Ray Tracing Learned Deformed SDFs
Final Report

Ryan Zesch

December 2021

1 Problem Summary

For my class project, I synthesized recent work on ray tracing learned implicit signed distance functions with my current research on deformable learned representations of objects for collision detection.

Neural networks can represent signed distance functions to arbitrary precision, and have been shown to offer interpolation based on latent codes in networks like DeepSDF [6]. Additionally, work has been done to speed up render times of learned signed distance functions, as in Neural Geometric Level of Detail [10] and DIST[5]. Based on these recent publications, I have trained neural networks to learn deformable signed distance functions based on linear modes of deformation. In addition, I have written a ray tracer that is able to visualize these networks well. This visualization has already become a very useful tool in my research, aiding in tuning network hyperparameters and detecting errors that will affect my collision detection work.

2 Previous Work

Ray tracing is a way of rendering scenes based on the realistic transfer of light as light scatters through an environment. While geometry can be represented in many ways in ray tracing, one well studied format is as a signed distance function (SDF). The most common way to render SDFs, which closely aligns with the ray tracing pipeline, is sphere tracing [3]. In this procedure, a ray r is marched along by taking a step of length $sdf(r(t_i))$ at time t_i. This allows for step sizes to be chosen adaptively until a threshold is reached.

Early representations of signed distance functions often relied on 3D occupancy or distance grid. While this approach works, it is expensive in terms of memory footprint, with space requirements growing cubically. More recently, various attempts have been made to learn a signed distance function in a neural network. Representing signed distance functions to arbitrary precision using neural networks is entirely possible, due to universal approximation properties of neural nets [4].

Recently, the network DeepSDF [6] has shown the ability to learn classes of objects (such as "chairs" or "airplanes") as signed distance functions, using a learned latent space. Following

DeepSDF, many other similar networks have improved upon the network architecture, such as MetaSDF [8] and the SIREN [9] network.

Another recent paper, Neural Geometric Level of Detail (NGLoD) [10], demonstrates that a sparse octree based network can yield great ray tracing speedups over standard MLP networks when learning a static SDF. Similarly, DIST [5] demonstrates various ways one can tweak a ray tracer to speed up ray tracing a neural network, such as parallelization, culling, and aggressive stepping.

In my implementation of a neural network, I used the Pytorch [7] Lightning [2] deep learning library. Additionally, I used SDFGen [1] to generate SDFs which were used in training data generation.

3 Description of Work

In this project, I have successfully developed a ray tracer in python which is able to ray trace deformed SDFs stored in neural nets, including functionality useful to my primary research.

First, I have trained multiple neural nets which have learned signed distance functions of objects under deformations. These neural nets are MLPs with ReLU non-linearities and a tanh final activation function. Training data for these nets was generated by first deforming objects in a FEM simulation and storing linear modes of deformation. For selected deformations, a static SDF was generated using SDFGen [1]. That static SDF was then sampled for training points, with points heavily weighted towards the surface of the mesh. Finally, the Pytorch [7] Lightning [2] model was trained on the linear modes and sampled points. I generated models for a bunny and a dragon, each using 200 deformations with 10,000 points per deformation.

In parallel, I developed a ray tracer in python in order to easily interface with the trained neural nets. As output, ray tracer supports Phong-like lighting with shadows, and simple ray traced lighting with diffuse and emissive materials. In addition, I developed functionality to render normal maps, depth maps, and SDF level sets.

In order to ray trace SDFs, I developed four methods. All methods inherit from a base sphere tracing class. This class took into account some optimizations presented in DIST [5], as previously mentioned. On top of this base class, I first developed a class directly querying the learned MLP SDF at each sphere tracing step.

Next, I developed a dense grid representation which the MLP can be converted to. Grid values were sampled from the original MLP, and distance queries were computed by trilinearly interpolating grid values.

Finally, I developed a sparse octree representation which the MLP can be converted into, based on the work of NGLoD [10]. In this representation, each octree cell is a dense grid which is trilinearly interpolated as before. In this representation, I enabled functionality to render the octree cells themselves, as well a method performing sphere tracing after intersecting a cell. Additionally, I utilized a modified version of their octree ray marching algorithm.

The largest challenge I faced in development was ensuring SDF normal vectors were acceptable. Normal vectors are computed as the gradient of the MLP with respect to the input point. While this works most of the time, occasionally normals return as the zero vector. This is likely due to the ReLU non-linearities in the network. Even replacing ReLU with tanh, these normal issues persisted. I tried multiple approaches to solve this issue, such as approximating normals using finite differencing or using the gradient of the trilinear interpolation where applicable. In the end, normals are generally decent in the interpolation methods, but are still noisy in the exact MLP representation.

4 Results

In order to compare the different SDF representations, I ray traced the same scene of a bunny with all methods. A 256$\times$256 image was generated using 1 ray per pixel in each case. In the octree based methods, a maximum depth of 5 was used for octree cells, with each cell using simple trilinear interpolation. The dense grid method used 72^3 sample points. The following is the resulting timing data.

Method	Direct	Octree (Trilinear)	Octree (Cube)	Dense Grid
Load Time	0.01s	34s	3s	30s
Render Time	22m43s	2m00s	48s	33s

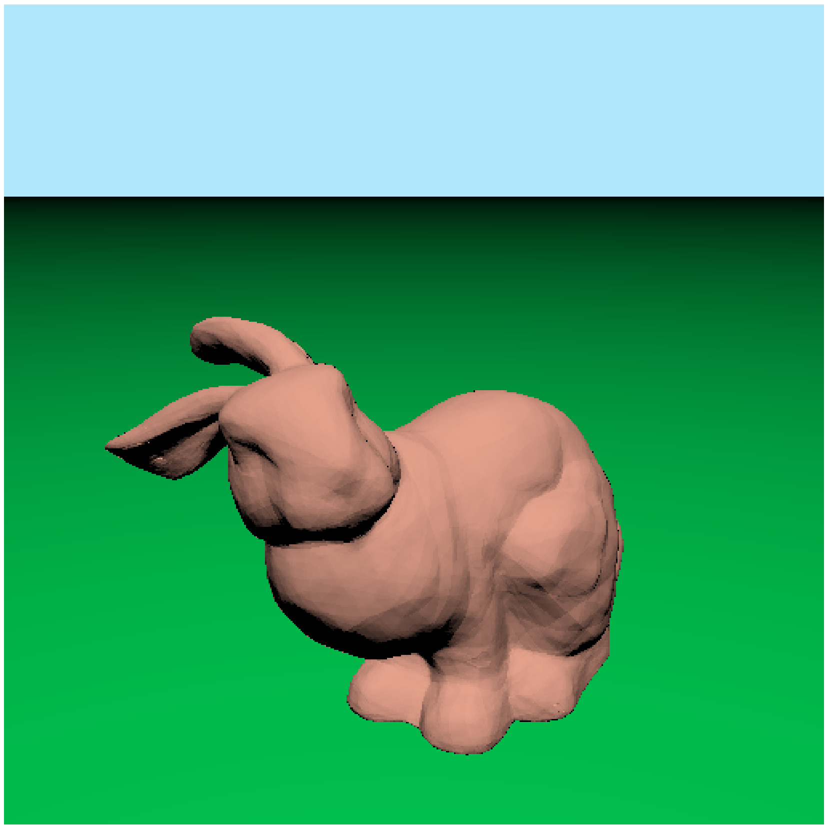

Figure 1: The test deformed bunny scene, as rendered by the direct MLP method.

Overall, the fastest SDF representation for ray tracing is the dense grid in my current implementation. Upon profiling the ray tracer, it is clear that this is due to querying the exact SDF being a very expensive operation, and due to the overhead of octree traversal not being faster than not having an octree in the first place.

Images and videos of the final results are available on the main webpage. In particular, there are videos demonstrating how the learned SDFs are able to deform by changing the input linear deformation code. The learned SDFs match training poses very well, although some finer details are lost in the network itself. Additionally, it is worth noting that the grid and octree based methods are only approximations of the actual SDF, and are prone to losing fine details based on the chosen grid size as all voxel based methods are. Note that some renders and videos include bad normal vectors as mentioned previously, indicated by an erroneous pure black pixel.

5 Analysis of Work

5.1 Novel Results

While I believe my work on learning deformable SDFs using linear modes is novel, I would attribute this idea to my related research rather than this project in particular.

5.2 Meeting Goals

On the whole, I feel as though I met most of the goals I set for myself. In particular, I have developed a pipeline to visualize learned deformable SDFs with enough detail and functionality to be very useful in my related work.

Some of my original goals were modified or dropped. In particular, I did not develop multiple different neural network architectures specifically for ray tracing. Rather, I decided to focus on the ray tracing steps and casting a learned neural net to other representations in order to make my interface work well with many architectures. This will be useful to me going forward as I experiment with different neural architectures for my work on collision detection.

5.3 Future Work

I will likely continue working on this project some, as it pertains to my research. One improvement I would like to make is to replace per pixel parallelization with larger batches, allowing for depth based batch querying of an actual neural SDF. This would likely make rendering the SDF itself more feasible. Additionally, I would like to continue experimenting with how to handle bad normal vectors.

On the neural network side, deformable SDFs for collision detection is my current research topic, so I continue working on different models.

References

[1] Christopher Batty. Sdfgen. https://github.com/christopherbatty/SDFGen, 2015.

[2] William Falcon and The PyTorch Lightning team. PyTorch Lightning, 3 2019.

[3] John C Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545, 1996.

[4] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[5] Shaohui Liu, Yinda Zhang, Songyou Peng, Boxin Shi, Marc Pollefeys, and Zhaopeng Cui. Dist: Rendering deep implicit signed distance function with differentiable sphere tracing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[6] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019.

[7] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[8] Vincent Sitzmann, Eric R Chan, Richard Tucker, Noah Snavely, and Gordon Wetzstein. Metasdf: Meta-learning signed distance functions. *arXiv preprint arXiv:2006.09662*, 2020.

[9] Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020.

[10] Towaki Takikawa, Joey Litalien, Kangxue Yin, Karsten Kreis, Charles Loop, Derek Nowrouzezahrai, Alec Jacobson, Morgan McGuire, and Sanja Fidler. Neural geometric level of detail: Real-time rendering with implicit 3D shapes. 2021.