Improved DeepVoxels: Better Grid Boundary Initialization and Reduced Number of Training Views

Zheng Yuan
SCPD
Zheng.yuan.nicolas@gmail.com

Abstract

DeepVoxel is a state of art technique in 3D reconstruction. It combines the classical camera geometry with deep learning methods elegantly and thus is worthy good attention in the spirits of this class. In the original paper, the authors use empirical numbers to assign value of two important parameters in the setup: voxel grid boundary and the number of training views. Yet, inspired by the homework in the class, I leverage depth map of each view to initialize the voxel grid with a tighter boundary. Also, I propose two subsampling methods to reduce the number of training views. Experiments show that the improved voxel boundary can result in better rendering of new views. They also show the model trained with a smaller number of views can deliver comparable quality, which suggests the view subsampling methods can reduce redundancy without losing information.

https://github.com/yuanzheng625/ImprovedDeepVoxel

1. Introduction

3D reconstruction is a very popular topic in computer vision studies due to its wide applications in robotics, entertainment, virtual tourism, artifact preservation, augmented reality, human computer interaction, animation etc. In many settings, it is assumed that we access a set of 2D images that are captured with known camera intrinsic and extrinsic parameters for each view. In the traditional methodology, however, correspondences across views are required to be established before applying bundle adjustment for the scene reconstruction. It is well known that finding correspondences is not only technically challenging but also computationally expensive. Thankfully, the new generation of 3D reconstruction algorithms successfully bypass this challenge: with the power of deep neural networks, the correspondence can be implicitly encoded within network weights, which we can learn through a lot of data. In the meantime, the 3D scene is also represented in feature vectors, rather explicit geometry.

DeepVoxel [1], published in CVPR 2019, is an exemplar deep learning method to approach the 3D reconstruction problem. It combines scene reconstruction from multiple view images with volumetric stereo. The whole pipeline is summarized as follows: given one 2D view, first it goes through a feature extraction network to get the 2D features that represent this view. Then the 2D features are passed through another neural network to infer the corresponding 3D voxel features. In particular, the positional mapping from 2D view features to the 3D voxel features is regulated by camera projection model. In fact, the 3D voxel feature so far comes from only a single 2D view, but we want to complete the 3D voxel feature through all available views. Therefore, the authors frame this 3D voxel feature as an update to the persistent 3D voxel features (hidden state) and use recurrent neural network (GRU) to integrate the update from multiple views. Then the 3D voxel features are mapped to another 2D feature from a different view (target). Again, the positional mapping is regulated by camera projection model. Finally, the loss is calculated by comparing the new projected view with its ground truth.

The reasons that make me select DeepVoxel as baseline are that 1) Not entirely as a black box deep learning model, it utilizes the camera projection geometry we learned firsthand from the class to constrain neural networks that bridge 2D view feature to its 3D voxel features. 2) The method integrates multiple views in the same fashion as in volumetric stereo (e.g. space carving) to refine the 3D persistent voxel feature, thus I can apply the technique in space carving [4] to the baseline.

Also, as all learning-based methods prefer, we want to explore what is the smallest amount of required data and how to use them efficiently to train a DeepVoxel model. Thus, I proposed two subsampling methods to reduce the number of 2D views for training and compare their performance with baseline.

2. Approach

2.1. Voxel grid boundary

Based on the paper and also the code [3] provided by the authors, the voxel grid boundary is $-1 < x < 1$, $-1 < y < 1$ and $-1 < z < 1$ in the voxel grid coordinate system. By placing the barycenter of the 3D objects approximately at the center of the world coordinates system, the grid system aligns with the world coordinates system but just off a translation and scale. With the world coordinate system and grid coordinate system unchanged, we argue that we can push the voxel grid boundary in the grid system a bit tighter.

The method is to introduce depth map for each view. In the dataset, the authors do provide the depth map (Fig. 1) for each view. In realty, when taking each 2D view, we can add a depth sensor to get the complementary depth image.

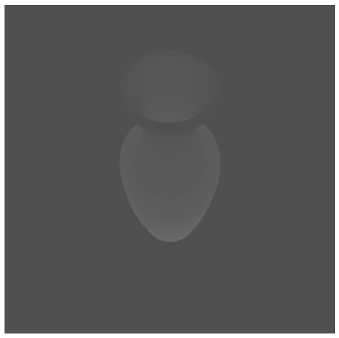
Fig.1 depth map of a 2D view in synthetic vase dataset

Therefore, for each pixel in one view, we can get its 3D position in the world system via,

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = M^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

where M^{-1} is the camera matrix of the view.

Then the 3D point in the grid coordinate system is,

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = H \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

where H is the transformation matrix from world system to grid system.

After computing all 3D points for one view in the synthetic vase dataset, we visualize it in Fig. 2.

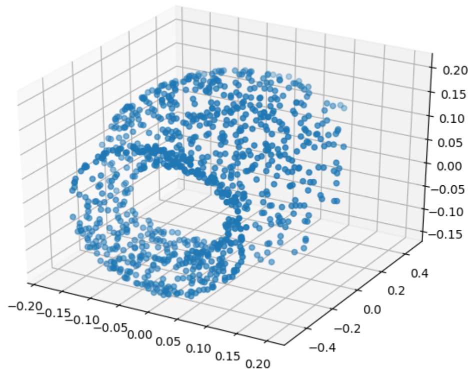
Fig. 2 All 3D points corresponding to one view

When we integrate all the 3D points from all views, we can get a much complete view of the whole 3D scene, as show in in Fig. 3.

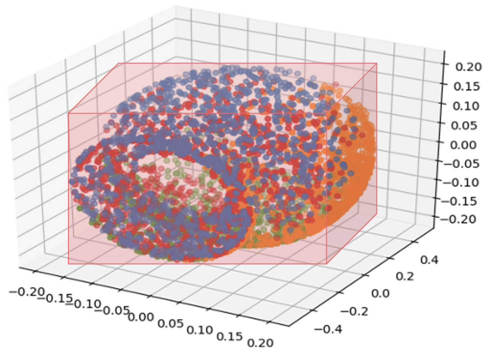
Fig. 3 All 3D points from multiple views and the bounding box that enclose all 3D points

Note that the authors make the cube assumption hardcoded in their implementation. To allow generic rectangle 3D voxel grids, I reimplement the mapping function between a 2D view and its mask on the 3D voxel grid.

2.2. 2D view subsampling

In the paper, the authors use about 300 to 400 views to train a specific 3D object. As shown in Fig. 4, the camera army forms a sphere like array that encompasses the object.

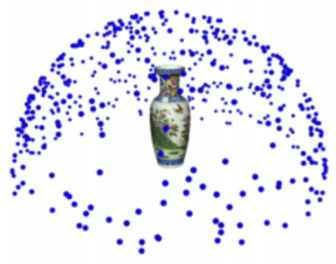
Fig. 4 A camera army surrounding the scene

Fig. 5 is a visualization of each camera position and orientation in the army. We can observe that the cameras forms a dense sphere, yet the neighbor views should cover significant overlapping areas. Therefore, we could subsample only one or a few among neighboring cameras to reduce redundancy (method 1).

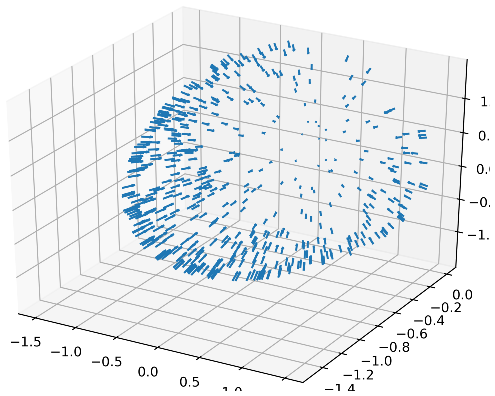

Fig. 5 Visualization of camera pose and position

Also, starting from the 3D object center, the cameras locate gradually away from it. Thus we can subsample cameras in different distance from the object center. Just like a satellite covering on the earth: the higher of the satellite, the more coverage it has but with reduced resolution. So it is a trade-off between coverage and details. We propose method 2 to sample cameras along the distance.

Method 1: I use convex hull algorithm (implementation in SciPy) to get the shell of the camera army sphere. The byproduct of this is the facets that every three vertexes lie on. This is an ad hoc to group different cameras into neighboring clusters. The subsampling method is as follows:

a) Initialize two sets Q0 and Q1, where Q1 is the set composed of camera views we keep after subsampling and Q0 is the set composed of camera views we discard
b) for each facet on the convex hull, get the three vertexes as candidates
c) remove the candidates that are already in Q0 and Q1
d) if there are still candidates left, sample one of them and put it into Q1 and the rest into Q0
e) return Q1 as the sampled camera views

This procedure can be done multiple times to generate different level of sparse camera views, shown in Fig. 6.

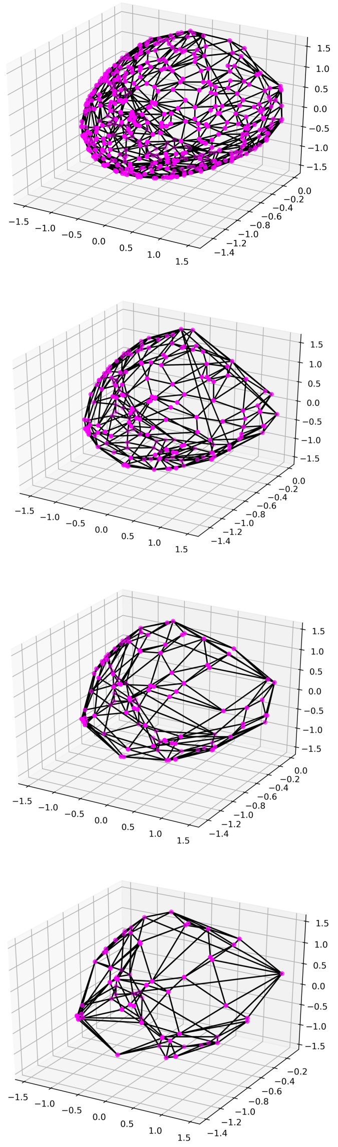

Fig. 6 convex hull (each vertex is a subsampled camera) of 479, 232, 142 and 67 views

Method 2: To subsample cameras according to the distance of each camera from the object center, I use multi-layer convex hull to approach this problem. For all cameras, we regard the cameras on the convex hull as the most outside ones. They form the layer with largest distance. Then we remove them and get the convex hull of the remaining cameras. The cameras on the new convex hull form the second layer and so on. Essentially this process is like peeling an onion, which provides us an effective way to traverse all the cameras. See Fig. 7.

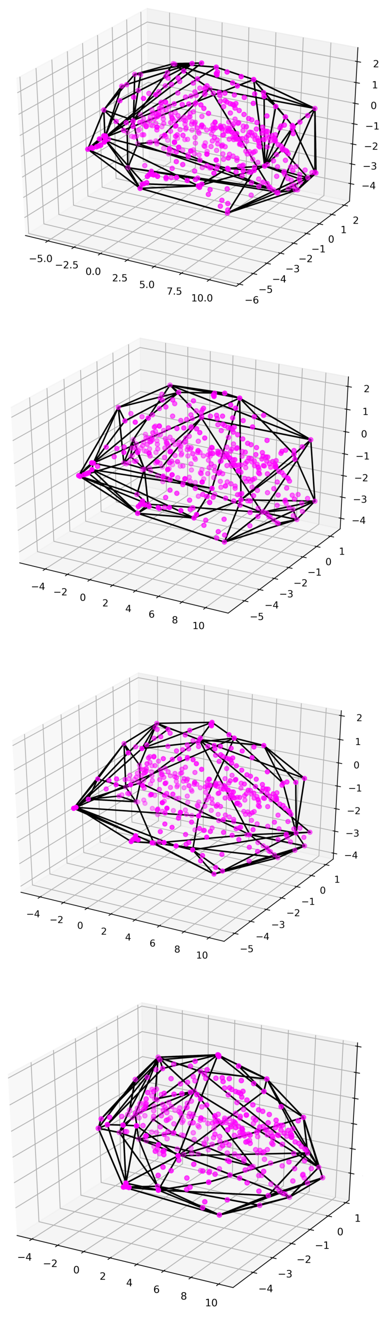

Fig. 7 convex hull (each vertex is a camera) with different distant from object center

3. Experiments

Experiment 1: Tighter voxel boundary initialization

The dataset we are using is the synthetic vase dataset (link) provided by the author. By running the boundary estimate code [5], we have the voxel grid boundary as $-0.205 < x < 0.205$, $-0.500 < y < 0.500$, $-0.204 < z < 0.204$. With the volume of voxel decrease from 8 to 0.16 (50 times), we can increase voxel resolution by 50 while maintaining the same level of computation.

Fig. 8 shows the model visual performance trained with better voxel grid. Comparing with baseline, the new render view (the third column) can recover more details to the ground truth. This is due to the voxel grid represent the scene more efficiently.

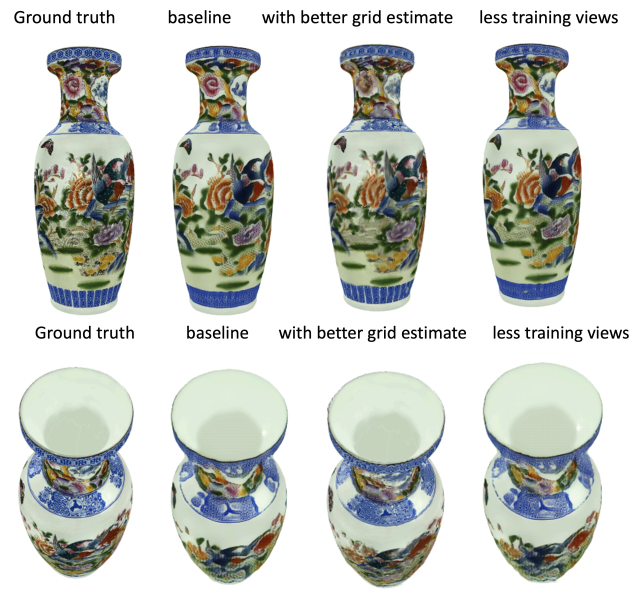

Fig. 8 Visual performance of the two proposed improvements. From left to right: ground truth of a new view, new view rendered by baseline DeepVoxel, new view rendered by improved voxel grid boundary and new view render by model trained with reduced number of views.

Experiment 2: Reduced number of training views

I subsample the cameras with 142 views using method 1 in Sec. 2.2. The remaining training parameters are the same as the baseline. As shown inn Fig. 8, the new render view (the fourth column) has comparable visual performance as the baseline.

4. Conclusion

In this project, I make two improvements over the baseline DeepVoxel. Inspired by the space carving method in the class, I better estimate the voxel grid boundary and thus learn a more efficient model to render new views. Also, I reduce the redundancies across camera views and subsample them to retrain the model. The experiments show that the proposed subsampling ad hoc results in comparable visual performance with the baseline but using a much smaller number of training views.

References

[1] Sitzmann Vincent, Thies Justus, Heide Felix, Niesner Matthias, Wetzstein Gordon and Zollhofer, Michael, DeepVoxels: Learning Persistent 3D Feature Embeddings, Proc. Computer Vision and Pattern Recognition (CVPR), IEEE, 2019

[2] Hartley Richard and Zisserman Andrew, Multiple View Geometry in Computer Vision, 2003, Cambridge University Press, isbn 0521540518.

[3] https://github.com/vsitzmann/deepvoxels

[4] http://web.stanford.edu/class/cs231a/lectures/lecture8_volumetric_stereo.pdf

[5] https://github.com/yuanzheng625/ImprovedDeepVoxel