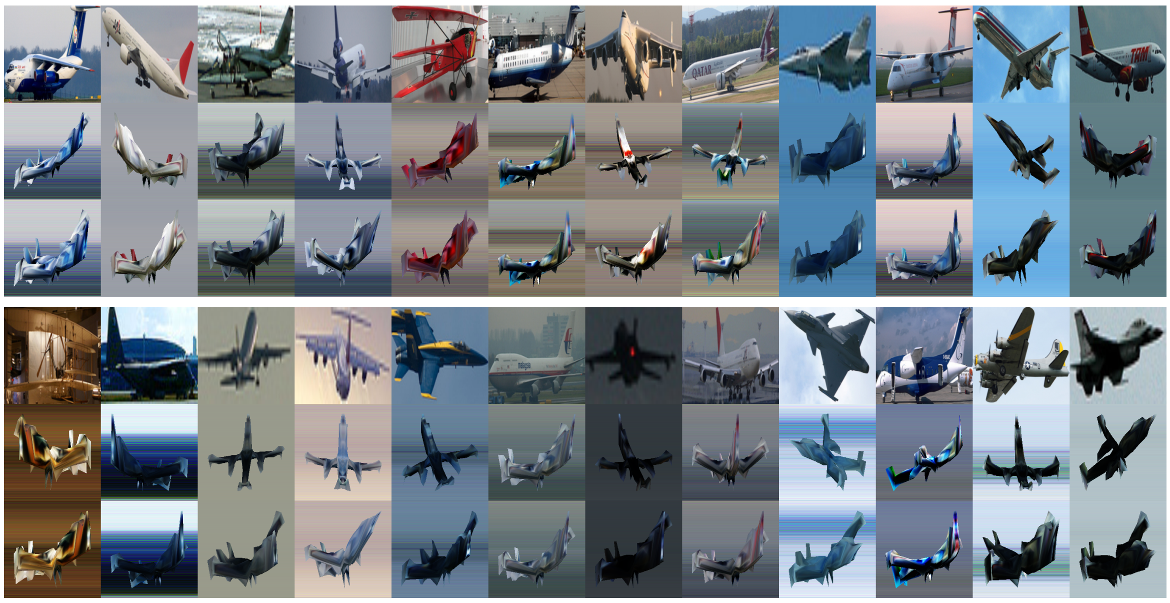


Figure 6.17 Randomly selected results on PASCAL *aeroplane* validation set.

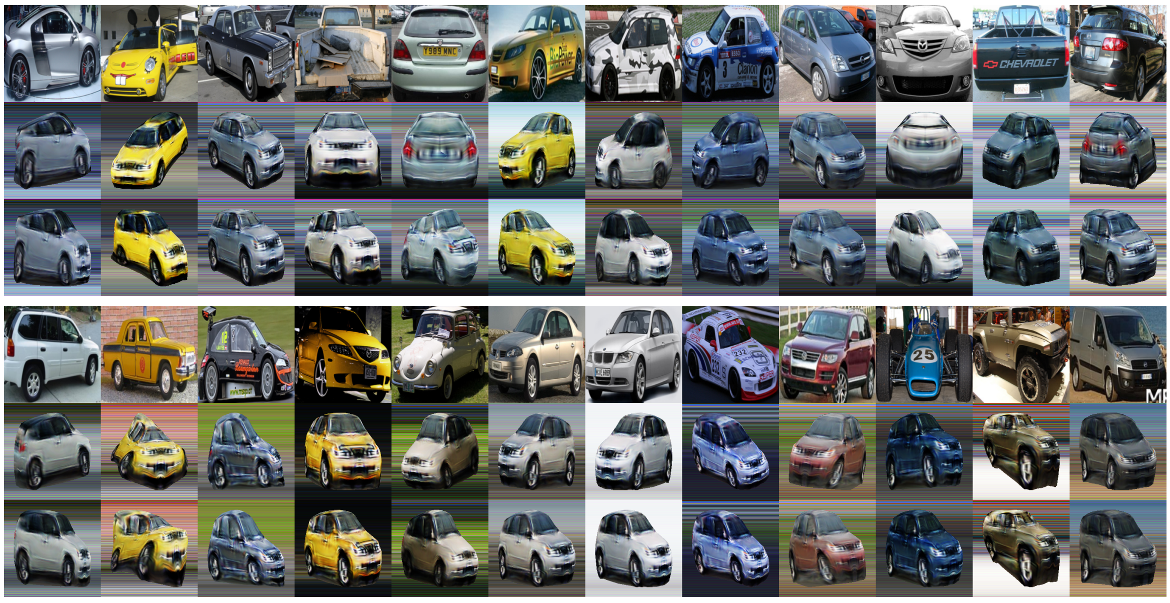


Figure 6.18 Randomly selected results on PASCAL *car* training set.

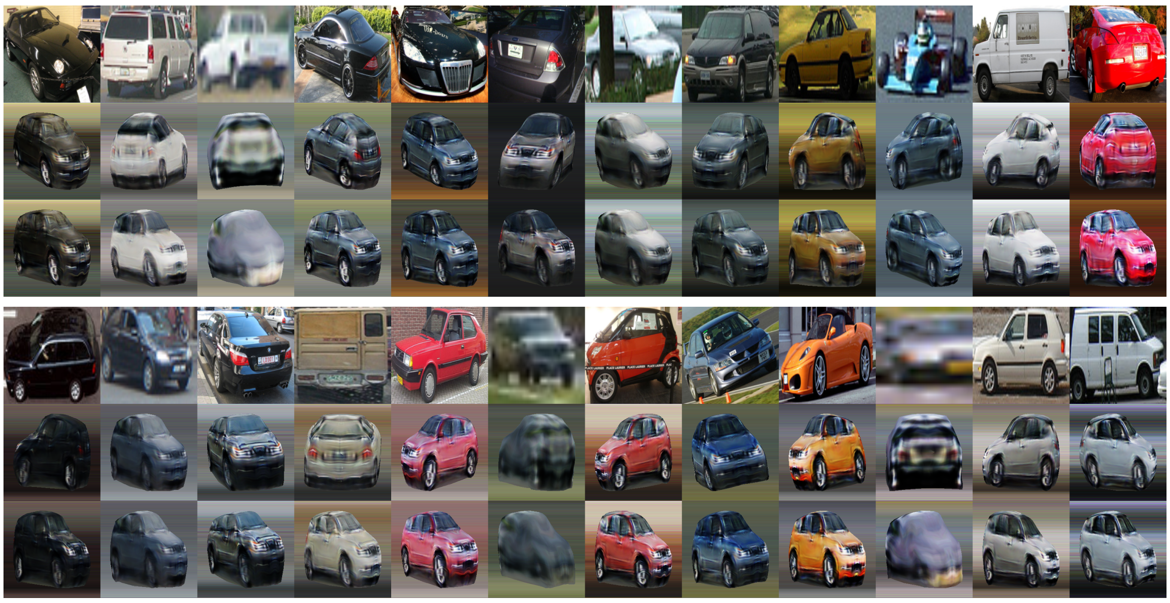


Figure 6.19 Randomly selected results on PASCAL *car* validation set.

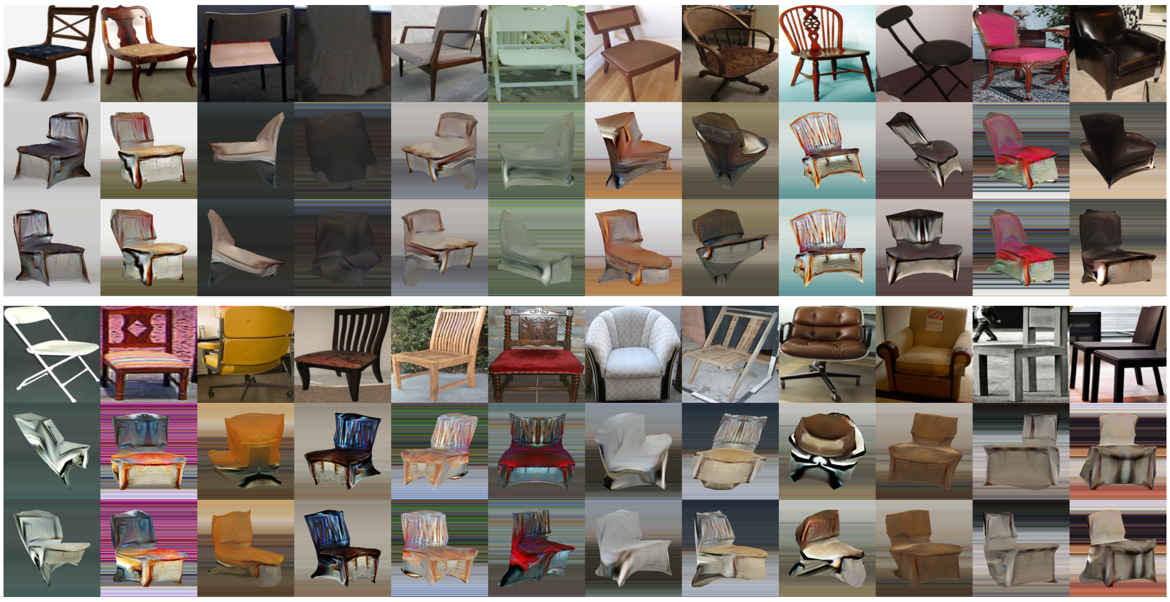


Figure 6.20 Randomly selected results on PASCAL *chair* training set.

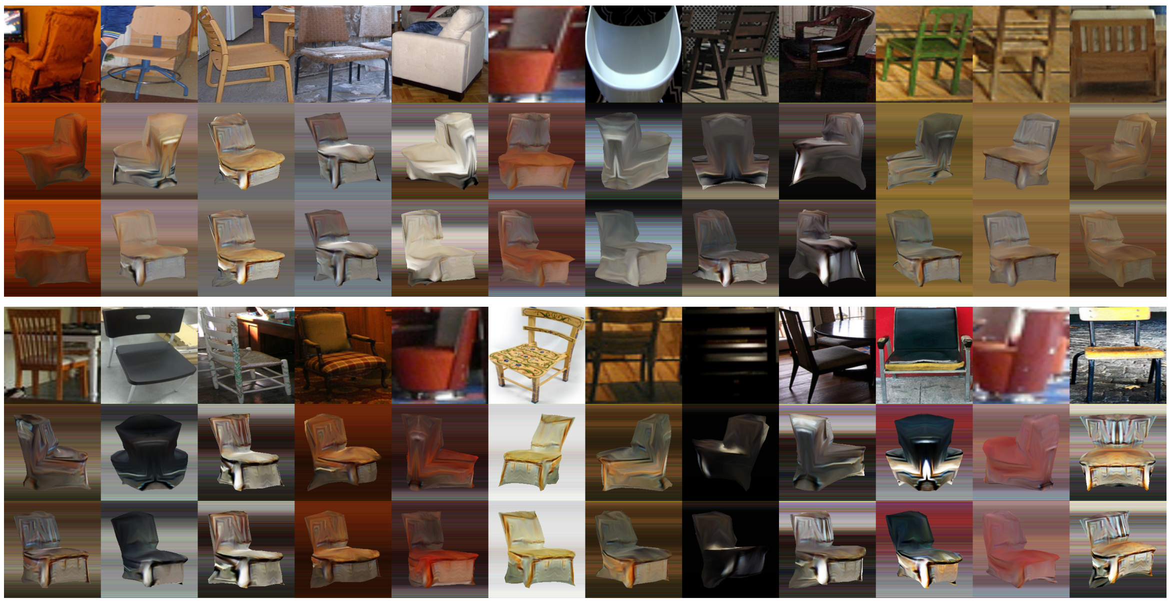


Figure 6.21 Randomly selected results on PASCAL *chair* validation set.

Chapter 7

Conclusion

Understanding the 3D structure of objects from an image is essential in computer vision for its broad applications in robotics, augmented reality, autonomous driving, and 3D designing. To enable single-view 3D object reconstruction in various object categories beyond ShapeNet dataset, we think that leveraging 2D images for training instead of employing 3D objects is a promising direction because we can access a massive amount of 2D visual data on the Internet. With this spirit, we developed several techniques for different dataset settings.

What we want to get is 3D, but what we have for training is 2D. Therefore, we must have something to bridge 3D world space and 2D image space. Rendering is a technique to translate 3D models into 2D images, but it is one-way street. To make it a bridge, we need inverse of rendering. We realize it by defining derivative of a rendering function. Also, since most modern computer vision methods are described by the language of neural networks, a compatibility with them is important. Also, to represent rich 3D models compactly, we have to use mesh as a 3D representation. In Chapter 3, we proposed a differentiable mesh rendering method for neural networks to realize end-to-end training of neural networks that include rendering functions. While some related renderers [8, 54] provide gradients by blurring output images, the proposed renderer was showed to provide reasonable gradients without affecting image quality. In addition, we found that using the gradient of a loss function to be optimized can improve the quality of gradients. These findings would be useful for designing more sophisticated differentiable rasterization functions.

When developing a robot that grasps a specific object, we have the physical object in most cases. In such a case, collecting multi-view images of the objects, camera parameters, and silhouette annotations are relatively easy with a calibrated 3D scanning system. In Chapter 4, we applied the proposed renderer to learning 3D object reconstruction from multi-view images with silhouette and camera parameter annotations. We demonstrated that the renderer is useful for learning with neural networks. We found that using meshes is better than using voxels in learning from images.

Learning 3D object reconstruction is to have knowledge of typical 3D shapes and their variations. By learning only from image reconstruction, correct knowledge may not be obtained because there are strange shapes that reconstruct images accurately. This issue is serious when the number of views per image is a few or one. We need additional criteria about correctness of 3D shapes. In Chapter 5, we proposed to learn the validity of views of reconstructed objects to overcome 3D shape ambiguity in images. We significantly improved 3D reconstruction accuracy with our proposed method, especially when the number of views for training is small. We think that learning only from reconstruction is not enough in general, and having other criteria that inform machines about 3D scenes is important.

Learning from images without any additional annotations is very low cost. To have 3D reconstruction models for everything around us, this learning scheme would be required. In Chapter 6, we proposed the first method for learning 3D object reconstruction from only images. We demonstrated that ours can learn 3D geometry on challenging CIFAR-10 and PASCAL datasets. We found that introducing a two-step training method and several heuristic constraints is useful to overcome the ill-posed reconstruction problem. We also had discussion on difficult cases and how they could be overcome. We introduced assumptions (or priors) of geometry, lighting, appearance on object surfaces and background. In some object categories, our assumptions do not hold.

One common principle in these developments is to introduce our structural knowledge into neural networks. 2D images are generated from 3D models, and the way of 3D-to-2D conversion is well known. We leverage it for weakly-supervised learning of 3D reconstruction with 2D image supervision in Chapter 3 and Chapter 4. A 3D shape can be looked at from multiple viewpoints, and all views have to be proper if the shape is correct. This knowledge improved accuracy of 3D reconstruction in Chapter 5. When supervision is very limited, we have to design a training method and constraints carefully with our knowledge about 3D scenes, as shown in Chapter 6. An images is 2D array of pixels, though the world that produced the image is originally in 3D space. The recent popular image recognition methods [47] and image generation methods [17] process images in the 2D space and ignore that fact that the world is 3D. Ours is on the opposite. Though our work presented here is limited in 3D reconstruction, using 3D inductive bias would be beneficial for other tasks in computer vision.

Since our method can estimate the rough 3D shapes of objects such as tables and chairs, it would be useful for AR applications and simple obstacle avoidance for robots. When it is improved further in the future, it will be used for robot grasping and assistance for

3D designing. Currently, the object category to handle is limited to thirteen classes in ShapeNet and other ones of simple shapes. One bottleneck is in the shape generation method. We create shapes by deforming a pre-defined sphere, which limits the topology of objects. Recently, several methods to overcome this issue has been developed in the context of 3D supervision [69, 75]. Integrating them with our view-based training would be a promising direction. Another issue is learning of complicated shapes from unannotated natural images. This is because the priors we designed are for generating simple 3D scenes. Instead, learning sophisticated priors from data would be required for object categories of complex shapes. Learning 3D shape priors in some object categories [3, 21, 118] and transferring the prior by leveraging the knowledge of generalization of 3D object reconstruction for unseen classes [96, 137] is a possible direction.

Applications of differentiable rendering are not limited to single-view 3D object reconstruction. In computer vision, many problems are solved by modeling and inverting an image generation process. Images are generated via rendering from the 3D world, and a polygon mesh is an efficient, rich and intuitive 3D representation. Therefore, "backward pass" of mesh renderers is important. After releasing the code of our renderer, it has been applied to human pose estimation [33, 56, 110, 125], face reconstruction [90, 119, 138], hand pose estimation [4, 136, 139], 3D animal reconstruction [140], object pose estimation [98, 116], object tracking [104], correspondence estimation [48, 131], finding 3D adversarial examples [1, 22, 120, 124], depth estimation [37], image manipulation [129], image segmentation [20], cartoon character generation [79], and scene generation [59], in addition to single-view object reconstruction [14, 36, 77, 122]. Besides ours and OpenDR [57], newer differentiable rendering methods are emerging [8, 53, 54, 58, 135]. We would like to see this research field grow more.

Toward single-view 3D object reconstruction of various object categories, we developed several essential components in this thesis. We hope that our work will lead to the achievement of this goal.

References

[1] Michael A Alcorn, Qi Li, Zhitao Gong, Chengfei Wang, Long Mai, Wei-Shinn Ku, and Anh Nguyen. Strike (with) a pose: Neural networks are easily fooled by strange poses of familiar objects. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[2] Martin Arjovsky and Léon Bottou. Towards principled methods for training generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[3] Abhishek Badki, Orazio Gallo, Jan Kautz, and Pradeep Sen. Meshlet priors for 3d mesh reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[4] Seungryul Baek, Kwang In Kim, and Tae-Kyun Kim. Pushing the envelope for rgb-based dense 3d hand pose estimation via neural rendering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[5] Jonathan T Barron and Jitendra Malik. Shape, illumination, and reflectance from shading. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 37(8):1670–1687, 2014.

[6] Harry Barrow, J Tenenbaum, A Hanson, and E Riseman. *Recovering intrinsic scene characteristics*. 1978.

[7] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv*, 2015.

[8] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[9] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European Conference on Computer Vision (ECCV)*, 2016.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[11] Vincent Dumoulin, Jonathon Shlens, Manjunath Kudlur, Arash Behboodi, Filip Lemic, Adam Wolisz, Marco Molinaro, Christoph Hirche, Masahito Hayashi, Emilio Bagan, et al. A learned representation for artistic style. In *International Conference on Learning Representations (ICLR)*, 2017.

[12] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision (IJCV)*, 88(2):303–338, 2010.

[13] Haoqiang Fan, Hao Su, and Leonidas Guibas. A point set generation network for 3d object reconstruction from a single image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[14] Qiaojun Feng, Yue Meng, Mo Shan, and Nikolay Atanasov. Localization and mapping using instance-specific mesh models. In *International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[15] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *Journal of Machine Learning Research (JMLR)*, 17(59): 1–35, 2016.

[16] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[17] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014.

[18] Roger Grosse, Micah K Johnson, Edward H Adelson, and William T Freeman. Ground truth dataset and baseline evaluations for intrinsic image algorithms. In *International Conference on Computer Vision (ICCV)*, 2009.

[19] Thibault Groueix, Matthew Fisher, Vladimir G Kim, Bryan C Russell, and Mathieu Aubry. Atlasnet: A papier-mache approach to learning 3d surface generation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[20] Shir Gur, Tal Shaharabany, and Lior Wolf. End to end trainable active contours via differentiable rendering. In *International Conference on Learning Representations (ICLR)*, 2020.

[21] JunYoung Gwak, Christopher B Choy, Manmohan Chandraker, Animesh Garg, and Silvio Savarese. Weakly supervised 3d reconstruction with adversarial constraint. In *International Conference on 3D Vision (3DV)*, 2017.

[22] Abdullah Hamdi and Bernard Ghanem. Towards analyzing semantic robustness of deep neural networks. In *International Conference on Computer Vision Workshop on Explaining Visual Artificial Intelligence Models*, 2019.

[23] Christian Hane, Shubham Tulsiani, and Jitendra Malik. Hierarchical surface prediction. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 42(6): 1348–1361, 2019.

[24] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[25] Paul Henderson and Vittorio Ferrari. Learning single-image 3d reconstruction by generative modelling of shape, pose and shading. *International Journal of Computer Vision (IJCV)*, 128(4):835–854, 2019.

[26] Philipp Henzler, Niloy Mitra, and Tobias Ritschel. Escaping plato's cave using adversarial training: 3d shape from unstructured 2d image collections. In *International Conference on Computer Vision (ICCV)*, 2019.

[27] Philipp Henzler, Niloy J Mitra, and Tobias Ritschel. Escaping plato's cave: 3d shape from adversarial rendering. In *International Conference on Computer Vision (ICCV)*, 2019.

[28] Berthold KP Horn. Shape from shading: A method for obtaining the shape of a smooth opaque object from one view. Technical report, Massachusetts Institute of Technology, 1970.

[29] Berthold KP Horn. Determining lightness from an image. *Computer graphics and image processing*, 3(4):277–299, 1974.

[30] Katsushi Ikeuchi and Berthold KP Horn. Numerical shape from shading and occluding boundaries. *Artificial intelligence*, 17(1-3):141–184, 1981.

[31] Eldar Insafutdinov and Alexey Dosovitskiy. Unsupervised learning of shape and pose with differentiable point clouds. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[32] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015.

[33] Boyi Jiang, Juyong Zhang, Yang Hong, Jinhao Luo, Ligang Liu, and Hujun Bao. Bcnet: Learning body and cloth shape from a single image. In *European Conference on Computer Vision (ECCV)*, 2020.

[34] Li Jiang, Shaoshuai Shi, Xiaojuan Qi, and Jiaya Jia. Gal: Geometric adversarial loss for single-view 3d-object reconstruction. In *European Conference on Computer Vision (ECCV)*, 2018.

[35] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision (ECCV)*, 2016.

[36] Angjoo Kanazawa, Shubham Tulsiani, Alexei A Efros, and Jitendra Malik. Learning category-specific mesh reconstruction from image collections. In *European Conference on Computer Vision (ECCV)*, 2018.

[37] Masaya Kaneko, Ken Sakurada, and Kiyoharu Aizawa. Tridepth: Triangular patch-based deep depth prediction. In *International Conference on Computer Vision Workshop on Deep Learning for Visual SLAM*, 2019.

[38] Abhishek Kar, Shubham Tulsiani, Joao Carreira, and Jitendra Malik. Category-specific object reconstruction from a single image. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[39] Abhishek Kar, Christian Häne, and Jitendra Malik. Learning a multi-view stereo machine. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[40] Hiroharu Kato and Tatsuya Harada. Learning view priors for single-view 3d reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[41] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[42] Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon. Differentiable rendering: A survey. *arXiv*, 2020.

[43] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[44] Roman Klokov and Victor Lempitsky. Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *International Conference on Computer Vision (ICCV)*, 2017.

[45] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2011.

[46] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[47] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

[48] Nilesh Kulkarni, Abhinav Gupta, and Shubham Tulsiani. Canonical surface mapping via geometric cycle consistency. In *International Conference on Computer Vision (ICCV)*, 2019.

[49] Andrey Kurenkov, Jingwei Ji, Animesh Garg, Viraj Mehta, JunYoung Gwak, Christopher Choy, and Silvio Savarese. Deformnet: Free-form deformation network for 3d shape reconstruction from a single image. In *Winter Conference on Applications of Computer Vision (WACV)*, 2018.

[50] Kiriakos N Kutulakos and Steven M Seitz. A theory of shape by space carving. *International Journal of Computer Vision (IJCV)*, 38(3):199–218, 2000.

[51] Edwin H Land and John J McCann. Lightness and retinex theory. *Journal of the Optical Society of America (JOSA)*, 61(1):1–11, 1971.

[52] Ke Li, Bharath Hariharan, and Jitendra Malik. Iterative instance segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[53] Tzu-Mao Li, Miika Aittala, Frédo Durand, and Jaakko Lehtinen. Differentiable monte carlo ray tracing through edge sampling. *ACM Transactions on Graphics (TOG)*, 37 (6):222:1–222:11, 2018.

[54] Shichen Liu, Weikai Chen, Tianye Li, and Hao Li. Soft rasterizer: Differentiable rendering for unsupervised single-view mesh reconstruction. In *International Conference on Computer Vision (ICCV)*, 2019.

[55] Shichen Liu, Shunsuke Saito, Weikai Chen, and Hao Li. Learning to infer implicit surfaces without 3d supervision. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[56] Wen Liu, Zhixin Piao, Jie Min, Wenhan Luo, Lin Ma, and Shenghua Gao. Liquid warping gan: A unified framework for human motion imitation, appearance transfer and novel view synthesis. In *International Conference on Computer Vision (ICCV)*, 2019.

[57] Matthew M Loper and Michael J Black. Opendr: An approximate differentiable renderer. In *European Conference on Computer Vision (ECCV)*, 2014.

[58] Guillaume Loubet, Nicolas Holzschuch, and Wenzel Jakob. Reparameterizing discontinuous integrands for differentiable rendering. *ACM Transactions on Graphics (TOG)*, 38(6):1–14, 2019.

[59] Andrew Luo, Zhoutong Zhang, Jiajun Wu, and Joshua B Tenenbaum. End-to-end optimization of scene layout. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[60] Steve Marschner and Peter Shirley. *Fundamentals of computer graphics*. CRC Press, 2015.

[61] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[62] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[63] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision (ECCV)*, 2020.

[64] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv*, 2014.

[65] Takeru Miyato and Masanori Koyama. cgans with projection discriminator. In *International Conference on Learning Representations (ICLR)*, 2018.

[66] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2018.

[67] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks. *Google Research Blog*, 2015.

[68] Alexander Mordvintsev, Nicola Pezzotti, Ludwig Schubert, and Chris Olah. Differentiable image parameterizations. *Distill*, 2018.

[69] Charlie Nash, Yaroslav Ganin, SM Eslami, and Peter W Battaglia. Polygen: An autoregressive generative model of 3d meshes. In *International Conference on Machine Learning (ICML)*, 2020.

[70] Shree K Nayar and Yasuo Nakagawa. Shape from focus. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 16(8):824–831, 1994.

[71] Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. Hologan: Unsupervised learning of 3d representations from natural images. In *International Conference on Computer Vision (ICCV)*, 2019.

[72] Michael Niemeyer, Lars Mescheder, Michael Oechsle, and Andreas Geiger. Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[73] Merlin Nimier-David, Delio Vicini, Tizian Zeltner, and Wenzel Jakob. Mitsuba 2: a retargetable forward and inverse renderer. *ACM Transactions on Graphics (TOG)*, 38 (6):203:1–203:17, 2019.

[74] Ido Omer and Michael Werman. Color lines: Image specific color representation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2004.

[75] Junyi Pan, Xiaoguang Han, Weikai Chen, Jiapeng Tang, and Kui Jia. Deep mesh reconstruction from single rgb images via topology modification networks. In *International Conference on Computer Vision (ICCV)*, 2019.

[76] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[77] Bo Peng, Wei Wang, Jing Dong, and Tieniu Tan. Learning pose-invariant 3d object reconstruction from single-view images. *arXiv*, 2020.

[78] Alex Paul Pentland. A new sense for depth of field. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 9(4):523–531, 1987.

[79] Omid Poursaeed, Vladimir Kim, Eli Shechtman, Jun Saito, and Serge Belongie. Neural puppet: Generative layered cartoon characters. In *Winter Conference on Applications of Computer Vision (WACV)*, 2020.

[80] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[81] Charles R Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[82] Charles Ruizhongtai Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[83] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *International Conference on Learning Representations (ICLR)*, 2016.

[84] Danilo Jimenez Rezende, SM Ali Eslami, Shakir Mohamed, Peter Battaglia, Max Jaderberg, and Nicolas Heess. Unsupervised learning of 3d structure from images. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[85] Stephan R Richter and Stefan Roth. Matryoshka networks: Predicting 3d geometry via nested shape layers. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[86] Gernot Riegler, Ali Osman Ulusoys, and Andreas Geiger. Octnet: Learning deep 3d representations at high resolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[87] Carsten Rother, Martin Kiefel, Lumin Zhang, Bernhard Schölkopf, and Peter V Gehler. Recovering intrinsic images with a global sparsity prior on reflectance. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2011.

[88] Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: nonlinear phenomena*, 60(1-4):259–268, 1992.

[89] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[90] Mihir Sahasrabudhe, Zhixin Shu, Edward Bartrum, Riza Alp Guler, Dimitris Samaras, and Iasonas Kokkinos. Lifting autoencoders: Unsupervised learning of a fully-disentangled 3d morphable model using deep non-rigid structure from motion. In *International Conference on Computer Vision Workshop on Geometry Meets Deep Learning*, 2019.

[91] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[92] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[93] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016.

[94] Thomas W Sederberg and Scott R Parry. Free-form deformation of solid geometric models. *ACM Transactions on Graphics (TOG)*, 20(4):151–160, 1986.

[95] Li Shen, Chuohao Yeo, and Binh-Son Hua. Intrinsic image decomposition using a sparse representation of reflectance. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 35(12):2904–2915, 2013.

[96] Daeyun Shin, Charless C Fowlkes, and Derek Hoiem. Pixels, voxels, and views: A study of shape representations for single view 3d object shape prediction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[97] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations (ICLR)*, 2015.

[98] Juil Sock, Pedro Castro, Anil Armagan, Guillermo Garcia-Hernando, and Tae-Kyun Kim. Tackling two challenges of 6d object pose estimation: Lack of real annotated rgb images and scalability to number of objects. *arXiv*, 2020.

[99] Supasorn Suwajanakorn, Noah Snavely, Jonathan J Tompson, and Mohammad Norouzi. Discovery of latent 3d keypoints via end-to-end geometric reasoning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[100] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[101] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In *International Conference on Computer Vision (ICCV)*, 2017.

[102] Maxim Tatarchenko, Stephan R Richter, René Ranftl, Zhuwen Li, Vladlen Koltun, and Thomas Brox. What do single-view 3d reconstruction networks learn? In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[103] Gabriel Taubin. A signal processing approach to fair surface design. In *International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 1995.

[104] Catherine Taylor, Chris Mullany, Robin McNicholas, and Darren Cosker. Vr props: An end-to-end pipeline for transporting real objects into virtual and augmented environments. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, 2019.

[105] Sebastian Thrun and Ben Wegbreit. Shape from symmetry. In *International Conference on Computer Vision (ICCV)*, 2005.

[106] Shubham Tulsiani and Jitendra Malik. Viewpoints and keypoints. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[107] Shubham Tulsiani, Tinghui Zhou, Alexei A Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[108] Shubham Tulsiani, Alexei A Efros, and Jitendra Malik. Multi-view consistency as supervisory signal for learning shape and pose prediction. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[109] Shimon Ullman. *The interpretation of visual motion.* MIT Press, 1979.

[110] Min Wang, Feng Qiu, Wentao Liu, Chen Qian, Xiaowei Zhou, and Lizhuang Ma. Ellipbody: A light-weight and part-based representation for human pose and shape recovery. *arXiv*, 2020.

[111] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2mesh: Generating 3d mesh models from single rgb images. In *European Conference on Computer Vision (ECCV)*, 2018.

[112] Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions on Graphics (TOG)*, 36(4):72:1–72:11, 2017.

[113] Andrew P Witkin. Recovering surface shape and orientation from texture. *Artificial intelligence*, 17(1-3):17–45, 1981.

[114] Oliver Woodford, Philip Torr, Ian Reid, and Andrew Fitzgibbon. Global stereo reconstruction under second-order smoothness priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 31(12):2115–2128, 2009.

[115] Robert J Woodham. Photometric method for determining surface orientation from multiple images. *Optical engineering*, 19(1):139 – 144, 1980.

[116] Di Wu, Yihao Chen, Xianbiao Qi, Yuyong Jian, Weixuan Chen, and Rong Xiao. Neural mesh refiner for 6-dof pose estimation. *arXiv*, 2020.

[117] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[118] Jiajun Wu, Chengkai Zhang, Xiuming Zhang, Zhoutong Zhang, William T Freeman, and Joshua B Tenenbaum. Learning shape priors for single-view 3d completion and reconstruction. In *European Conference on Computer Vision (ECCV)*, 2018.

[119] Shangzhe Wu, Christian Rupprecht, and Andrea Vedaldi. Unsupervised learning of probably symmetric deformable 3d objects from images in the wild. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[120] Xugang Wu, Xiaoping Wang, Xu Zhou, and Songlei Jian. Sta: Adversarial attacks on siamese trackers. *arXiv*, 2019.

[121] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[122] Nan Xiang, Li Wang, Tao Jiang, Yanran Li, Xiaosong Yang, and Jianjun Zhang. Single-image mesh reconstruction and pose estimation via generative normal map. In *International Conference on Computer Animation and Social Agents (CASA)*, 2019.

[123] Yu Xiang, Roozbeh Mottaghi, and Silvio Savarese. Beyond pascal: A benchmark for 3d object detection in the wild. In *Winter Conference on Applications of Computer Vision (WACV)*, 2014.

[124] Chaowei Xiao, Dawei Yang, Bo Li, Jia Deng, and Mingyan Liu. Meshadv: Adversarial meshes for visual recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[125] Sasuke Yamane, Hirotake Yamazoe, and Joo-Ho Lee. Human motion generation based on gan toward unsupervised 3d human pose estimation. In *Asian Conference on Pattern Recognition (ACPR)*, 2019.

[126] Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. Perspective transformer nets: Learning single-view 3d object reconstruction without 3d supervision. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.

[127] B. Yang, S. Rosa, A. Markham, N. Trigoni, and H. Wen. Dense 3d object reconstruction from a single depth view. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 41(12):2820–2834, 2018.

[128] Jimei Yang, Scott E Reed, Ming-Hsuan Yang, and Honglak Lee. Weakly-supervised disentangling with recurrent transformations for 3d view synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2015.

[129] Shunyu Yao, Tzu Ming Hsu, Jun-Yan Zhu, Jiajun Wu, Antonio Torralba, Bill Freeman, and Josh Tenenbaum. 3d-aware scene manipulation via inverse graphics. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[130] Wang Yifan, Felice Serena, Shihao Wu, Cengiz Öztireli, and Olga Sorkine-Hornung. Differentiable surface splatting for point-based geometry processing. *ACM Transactions on Graphics (TOG)*, 38(6), 2019.

[131] Yang You, Chengkun Li, Yujing Lou, Zhoujun Cheng, Lizhuang Ma, Cewu Lu, and Weiming Wang. Semantic correspondence via 2d-3d-2d cycle. *arXiv*, 2020.

[132] M Ersin Yumer and Niloy J Mitra. Learning semantic deformation flows with 3d convolutional networks. In *European Conference on Computer Vision (ECCV)*, 2016.

[133] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference (BMVC)*, 2016.

[134] Xiaohui Zeng, Chenxi Liu, Yu-Siang Wang, Weichao Qiu, Lingxi Xie, Yu-Wing Tai, Chi-Keung Tang, and Alan L Yuille. Adversarial attacks beyond the image space. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[135] Cheng Zhang, Bailey Miller, Kai Yan, Ioannis Gkioulekas, and Shuang Zhao. Path-space differentiable rendering. *ACM Transactions on Graphics (TOG)*, 39(6):143:1–143:19, 2020.

[136] Xiong Zhang, Qiang Li, Hong Mo, Wenbo Zhang, and Wen Zheng. End-to-end hand mesh recovery from a monocular rgb image. In *International Conference on Computer Vision (ICCV)*, 2019.

[137] Xiuming Zhang, Zhoutong Zhang, Chengkai Zhang, Josh Tenenbaum, Bill Freeman, and Jiajun Wu. Learning to reconstruct shapes from unseen classes. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[138] Hang Zhou, Jihao Liu, Ziwei Liu, Yu Liu, and Xiaogang Wang. Rotate-and-render: Unsupervised photorealistic face rotation from single-view images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[139] Christian Zimmermann, Duygu Ceylan, Jimei Yang, Bryan Russell, Max Argus, and Thomas Brox. Freihand: A dataset for markerless capture of hand pose and shape from single rgb images. In *International Conference on Computer Vision (ICCV)*, 2019.

[140] Silvia Zuffi, Angjoo Kanazawa, Tanja Berger-Wolf, and Michael J Black. Three-d safari: Learning to estimate zebra pose, shape, and texture from images" in the wild". In *International Conference on Computer Vision (ICCV)*, 2019.

Publications

Reviewed Conference

1. Deniz Beker, Hiroharu Kato, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon, “Self-Supervised Differentiable Rendering for Monocular 3D Object Detection”, European Conference on Computer Vision (ECCV), 2020.

2. Hiroharu Kato and Tatsuya Harada, “Learning View Priors for Single-view 3D Reconstruction”, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019.

3. Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada, “Neural 3D Mesh Renderer”, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018.

4. Hiroharu Kato and Tatsuya Harada, “Image Reconstruction from Bag-of-Visual-Words”, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2014.

5. Hiroharu Kato, Tatsuya Harada, and Yasuo Kuniyoshi, “Visual Anomaly Detection from Small Samples for Mobile Robots”, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2012.

Preprints

1. Hiroharu Kato, Deniz Beker, Mihai Morariu, Takahiro Ando, Toru Matsuoka, Wadim Kehl, and Adrien Gaidon, “Differentiable Rendering: A Survey”, arXiv:2006.12057, 2020.

2. Hiroharu Kato and Tatsuya Harada, “Self-supervised Learning of 3D Objects from Natural Images”, arXiv:1911.08850, 2019.

3. Andrew Shin, Leopold Crestel, Hiroharu Kato, Kuniaki Saito, Katsunori Ohnishi, Masataka Yamaguchi, Masahiro Nakawaki, Yoshitaka Ushiku, and Tatsuya Harada, "Melody Generation for Pop Music via Word Representation of Musical Properties", arXiv:1710.11549, 2017.
4. Hiroharu Kato and Tatsuya Harada, "Visual Language Modeling on CNN Image Representations", arXiv:1511.02872, 2015.
5. Hiroharu Kato and Tatsuya Harada, "Image Reconstruction from Bag-of-Visual-Words", arXiv:1505.05190, 2015.

Talks

1. Hiroharu Kato, "Recent Trends in Differentiable Rendering", The Symposium on Sensing via Image Information (SSII). 2020 (in Japanese).
2. Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada, "Neural 3D Mesh Renderer", The Japanese Society for Artificial Intelligence (JSAI), 2019 (in Japanese).
3. Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada, "[CVPR2018] Neural 3D Mesh Renderer", Meeting on Image Recognition and Understanding (MIRU). 2018 (in Japanese).
4. Hiroharu Kato, "Modality Translation and Image Generation", The Symposium on Sensing via Image Information (SSII). 2018 (in Japanese).
5. Hiroharu Kato and Tatsuya Harada, "[CVPR2014] Image Reconstruction from Bag-of-Visual-Words", Meeting on Image Recognition and Understanding (MIRU). 2014 (in Japanese).

Awards

1. NVIDIA Pioneering Research Award, Aug. 2018.