CSCE 5563: Intro to Deep Learning Final Project

Project #13

NeuS: Learning Neural Implicit Surfaces by Volume Rendering for Multi-view Reconstruction

Christy Dunlap
University of Arkansas
Department of Mechanical Engineering
cldunlap@uark.edu

Mishek Musa
University of Arkansas
Department of Mechanical Engineering
mjmusa@uark.edu

1. Problem Setup

The authors of this work seek to solve one of the fundamental challenges facing computer vision and graphics, which is the reconstruction of surfaces from multi-view images. The traditional approach to multi-view surface reconstruction typically employs either a point- and surface-based reconstruction method or a volumetric reconstruction method. In the case of point- and surface-based reconstruction local depth maps of each pixel are estimated and then fused into a global dense point cloud. The challenges with these techniques are that the reconstruction quality heavily relies on the quality of correspondence matching and often leads to severe artifacts or missing parts in the reconstruction. For volumetric reconstruction, the occupancy and colors in a voxel grid are estimated and then evaluated for consistency. However, these techniques are also limited in their performance due to limited achievable voxel resolution.

More recently, neural implicit representations have become a promising alternative to the traditional 3D reconstruction methods. Neural implicit representations involve parameterizing a continuous differentiable signal with a neural network since it is continuous and can achieve high spatial representation. In terms of the problem of learning implicit neural representations that encode both geometry and appearance in 3D space from 2D images, two main approaches are used, namely, implicit surface rendering and implicit volume rendering. The state-of-the-art (SOTA) in each approach are Implicit Differentiable Renderer (IDR) [4] and Neural Radiance Fields (NeRF) [2] respectively.

1.1. Related Works

1.1.1 Implicit Differentiable Renderer (IDR)

The IDR approach uses two MLPs to learn a signed distance function for surface representation and surface rendering using 2D images and masks for supervision. A signed distance function is simply a function that represents a shape's surface by a continuous field where the magnitude of a point in the field indicates the distance to the surface boundary and the sign indicates whether the point is inside or outside the shape. This technique has been shown to produce impressive results, however, it fails to reconstruct objects that have complex structures with abrupt depth changes or severe self-occlusions. Additionally, object masks are required for supervision in order for the network to converge to a valid surface. It can be seen in figure 1 that IDR reconstructs the surface with high quality, but due to the surface rendering method it is not able to handle the sudden change in depth. This is because the surface rendering method only samples one point along a given ray at the first intersection of the ray and the surface and so surfaces that are further back are predicted to be a lot closer than they actually are.

1.1.2 Neural Radiance Fields (NeRF)

The NeRF approach uses a volume rendering method to learn a volumetric radiance filed for novel view synthesis. Similar to the previous method, 2D images are used to train the model however rather than learning a signed distance function, the volume density and color or points along rays are learned by an MLP and then a volume rendering approach is used to reconstruct the scene. This technique, unlike IDF, is able to handle abrupt depth changes since rather than sampling just one point along a ray, multiple points along each ray are sampled and $\alpha-$compositing of colors is used to synthesize high-quality images. NeRF, however, is intended for novel view synthesis, not surface reconstruction, and so it is difficult to extract high-quality surfaces from the learned volume density field. This can be seen in figure 1 where NeRF is able to handle the abrupt depth change due to multiple points being sampled along the ray, however, the reconstruction quality is noisy in some of the planar regions since the surface is extracted as a level-set

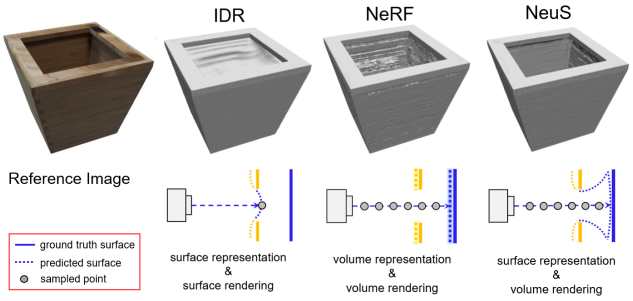


Figure 1. Illustration comparing the IDR, NeRF, and NeuS techniques. An example of a bamboo planter, where there are occlusions at the top of the planter. Compared to the SOTA methods, NeuS is able to achieve better reconstruction quality and handle the occlusions.

surface of a density field.

1.2. Motivation & Proposed Solution

The motivation of this work is to develop an approach that enables accurate 3D implicit surface reconstruction without mask supervision, which can handle occlusions and abrupt depth changes. The authors attempt to achieve this using the signed distance function for surface representation and a novel volume rendering scheme to ensure unbiased surface reconstruction in the first-order approximation of SDF. This method is the "best of both worlds" in terms of the two SOTA approaches as like IDR they use a surface representation to allow for high quality surface extraction but like NeRF they use a volume rendering approach to account for abrupt changes in depth. An illustration of this can be seen in figure 1.

2. Data

In this work, three datasets are used for the evaluation of the model. The first dataset, the DTU (Technical University of Denmark) dataset [1], was used to evaluate the approach and other baseline methods. The second dataset, the BlendedMVS (Multi-View Stereo) dataset [3], was used to further evaluate the model on more challenging, low-resolution images. More details on these datasets can be found in the subsequent sections. The last dataset consisted of images taken by the authors of two thin structured objects (32 images each) to test the approach on thin structure reconstruction.

2.1. DTU Dataset

The DTU Dataset consists of high-resolution scenes scanned using a custom robotic setup. The setup consists of an industrial robot with a structured light scanner/camera mounted to the end effector. This enables repeatable control of the camera position so that large quantities of high-quality data can be produced. However, the kinematics of the robot arm is not directly used to acquire the pose of the camera due to difficulty in accurately controlling the position of the robot. Rather, a calibration procedure is used to record the relative positions of the camera using the camera calibration toolbox for MATLAB and determines the internal and external camera parameters. In this work, 15 scenes/objects from the DTU Dataset are used. The scenes are chosen to represent a wide variety of materials and geometry. Each scene ranges from 49-64 high-resolution (1600×1200 pixels) images and their subsequent camera poses. The scenes were tested with and without foreground masking.

2.2. BlendedMVS Dataset

The BlendedMVS Dataset is a large-scale synthetic dataset created for multi-view stereo training. The dataset utilizes rendered, textured 3D models of architectures, street-views, sculptures, and small objects at different viewpoints to generate training images and depth maps, rather than expensive scanner setups. Additionally, since the camera poses are not generated by a repeatable robot arm, the unstructured camera trajectories allows networks to be more generalizable to real-world reconstructions. In this work, 7 scenes from the BlendedMVS Dataset are used. Each of the scenes consists of 31-143 images with low-resolution (768×576 pixels).

3. Network Design

3.1. Model Design

To reconstruct the 3D scene, two main functions are solved for; signed distance function (SDF) ($\mathbb{R}^3 \rightarrow \mathbb{R}$) and a color function ($\mathbb{R}^3 \times \mathbb{S}^2 \rightarrow \mathbb{R}^3$). The signed distance function is a function that takes in spatial locations as inputs and returns a single value that describes the points location in relation to the surface. A positive number implies it is outside the surface, negative implies inside and 0 means that point is located on the surface. So the surface of the object can be extracted as

$$S = \{x \in \mathbb{R}^3 | f(x) = 0\} \tag{1}$$

The color function takes in inputs of spatial location and a viewing direction and out puts a color for that point. Both of these functions are encoded as MLPs.

The inputs to the model are points or spatial locations and the outputs are the SDF value and the color approximation at that point. More specifically, the spatial location is fed into the SDF MLP which is composed of 8 layers with 256 neurons each with a single skip connection on the fourth layer. The output of this MLP is the SDF value and a 256 dimensional feature vector. Next, this feature vector, the viewing direction and the normal vector of the SDF, and

original spatial location is fed into the color MLP which is composed of 4 layers with 256 neurons. The output of this MLP is the predicted color in the input viewing direction.

To train this model, images taken from known camera locations are used as targets during training (these camera positions can be approximated with their preprocessing code). The colors at each pixel are what is used for training. The color at a single pixel is defined as

$$C(o,v) = \int_0^{+\infty} w(t)c(p(t),v)dt \tag{2}$$

where o is the center of the camera, v is the viewing direction, $c(p(t),v)$ is the color at point $p(t)$ in viewing direction v, $p(t)$ is the ray from the pixel defined as $\{p(t) = o + tv | t \geq 0\}$, and $w(t)$ is a weight function. The weight function meets two criterion; unbiased and occlusion-aware. Unbiased meaning that the local max of the weight function is at the surface intersection and occlusion-aware meaning that if two points along a ray have the same SDF value, the point nearest to the view point will contribute more to the final weight. Using such a weight function and converting the color at a pixel function to an approximate descritized form, the color at a pixel is approximated by the equation

$$\hat{C} = \sum_{i=1}^{n} T_i \alpha_i c_i \tag{3}$$

where T_i is the accumulated transmittance and is defined as

$$T_i = \prod_{j=1}^{i-1}(1-\alpha_j) \tag{4}$$

α_i is the discrete opacity values defined as

$$\alpha_i = max\left(\frac{\Phi_s(f(p(t_i))) - \Phi_s(f(p(t_{i+1})))}{\Phi_s(f(p(t_i)))}, 0\right) \tag{5}$$

Φ_s is the Sigmoid function $\Phi_s = (1 - e^{-sx})^{-1}$, and c_i is the color approximation at the point.

In order to train the model, batches of multiple points along rays are fed into the model with input size (batchSize * Nsamples, 3). The output is then the SDF values with shape (batchSize*Nsamples,1) and color approximations (batchSize*Nsamples, 3). Next, the approximent color at the pixels is calculated using equation 3. Where n is equal to Nsamples and c_i is the color output from the model.

3.2. Loss

The loss is defined as a summation of different loss terms

$$L = L_{color} + \lambda L_{reg} + \beta L_{mask} \tag{6}$$

- $L_{color} = \frac{1}{m}\sum_k L1(\hat{C}_k, C_k)$
- $L_{reg} = \frac{1}{nm}\sum_{k,i}(||\nabla f(\hat{p}_{k,i})||_2 - 1)^2$
- $L_{mask} = BCE(M_l \hat{O}_k)$

The color loss is the main loss term, it compares the approximated color at the pixel to the true color value at the pixels from the images. The L_{reg} is a loss term used for regularization. The L_{mask} is an optional loss term used only when mask images are provided for training.

3.3. Main Contribution of Paper

Reconstructing scenes from images is no new task. The model presented in this paper has novelty due to its use of an unbiased and occlusion-aware weight function. It also uses volume rendering to better estimate depth of objects. This makes it better at accounting for sharp drops in the object surfaces.

3.4. Model Comparison

The NeuS model combines aspects from both NeRF and IDR to yeild better surface and depth rendering. Figure 2 shows the chamfer distances of scene reconstruction from the DTU dataset for the NeRF, IDR, and NeuS model. The chamfer distance is a common metric used to compare point clouds. A smaller value is desired. It is observed that the NeuS model performs the best for the majority of scans especially for the reconstructions with out mask. This is also shown in figures 3 and 4. These figures show reconstructions from all 3 models along with a reference true image. Figure 3 shows the constructions using masks while figure 4 shows them without masks. It can be seen that the NeuS model reconstructions have smooth surfaces while achieving good object depths.

	w/ mask			**w/o** mask			
ScanID	IDR	NeRF	Ours	COLMAP	NeRF	UNISURF	Ours
scan24	1.63	1.83	**0.83**	**0.81**	1.90	1.32	1.00
scan37	1.87	2.39	**0.98**	2.05	1.60	**1.36**	1.37
scan40	0.63	1.79	**0.56**	**0.73**	1.85	1.72	0.93
scan55	0.48	0.66	**0.37**	1.22	0.58	0.44	**0.43**
scan63	**1.04**	1.79	1.13	1.79	2.28	1.35	**1.10**
scan65	0.79	1.44	**0.59**	1.58	1.27	0.79	**0.65**
scan69	0.77	1.50	**0.60**	1.02	1.47	0.80	**0.57**
scan83	1.33	**1.20**	1.45	3.05	1.67	1.49	**1.48**
scan97	1.16	1.96	**0.95**	1.40	2.05	1.37	**1.09**
scan105	**0.76**	1.27	0.78	2.05	1.07	0.89	**0.83**
scan106	0.67	1.44	**0.52**	1.00	0.88	0.59	**0.52**
scan110	**0.90**	2.61	1.43	1.32	2.53	1.47	**1.20**
scan114	0.42	1.04	**0.36**	0.49	1.06	0.46	**0.35**
scan118	0.51	1.13	**0.45**	0.78	1.15	0.59	**0.49**
scan122	0.53	0.99	**0.45**	1.17	0.96	0.62	**0.54**
mean	0.90	1.54	**0.77**	1.36	1.49	1.02	**0.84**

Figure 2. Chamfer distance comparisons between the IDR, NeRF, and NeuS model with and without masks. [3]

4. Demo

The Neus network was used to create a 3D reconstruction of a toy razorback from multiple images taken at dif-

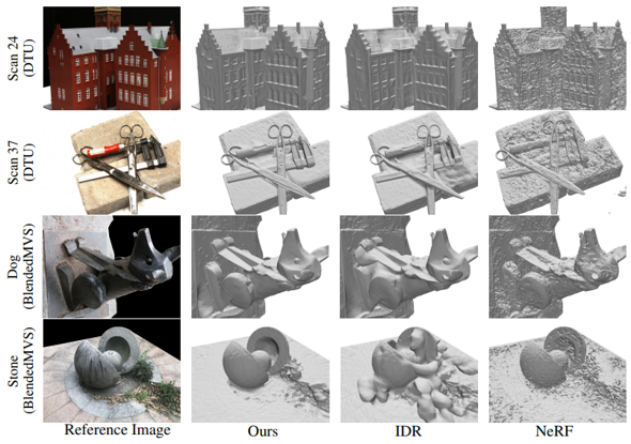


Figure 3. 3D reconstruction from NeRF, IDR, and NeuS model with masks used for training. [3]

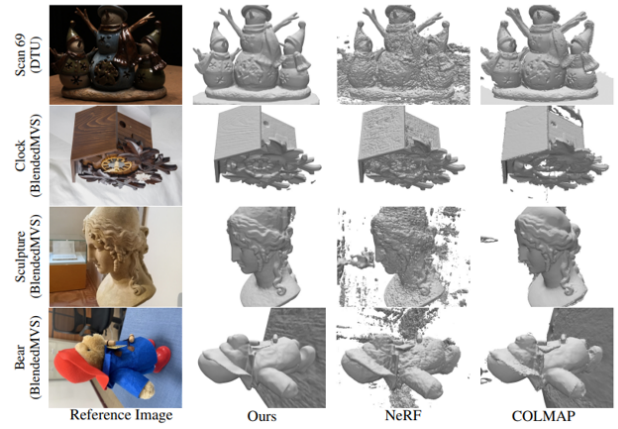


Figure 4. 3D reconstruction from NeRF, IDR, and NeuS model without masks used for training. [3]

ferent angles. The GitHub repository was cloned and an anaconda environment was created with python version 3.7. A computer with an Nvidia RTX 3090 GPU was used for training. Because of incompatibilities with the cuda version, the command "conda install pytorch=1.8 torchvision torchaudio pytorch-cuda=11.7 -c pytorch -c nvidia" was used to install pytorch. The rest of the requirements were installed using pip. A few extra libraries which are not listed in the requirements but are used were also installed such as tensorboard, scikit-image, and imageio.

4.1. Dataset

The dataset consisted of 33 images generated by placing a toy razorback on a table and capturing pictures using a phone camera from different locations. Figure 5 shows a few of the images taken.

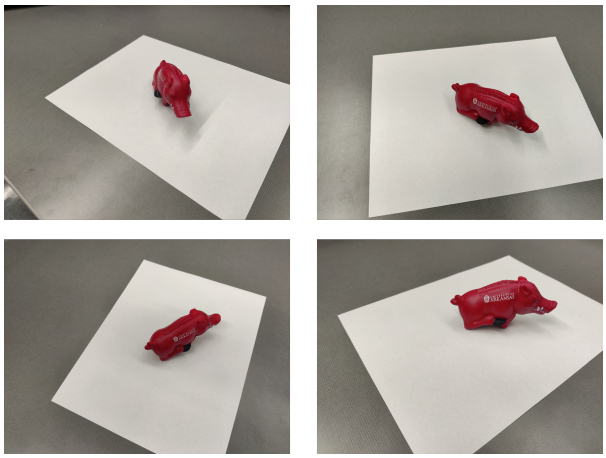


Figure 5. Example images of the toy razorback used in training the model.

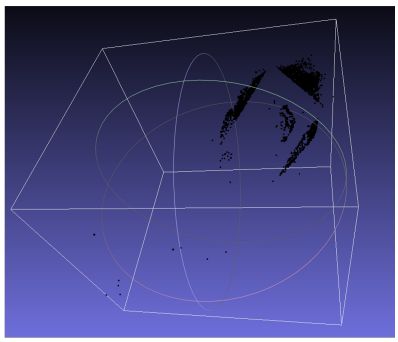
4.2. Camera Calibration

To get the camera positions, the code in the *preprocess_custom_data /colmap_preprocess* was used. Option 2 from the guide found here was followed. The images of the razorback were saved in *case_id /images /*. Then the command *python img2poses.py case_id* was ran. Initially, this code was unsuccessful at generating a sparse point cloud. It only generated the *colmap_output.txt* text file. We went through this file and noted which images were registered. After going through this list, 5 images were not registered so those were taken out of the folder. After this, the *img2poses.py* python file was ran again. Now a *sparse_points.ply* file was created in the *case_id* folder. This file was opened in meshlab and stray points were removed to create a more confined bounding box. This is shown in figure 6. This new version was saved as *sparse_points_interest.ply*. Next the python file *gen_cameras.py* was ran. This generated a new folder in the *case_id/images /* folder called preprocessed. This folder contains a folder of numbered images, a folder of masking images, and a zipped file called *cameras_sphere.npz*. The preprocessed folder and its contents were moved to the cloned git hub folder and saved in the location *Neus / public_data /case_id /*.

4.3. Model Implementation

First the model was trained using our images and generated camera sphere. We chose to not using the masks so the command *python exp_runner.py –mode train –conf . / confs /womask.conf –case case_id* was ran. This initiated the training process. It took around 8 hrs to fully train the model on a computer with an Nvidia RTX 3090 GPU. After training, the 3d model was extracted using the command *python exp_runner.py –mode validate_mesh –conf . /confs*

a.)



b.)

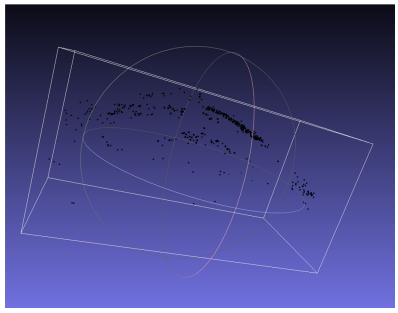


Figure 6. a.) shows the original point cloud generated and b.) shows the new point cloud with more confined bounding box after removing stray points.

/womask.conf –case case_id .

4.4. Results

The 3D reconstruction generated by the Neus model is shown in figure 7.

Figure 7. 3D reconstruction of the razorback generated by the Neus model.

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*, pages 1–16, 2016. 2

[2] 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. *CoRR*, abs/2003.08934, 2020. 1

[3] Yao Yao, Zixin Luo, Shiwei Li, Jingyang Zhang, Yufan Ren, Lei Zhou, Tian Fang, and Long Quan. Blendedmvs: A large-scale dataset for generalized multi-view stereo networks, 2019. 2, 3, 4

[4] Lior Yariv, Matan Atzmon, and Yaron Lipman. Universal differentiable renderer for implicit neural representations. *CoRR*, abs/2003.09852, 2020. 1