Unsupervised Learning of Probably Symmetric Deformable 3D Objects from Images in the Wild
Development Track

Yuseung Lee
KAIST
phillip0701@kaist.ac.kr

Inhee Lee
KAIST
ininin0516@kaist.ac.kr

Abstract

In this paper, we present the reproduction results of Unsup3D [13] from scratch and the challenges we faced during the implementation. The objective of Unsup3D is to learn the underlying 3D shape from single 2D images without any supervision. It decomposes the input into four visual components and reconstructs it through photo-geometric autoencoder. To obatain a geometric cue, Unsup3D applies symmetric assumption and introduces a confidence map. We implemented most parts of Unsup3D excluding the neural renderer and resolved CUDA-related issues with gradient clipping. Our implementation achieves close performance to the original paper and the author's code trained on our environment. Code is available at: https://github.com/Team8-CS492-3DML/unsup3d-rep.

1. Introduction

Inferring the underlying 3D structure from images is a crucial aspect in the advance of computer vision techniques. In particular, many object in nature are deformable (*e.g.* animals, human faces) which makes it difficult to obtain static multi-view images for 3D reconstruction. This paper has been motivated by such challenges in collecting sufficient multi-view images of deformable objects and as a solution proposes a novel pipeline, photo-geometric autoencoding, for 3D reconstruction based on single-view images.

The proposed method, abbreviated as ***Unsup3D*** [13], takes a single image as input, and as the image is passed through the photo-geometric autoencoding pipeline it is decomposed into four factors: depth, albedo, viewpoint and illumination. By taking a combination of the decompositions it derives composite factors (i.e., normal, shading, canonical view) each of which imply an essential information of the underlying 3D structure of the input image. Then it utilizes an external neural renderer [6] to produce an output image, which is a reconstruction of the input image.

Unsup3D is expected to successfully reconstruct the input image under two major constraints [13]. First, it learns the 3D structures in an unsupervised manner, without referring to any ground truth 3D information or other 3D models. This allows to overcome the bottleneck of collecting labeled images and thus the model becomes highly applicable to images in the wild. Second, the model must be trained on datasets composed of only single-view images. As it learns to predict the 3D shape of an object from a single image this method can expand the domain of 3D reconstruction from static objects to include deformable objects.

2. Method Summary

To understand the underlying 3D structure from a given 2D image $I \in \mathbb{R}^{3\times H\times W}$ as input, *Unsup3D* decomposes the image into four visual elements: depth map $d \in \mathbb{R}^{1\times H\times W}$, albedo $a \in \mathbb{R}^{3\times H\times W}$, light direction $l \in \mathbb{S}^2$ and viewpoint $w \in \mathbb{R}^6$. The objective of *Unsup3D* is to find a function f that follows $f : I \rightarrow (a, d, l, w)$. Since f must be learned without any external supervision on the 3D information of the input, the structure of *Unsup3D* is designed in the form of an autoencoder, of which the *encoder* is given as a set of neural networks predicting (a, d, l, w) respectively, and the *decoder* is given as a neural rendering pipeline that outputs the reconstruction $\hat{I}$ of the input I when given the decomposition components (a, d, l, w).

2.1. Photo-geometric autoencoding

The overall process of decomposing an input image into four visual components *(i.e. depth, albedo, light, view)* and reconstructing the input from the decomposed elements is proposed as *photo-geometric autoencoding*. The pipeline consists of two main steps: *lighting* Λ and *reprojection* Π as follows:

$$\hat{I} = \Pi(\Lambda(a, d, l), d, w). \tag{1}$$

When an input image is passed through each neural network, we can derive the predictions for each decomposition component. Then the *lighting* function Λ takes albedo a, depth d, light l as input and computes a shading map $\in \mathbb{R}^{1\times H\times W}$, which shows the predicted shading of the input image from a canonical viewpoint $w = 0$. The shading map is then passed to the *reprojection* function Π along with depth d and viewpoint w, and in consequence the reconstruction $\hat{I}$ of the input image I is derived. The full pipeline is shown in Fig.1.

Symmetric Assumption. Since *Unsup3D* must not take any prior knowledge of the underlying 3D structure of the input images, the proposed pipeline is prone to degenerate decompositions (*e.g.* a flat depth map). To prevent the model from falling into such degenerate solutions, the concept of a *virtual second view* is introduced. Given an input I and its decompositions, the model can obtain a virtual second view from another angle by horizontally flipping the albedo a and depth d. Then the model passes the flipped albedo a' and depth d' through the same autoencoding pipeline and obtains a second reconstruction $\hat{I}'$ as follows:

$$\hat{I}' = \Pi(\Lambda(a', d', l), d', w),\ a' = flip\, a,\ d' = flip\, d. \quad (2)$$

By calculating the loss between the second reconstruction $\hat{I}'$ and original input I, we enforce the network to estimate the albedo a and depth map d in symmetric shape. It's why we call it as symmetric assumption.

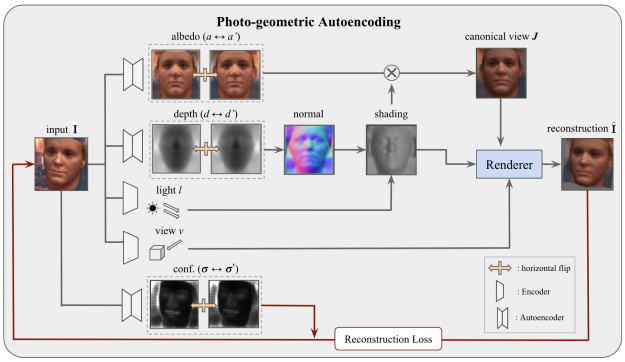

Figure 1. **Photo-geometric Autoencoding.**

2.2. Image formation pipeline

All images taken by camera are distorted since the camera gathers lights into a single point. To calibrate the distortion, we use camera projection matrix K which maps actual 3D points $P \in \mathbb{R}^3$ to 2D image's pixels $p = (u, v, 1)$. Here we assume the field of view (FOV) $\theta_{FOV} \approx 10°$.

$$p \propto KP,\ K = \begin{bmatrix} f & 0 & c_u \\ 0 & f & c_v \\ 0 & 0 & 1 \end{bmatrix},\ \begin{cases} c_u = \frac{W-1}{2}, \\ c_v = \frac{H-1}{2}, \\ f = \frac{W-1}{2tan\frac{\theta_{FOV}}{2}}. \end{cases} \quad (3)$$

After applying 3D rotation R and translation T which are estimated by the viewpoint network w, we can reconstruct original pixels by following function:

$$p' \propto K(d_{uv} \cdot RK^{-1}p + T), \quad (4)$$

The Neural Mesh Renderer [6] can substitute projection matrix K in the Eq. (4), to reconstruct input image without blanked pixels in foreground. You can check the details of mesh generation and image reconstruction in Appendix Sec. 6.3

As in the original work, we compute a tangent vector at (u, v) as $t_u^{uv} = d_{u+1,v} \cdot K^{-1}(p+e_x) - d_{u-1,v} \cdot K^{-1}(p-e_x)$. Then we can derive the normal vector one each pixel as $n_{uv} \propto t_{uv}^u \times t_{uv}^v$, which is called as normal map. The shading is derived by inner product of light direction l and normal vector with weighting coefficient. Finally, the canonical view J is obtained by multiplication of the shading and the albedo a. The light direction $l = \text{Norm}(l_x, l_y, 1)$ is an unit vector determined by two parameter l_x and l_y which is output of the network.

$$J_{uv} = (k_s + k_d \max\{0, < l, n_{uv} >\}) \cdot a_{uv} \quad (5)$$

Here the weight coefficients k_s and k_d are scalar weighting the ambient and diffuse light intensity and estimated by network.

2.3. Loss functions

Unsup3D uses a modified L1 loss as a reconstruction loss which is combined with confidence map. This loss drives the model to learn meaningful confidence map itself [7]. It can be also thought as the negative log-likelihood of a factorized Laplacian distribution on the L1 loss [13].

$$\mathcal{L}(\hat{I}, I, \sigma) = -\frac{1}{|\Omega|} \sum_{uv\in\Omega} ln \frac{1}{\sqrt{2}\sigma_{uv}} exp - \frac{\sqrt{2}l_{1,uv}}{\sigma_{uv}}, \quad (6)$$

We used the same reconstruction loss metric on both original and flipped cases. The weight between two losses are set as $\lambda_f = 0.5$.

$$\varepsilon(\Phi; I) = \mathcal{L}(\hat{I}, I, \sigma) + \lambda_f \mathcal{L}(\hat{I}', I, \sigma'), \quad (7)$$

Additionally, perceptual loss is utilized to mitigate sensitivity on small geometry imperfection and blurry reconstruction [13]. The perceptual loss indicates the distance of intermediate feature on visual classifier. The feature from a single layer *relu3_3* of pre-trained VGG16 [12] is used here.[1]

[1] we used pre-trained model provided in *torchvision* https://pytorch.org/vision/stable/models.html

$$\mathcal{L}_p^{(k)}(\hat{I}, I', \sigma^{(k)}) = -\frac{1}{|\Omega_k|} \sum_{uv \in \Omega_k} ln \frac{1}{\sqrt{2\pi(\sigma_{uv}^{(k)})^2}} exp - \frac{(l_{uv}^{(k)})^2}{2(\sigma_{uv}^{(k)})^2}, \tag{8}$$

where $l_{uv}^{(k)} = \left| e_{uv}^{(k)}(\hat{I}) - e_{uv}^{(k)}(I) \right|$ for each pixel uv in layer k. By replacing the loss function $\mathcal{L}$ in Eq. (7) into $\mathcal{L}+\lambda_p\mathcal{L}_p$ where $\lambda_p = 1$, we can get the learning objectiveΩ.

3. Implementation Details

Based on the *photo-geometric autoencoding* pipeline shown in Fig.1, we implemented every necessary component except for the Neural Mesh Renderer. We borrowed some parts of the dataloader for the BFM dataset in order to accurately load the ground truth depth from .png files.

Neural Mesh Renderer. We imported an external library for neural rendering based on the *Neural Mesh Renderer* (NMR) from [6]. Since the official library[2] is not compatible to newer versions of PyTorch, we have utilized a revised version[3] which ensures the compatibility for PyTorch 1.5+.

Image Preprocessing. We used the official train/test/val splits for both **BFM** and **CelebA**. For **CelebA**, we cropped and resized the raw image based on author's predefined face bounding box. For **BFM**, we center cropped 75% of image to reduce the background. All images are resized to (64,64) and normalized to have pixel value range between (-1,1).

Gradient Clipping. For both the author's code and our implementation, there was a major issue of gradient explosion when training the model with CUDA 11.3 and RTX3090. (See Appendix for details). To mitigate this issue, we clipped large gradient during training which prevents the model falling in degenerated solution.

4. Experimental Results

4.1. Setup

Our implementation of *Unsup3D* was mainly trained and evaluated based on PyTorch 1.11.0, CUDA 11.3 with a GPU instance of GeForce RTX 3090. We trained the model using the Adam optimizer by setting the batch size as 64, learning rate as 1e-4, $(\beta_1, \beta_2) = (0.9, 0.999)$, and weight decay as 5e-4, equal to the settings in the original paper [13]. We utilized the default initialization method from PyTorch [10] for the neural networks. Gradient clipping is by default applied in training for both ours and author's implementations.

[2] https://github.com/daniilidis-group/neural_renderer

[3] https://github.com/adambielski/neural_renderer

We trained the model for 30 epochs (i.e. around 70k steps) and each training process took 12 to 18 hours depending on GPU.

Datasets For training and testing we used **BFM [11], CelebA [8]** datasets. The original BFM dataset consists of human faces without background. As in the original paper, we used a synthetic dataset which is produced by adding random backgrounds to each image in the BFM dataset. We used the download links provided by the author[4] to obtain the CelebA dataset and the synthesized version of BFM dataset. BFM consists of 200k images (train: 160k, test: 20k, val: 20k)[5], and CelebA consists of over 200k images (train: 162k, test: 20k, val: 20k).
The original paper has conducted experiment on **3DFAW** [4] [5] [15] [16], synthetic cars from **ShapeNet** [2] and **cat datasets** [9] [14]. However, we could not train our implementation on these datasets due to the schedule delay from CUDA-related issues during training. Instead, we trained **CelebA** as a representative of an in-the-wild dataset and **BFM** as a representative of synthetic dataset which has ground truth 3D information.

Metrics For quantitative evaluation of our model, we used **SIDE** (scale-invariant depth error) [3] and **MAD** (mean angle deviation). After predicting the canonical depth map d, the pixels in the depth map are warped with respect to the predicted view w to obtain a new depth map $\bar{d}$, which shows the depth from the original viewpoint. SIDE measures the difference between $\bar{d}$ and the ground truth depth map by the formula $E_{SIDE}(\bar{d}, d^*) = (\frac{1}{WH}\sum_{uv}\Delta_{uv}^2 - (\frac{1}{WH}\sum_{uv}\Delta_{uv})^2)^{\frac{1}{2}}$, where $\Delta_{uv} = \log \bar{d}_{uv} - \log d_{uv}^*$. MAD is calculated by comparing the normal map predicted depth map $\bar{d}$ and the normal map derived from the ground truth depth map. Both SIDE and MAD are only calculated on the BFM dataset since it provides the ground truth depth.

4.2. Results

All qualitative and quantitative results presented in this section are based on the test sets, which are *unseen* data for the neural networks in our model. Thus if the results are promising, it implies that the model is showing a satisfying level of generalization.

Comparison with Baselines Tab. 1 shows the baseline performance on the **BFM** dataset to compare the quality of reconstructed output. **Row(1)** is a fully-supervised baseline trained with L1 loss. **Row(2)** and **row(3)** are trivial

[4] https://github.com/elliottwu/unsup3d/tree/master/data

[5] Due to broken images, we discarded 4 from train images and 3 from val images for training.

No.	Baseline	SIDE ($\times 10^{-2}$)	MAD ($deg.$)
(1)	Supervised (rep)	0.410 ±0.103	10.78 ±1.01
(2)	Const. null depth (rep)	2.723 ±0.371	43.34 ±2.25
(3)	Average g.t. depth (rep)	1.990 ±0.556	23.26 ±2.85
(4)	Ours (unsupervised)	1.112 ±0.275	17.93 ±2.40

Table 1. **Comparison with baselines.** for (rep) means that the value is from reported in original paper. For detail, refer to Sec. 4.2

baselines estimating a fixed uniform depth and an average ground truth depth of the test dataset. Our unsupervised method outperforms the trivial baselines for both SIDE and MAD metrics.

Qualitative Results Fig. 2 shows that our *Unsup3D* model can successfully reconstruct the 3D shape of human faces including the details of eyes and nose. The *rotated views* are rendered by a Neural Mesh Renderer [6] with different viewpoints given as input. See Fig. 5 and Fig. 6 for more qualitative results of image decomposition and input reconstruction.

Ablation Study Tab. 2 shows the ablation results for each module of our *Unsup3D* implementation. **row(3)** and **row(4)** show that depth and albedo flipping are both necessary components during training. Especially, removing albedo flip has led to a severe degradation in performance, since the model cannot predict the shading information without a symmetric assumption on the albedo. Removing the perceptual loss shows slight degrades reconstruction quality. **Row(6)** uses self-supervised VGG16 [1] on perceptual loss instead[6] and shows slight worse performance. **Row(7)** uses constant uniform confidence map and you can check it is degenerated. You can check qualitative comparison of ablation from Fig. 4 in Appendix.

Comparison with Author's Code We conducted experiment on ours with 75% center crop and author's with 66% center crop on **BFM** dataset, since the data pre-processing codes are different. As shown in Sec. 4.3, ours and author's show similar results on **MAD**, but ours is slightly worse than author's on **SIDE** since we our model was train on a wider image, thus needing to handle a wider background. Both results are worse than the original paper, especially for **SIDE**. The difference is oriented from issues on CUDA 11.3 and the crop size. (See Sec. 6.1 for details) Auxiliary hyper-parameter tuning could possibly improve the model's performance on CUDA 11.3.

[6] https : / / github . com / facebookresearch / DeeperCluster

No.	Method	SIDE ($\times 10^{-2}$)	MAD ($deg.$)
(0)	Reported	0.793 ±0.140	16.51 ±1.56
(1)	Author's full	0.932 ±0.213	17.75 ±1.99
(2)	Ours full	1.112 ±0.275	17.93 ±2.40
(3)	w/o albedo flip	2.462 ±*0.429*	36.68 ±1.82
(4)	w/o depth flip	1.771 ±0.356	37.48 ±2.75
(5)	w/o perc. loss	1.119 ±0.262	19.36 ±2.45
(6)	w/ self-sup. perc. loss	1.316 ±0.235	18.72 ±2.43
(7)	w/o confidence	6.173 ±0.676	68.35 ±5.99

Table 2. **Ablation study.** the value written behind ± is standard deviation of error. Refer to Sec. 4.2

4.3. Limitations

Our implementation of *Unsup3D* shows better performance compared to the ablated models as in Sec. 4.2. However, the SIDE error of our model is relatively higher than the original paper. In addition, when trained without the confidence network our model showed degeneration for depth and albedo, whereas the author's implementation has shown moderate accuracy even without confidence. Thus our version of *Unsup3D* is highly dependent on the existence of confidence compared to the original paper.

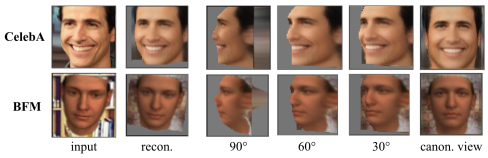

Figure 2. **Reconstruction examples of BFM and CelebA images.**

5. Conclusion

With this project, we implemented the *Unsup3D* from scratch and trained the model on BFM and CelebA dataset. We successfully reproduced results closed to the reported one. We also found out gradient explosion occurs during the training in CUDA 11.3 and resolved it through adding gradient clipping on training. However, gradient clipping is just a temporary expedient to train model so additional study should be conducted about it.

Acknowledgements We would like to thank Minhyuk Sung for providing us insightful advice on critical issues on training. We borrowed a modified NMR from Adam Bielski[7]. Yuseung Lee and Inhee Lee contributed equally on implementation, data processing, report and poster.

[7] https://github.com/adambielski/neural_renderer

References

[1] Mathilde Caron, Piotr Bojanowski, Julien Mairal, and Armand Joulin. Unsupervised pre-training of image features on non-curated data, 2019. 4

[2] Angel X. Chang, Thomas A. Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qi-Xing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository. *CoRR*, abs/1512.03012, 2015. 3

[3] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 2366–2374, Cambridge, MA, USA, 2014. MIT Press. 3

[4] Ralph Gross, Iain Matthews, Jeffrey Cohn, Takeo Kanade, and Simon Baker. Multi-pie. *Image and Vision Computing*, 28(5):807–813, 2010. Best of Automatic Face and Gesture Recognition 2008. 3

[5] László A. Jeni, Jeffrey F. Cohn, and Takeo Kanade. Dense 3d face alignment from 2d videos in real-time. In *2015 11th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*, volume 1, pages 1–8, 2015. 3

[6] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer, 2017. 1, 2, 3, 4

[7] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision?, 2017. 2

[8] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015. 3

[9] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. Cats and dogs. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3498–3505, 2012. 3

[10] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 3

[11] Pascal Paysan, Reinhard Knothe, Brian Amberg, Sami Romdhani, and Thomas Vetter. A 3d face model for pose and illumination invariant face recognition. In *2009 Sixth IEEE International Conference on Advanced Video and Signal Based Surveillance*, pages 296–301, 2009. 3

[12] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014. 2

[13] Shangzhe Wu, Christian Rupprecht, and Andrea Vedaldi. Unsupervised learning of probably symmetric deformable 3d objects from images in the wild, 2019. 1, 2, 3, 7

[14] Weiwei Zhang, Jian Sun, and Xiaoou Tang. Cat head detection - how to effectively exploit shape and texture features. In David Forsyth, Philip Torr, and Andrew Zisserman, editors, *Computer Vision – ECCV 2008*, pages 802–816, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. 3

[15] Xing Zhang, Lijun Yin, Jeffrey F. Cohn, Shaun Canavan, Michael Reale, Andy Horowitz, and Peng Liu. A high-resolution spontaneous 3d dynamic facial expression database. In *2013 10th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*, pages 1–6, 2013. 3

[16] Xing Zhang, Lijun Yin, Jeffrey F. Cohn, Shaun Canavan, Michael Reale, Andy Horowitz, Peng Liu, and Jeffrey M. Girard. Bp4d-spontaneous: a high-resolution spontaneous 3d dynamic facial expression database. *Image and Vision Computing*, 32(10):692–706, 2014. Best of Automatic Face and Gesture Recognition 2013. 3