Differentiable Volumetric Rendering: Learning Implicit 3D Representations without 3D Supervision Development track

Hyunho Yang
KAIST
haebo1@kaist.ac.kr

Sungbin Lee
KAIST
dltjdqls@kaist.ac.kr

Abstract

This paper introduces a way of learning implicit 3D representations without 3D supervision, so called "Differentiable Volumetric Rendering". The paper shows that we can derive the gradients of predicted depth map. This enables differentiable rendering of implicit representation and eventually allows us to learn implicit representation without 3D ground truth. The main challenge was the time taken when we run the model with depth information obtained by SfM. It prevented us from running the experiments repeatedly. So we modified the way of handling depth information obtained by SfM and could improve it. As single-view reconstruction takes 2 to 3 weeks for running, we only implemented multi-view reconstruction for feasibility. We could obtain the results that outperform the original work and could obtain it with lesser time thanks to the improvement we mentioned. Code is available at: https://github.com/HyunHo99/Group17-dvr

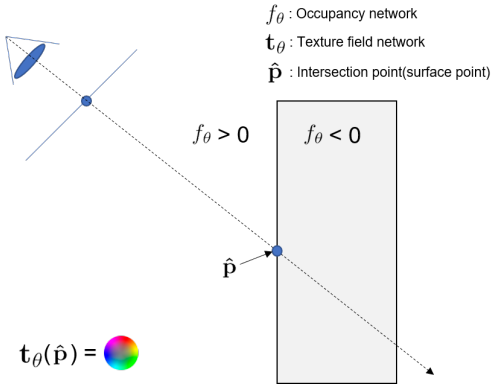

Figure 1. Overall process of our method. Using the occupancy network f_θ and differentiable ray casting, find the intersection $\hat{p}$. Then, pass the position value to texture field network t_θ to get RGB color value. Eventually, we can train the networks using RGB images(Optionally, we can use depth images also) to get 3D geometry and texture of a scene.

1. Introduction

3D reconstruction is a representative subject of computer vision. And learning-based 3D reconstructions have shown impressive results recently [3, 4, 7]. However, most approaches required 3D ground truth. And as 3D ground truth data are hard to obtain in real life, they were often limited to synthetic data.

To solve this, recent works investigate a way that can obtain 3D reconstruction with only 2D supervision. This can be achieved by making the rendering process differentiable and it shows compelling results [2, 5]. However, it is limited to specific 3D representations like voxels or meshes that suffer from low resolution(by computational cost limit). In terms of this limitation, implicit representation can be helpful as it requires constant memory usage. But again, current approaches with implicit representation require 3D ground truth.

To overcome those limitations, this paper introduces a way to learn implicit 3D representations with only 2D supervision. With this approach that doesn't require 3D ground truth, we can learn 3D representations of real world data that can be applied to more practical situations. The model outputs occupancy network and texture network. They predict the resulting image and learn the network with the loss between predicted image and ground truth image. At this point, if a ground truth depth is also provided, model can choose to learn with the loss between predicted depth and ground truth depth directly. And that part using ground truth depth was main challenging point. Specifically, learning with depth information obtained by SfM took almost 1 day and it prevented us from experimenting repeatedly. So we modified the way of handling depth information obtained by SfM. With this modification, we reduced the time by half and also the result outperformed the original work.

In addition to this challenge, implementing backward pass operation and the sampling algorithm for obtaining intersection via ray casting were also tricky and challenging. Despite those challenges, we made it to implement the whole multi-view reconstruction part of the original work. And as the marching cube part is originally from the baseline [6], we borrowed that part from it. For the experimental results, we used original work's dataset metrics and obtained comparable evaluation results.

In summary, our achievements are:

- reimplement "Differentiable Volumetric Rendering" on our own from scratch
- modify the way of handling ground truth depth obtained by SfM and got improvements in terms of the time and result.

2. Method Summary

For a 3D scene, we use two networks: occupancy network, texture field network. Occupancy network

$$f_\theta : \mathbb{R}^3 \to \mathbb{R} \tag{1}$$

describes probability of occupancy of a point in the scene. In our implementation to simplify ray casting calculation, we use real value codomain. Negative output implies the point is inside of the object, positive output implies the point is outside of the object. Texture field network

$$t_\theta : \mathbb{R}^3 \to \mathbb{R}^3 \tag{2}$$

describes the RGB color value of points in the scene.

Our problem is, from given 2D images of a scene with corresponding camera parameter, object mask and depth(optionally), training f_θ and t_θ. To train networks for the 3D scene using 2D images, we use differentiable ray casting.

Let a pixel of given image I as u, the origin of camera as r_0. Then we can write a camera ray r_0 to u as following

$$r(x) = r_0 + wx \tag{3}$$

where $w = u - r_0$. We can find a first intersection with the object and the ray, and we will denote the intersection as $\hat{p} = r(\hat{d})$ where $\hat{d}$ is depth value. We will explain about intersection finding method in more detail on implementation detail part.

In backward process, differentiable ray casting take gradient λ as input, and it should return $\lambda \frac{\partial \hat{d}}{\partial \theta}$. To calculate $\frac{\partial \hat{d}}{\partial \theta}$, differentiate both side of $f_\theta = c$ with respect to θ. Then, since $\hat{p} = r_0 + w\hat{d}$, we obtain

$$\frac{\partial f_\theta(\hat{p})}{\partial \theta} + \frac{\partial f_\theta(\hat{p})}{\partial \hat{p}} w \frac{\partial \hat{d}}{\partial \theta} = 0 \tag{4}$$

Eventually, we can get

$$\frac{\partial \hat{d}}{\partial \theta} = -\left(w \frac{\partial f_\theta(\hat{p})}{\partial \hat{p}} \right)^{-1} \frac{\partial f_\theta(\hat{d})}{\partial \theta} \tag{5}$$

Since f_θ is explicitly formulated MLP, we can simply calculate $\frac{\partial f_\theta(\hat{d})}{\partial \theta}$ and $\frac{\partial f_\theta(\hat{p})}{\partial \hat{p}}$. In this way, we can construct differentiable ray casting module. Overall process illustrated on figure 1.

3. Implementation details

We borrowed the code for mesh extraction using occupancy network from [6]. To adopt this code, we map the output of f_θ to [0,1] using Bernoulli distribution to match with original occupancy output and assign the color to each vertex of mesh using t_θ.

3.1. Network detail

We implement f_θ and t_θ as a single network. The network takes 3D position as input, and passes to linear network with ReLU activation. The linear network outputs 512 dimension features then passes to 5 ResNet blocks with ReLU activation and 512 hidden dimension size. Finally, last linear layer outputs 4 dimension features. We divide this into 2 features: 1 dimension for occupancy and 3 dimension for texture. For a feature for texture, apply sigmoid function.

3.2. Loss detail

In this part, we will describe loss used for training and how to calculate the loss. Actually, some of the original paper's explanation is not matched with the official code, so following explanations are based on the code, not the paper.

$$\mathcal{L}_{rgb} = \sum_{u \in U} \left\| I_u - \hat{I}_u \right\|_1 \tag{6}$$

Where I_u is color value of Image I at pixel u, I is ground truth image, $\hat{I}$ is predicted image and U is set of randomly sampled pixels inside of the object mask.

$$\mathcal{L}_{freespace} = \sum_{p \in P_1} BCE(f_\theta(p), 0) \tag{7}$$

$$\mathcal{L}_{occupancy} = \sum_{p \in P_2} BCE(f_\theta(p), 1) \tag{8}$$

Where BCE is binaray cross entropy, P_1 is the set of random 3D points outside of the object and P_2 is the set of random 3D points inside of the object.

$$\mathcal{L}_{normal} = \sum_{u \in U} \| n(\hat{p}_u) - n(q_u) \|_2 \tag{9}$$

Where $n()$ is surface normal at that point, $\hat{p}_u$ is predicted surface point at pixel u and q_u is random neighbor within a distance of 1e-3 from $\hat{p}_u$.

$$\mathcal{L}_{depth} = \sum_{u \in U_d} \left\| r(d_u) - r(\hat{d}_u) \right\|_2 \tag{10}$$

This depth loss is active when ground truth depths are given. Where d_u is ground truth depth at u and $\hat{d}_u$ is predicted depth at pixel u. If MVS depth is given, U_d is simply same as U. However, with SfM depth, letting U_d be same as U results poor training because SfM ground truth is too sparse. It means, if we just randomly sample the pixels inside of the object mask, most of the pixels do not have SfM ground truth depth.

This is our main difference from the original code. Original code just set U_d as *all* of the pixels that can matched with SfM depth and do not calculate the L_{rgb} and L_{normal} using the U_d. In practice, the size of the original U_d is 3000 on average. So it does more than twice the ray casting calculation than the other. Instead, we use novel sampling method when we have SfM depth. Instead that U_d as all of the pixels that can matched with SfM depth, replace U as following: 1/4 of pixels are sampled from the pixels that can matched with SfM depth, 3/4 of pixels are sampled inside of the object mask. Then use the U to calculate $\mathcal{L}_{rgb}$, $\mathcal{L}_{normal}$ and $\mathcal{L}_{depth}$.

3.3. Ray Casting detail

To find the intersection, we sample points on a ray and calculate the occupancy. We construct unit cube($[-1, 1]^3$) enclosing an object and use it to more efficient sampling. We find intersections between the ray and the unit cube, and if there are exactly two intersections: $r(d_1)$ and $r(d_2)$, then uniformly sample n sample sets on range $[d_1, d_2]$. If there are not exactly two intersections, then uniformly sample n sample sets on range $[0, M]$. With obtained sample set, we will denote the points in the set as x_i, and $x_0 < x_1 < ... < x_n$. Then, find

$$\min \{k : f(x_{k+1}) < 0 \wedge f(x_k) > 0\} \tag{11}$$

Finally, calculate $\hat{d}$ using secant method with x_k and x_{k+1}.

4. Experimental Results

As we mentioned, we conduct experiments for multi-view reconstruction since training model for single-view reconstruction in the original paper takes too long time(2 weeks$\sim$3 weeks). We conducted our experiments with DTU MVS dataset [1] scene skull, angel, birds as original paper does. The original paper's author manually removed light inconsistent images. We also discarded those images for pair comparison. MVS depth obtained by Colmap [8] with camera parameters, and SfM depth obtained by Colmap without camera parameters. We train our model with following configures: maximum depth M = 1400, size of U = 2048, sample size n start with 16 and doubling after 50K, 150K, 250K iterations, Adam optimizer with learning rate 10^{-4}. Finally, We use Chamfer-L1 distance to evaluate results. Accuracy is mean of minimum distance from predicted surface to ground truth surface, Completeness is mean of minimum distance from ground truth surface to predicted surface,

	Accuracy	Completeness	Chamfer-L1↓
Original W/O depth	1.270	0.849	1.060
Ours W/O depth	1.252	1.028	1.140
Original MVS depth	1.049	0.974	1.012
Ours MVS depth	1.059	0.824	0.946

Table 1. Results for "skull" scene

	Accuracy	Completeness	Chamfer-L1↓
Original W/O depth	1.155	0.743	0.949
Ours W/O depth	1.019	0.859	0.939
Original MVS depth	0.722	0.718	0.720
Ours MVS depth	0.762	0.726	0.744

Table 2. Results for "birds" scene

	Accuracy	Completeness	Chamfer-L1↓
Original W/O depth	0.738	0.687	0.712
Ours W/O depth	0.636	0.598	0.617
Original MVS depth	0.595	0.633	0.614
Ours MVS depth	0.609	0.638	0.623

Table 3. Results for "angel" scene

	Accuracy	Completeness	Chamfer-L1↓
Original W/O depth	1.054	0.760	0.907
Ours W/O depth	0.969	0.828	0.899
Original MVS depth	0.789	0.775	0.782
Ours MVS depth	0.810	0.730	0.770
Original SfM depth	0.940	0.821	0.881
Ours SfM depth	0.801	0.798	0.800

Table 4. Effect of depth supervision(average of 3 scenes)

4.1. Comparison with Original Paper

Table 1,2,3,4 shows quantitative results and figure 2 shows result images. Our results show only a small difference from the original results, and show even better results on average. Our implementation has same tendency with respect to normal loss(Figure 3). Moreover, our im-

Figure 2. Result images. Official images are made by pre-trained model that given by author. Our code generate comparable results.

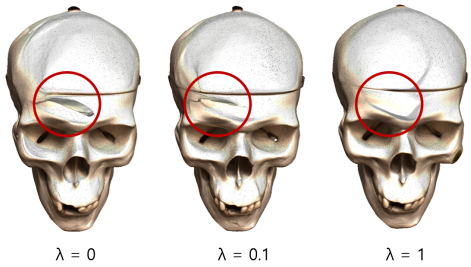

Figure 3. Effect of normal loss. If we increase of the weight(λ) of $\mathcal{L}_{normal}$, a surface becomes smooth.

plementation of SfM depth takes half the training time than the original one. You can see in Figure 4 that our results are much more precise. In our opinion, the outperforming is resulted by letting U_d to participate to the other loss such as normal loss, so it was overfitted well to not only SfM ground truth points but also neighbors.

4.2. Failure case

As figure 5 shows, in some results, unnecessary growth was observed. It seems like limitation of surface based rendering. Since it intersects to only one point for a ray, it tends to easily fall into local minimum and it results unnecessary growth.

5. Conclusion

Differentiable Volumetric Rendering made it possible to learn implicit 3D representation without requiring 3D supervision. We could successfully get comparable experimental results to original work by re-implementing it from scratch. Also by modifying the way of handling ground truth depth information obtained by SfM, we could get some improvements in running time and results.

Figure 4. Result images from model trained with SfM depth supervision. Official images are made by pre-trained model that given by author. Our code trained 2X faster and generate outperform result than original code.

Figure 5. Failure case. In our result of angel with MVS depth, unnecessary growth was observed.

6. Acknowledgements

Hyunho Yang: Implement code and conducted experiments. Wrote method summary, implementation details and experimental results.
Sungbin Lee: Wrote abstract, introduction and conclusion. Made paper poster
We borrowed marching cube code from official occupancy network github. We downloaded DTU MVS dataset from official original code github. We used official DTU MVS evaluation code.

References

[1] Henrik Aanæs, Rasmus Ramsbøl Jensen, George Vogiatzis, Engin Tola, and Anders Bjorholm Dahl. Large-scale data for multiple-view stereopsis. *International Journal of Computer Vision*, 120(2):153–168, Nov 2016. 3

[2] Wenzheng Chen, Jun Gao, Huan Ling, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to predict 3d objects with an interpolation-based differentiable renderer. In *Advances In Neural Information Processing Systems*, 2019. 1

[3] 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 *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016. 1

[4] Haoqiang Fan, Hao Su, and Leonidas J. Guibas. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 1

[5] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 1

[6] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space, 2018. 2

[7] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 1

[8] Johannes L. Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 3