# Module 3 HW. Implicit 3D Representations

**GitHub**: https://github.com/denysgerasymuk799/UCU_CV_Implicit_3D_Representation

**Notebook1** (my NYU account):
https://colab.research.google.com/drive/1gu3b_pIMRQLoeLhrK-zGlzIp3i2YFCca?usp=sharing

**Notebook2**:
https://colab.research.google.com/drive/1Q4O5YrAWC79PELZhYttKPPPzoONBoCPz?usp=sharing

## Problem definition

In this assignment, we needed to make an implicit 3D representation of object meshes. Nowadays, continuous implicit representation is a common way to represent 3D objects. Generally, an implicit function represents a geometry as a function that operates on a 3D point that satisfies:

1. F(x,y,z) < 0 - interior point
2. F(x,y,z) > 0 - exterior point
3. F(x,y,z) = 0 - surface point

Specifically, the Signed Distance Function (SDF) satisfies those properties. The SDF is simply the distance of a given point to the object boundary and sets the sign of the distance according to the rules above. Keep in mind that the SDF has many nice properties (can be used to calculate intersections easily and make ray-casting fast). [paragraph source]

To obtain SDF from some shape, I mainly have used neural networks. Therefore, the pipeline of solution has the following steps:

1) Create a loss function for the task.
2) Develop a way to get ground truth SDF, random points in the object bounding volume and random points near the surface (with noise std == 1e-2).
3) Find or implement existing NN architectures.
4) Create a function to train a model.

5) Develop a function to test fitted models to get:
    a) Occupancy F1 for points near the surface (with noise std == 1e-2) > 0.9 on average
    b) Occupancy F1 for points in bounding volume of the object > 0.95 on average.
6) Find an average of memory usage, execution time, and two above occupancy F1 metrics.

From the above list, we can notice that steps 2 and 3 require third party libraries/code. I will describe them in the next sections.

## Finding ground truth SDFs and points generation

To complete step 2, I needed to find some third-party libraries that could compute ground truth SDFs and generate points from meshes. During this work, I tested **mesh_to_sdf, trimesh.proximity.signed_distance, and pysdf**.

At the beginning, I found the **mesh_to_sdf** library and used it for SIREN-like NN on Tensorflow (notebook1). I could generate 50_000 test points locally using it and tested my model in Google Colab, but it was not so fast (code for test points generation). Hence, I found faster alternatives **trimesh.proximity.signed_distance** and **pysdf** that I used for an original version of SIREN on PyTorch (notebook2). However, they require a lot of memory for test points. My loss function in notebook2 needed fast execution, so that drawback was allowable.

## Experiments

In general, the problem is quite complex. Good occupancy F1 scores require diligent fine-tuning, and even more, model fine-tuning for each object (NOT general parameters for all objects in a dataset). I tried to run a LOT of projects (>20) on GitHub, but they had complicated pipelines, which needed a lot of adaptations for our task. But I still could succeed with some of them. Also, I used some existing pytorch models and created a pipeline from scratch. In this section, I will describe them.

## DeepSDF

### Why have I chosen this model?

DeepSDF was one of the first models that started to use neural networks for implicit 3D representations. In the past, classical approaches were used to approximate an SDF by discretizing space by a 3D grid, where each voxel represents a distance. These voxels are expensive in memory, especially when a fine-grained approximation is needed.

DeepSDF uses much less memory and makes approximations using a neural network that simply learns to predict for a given point its SDF value. To train the network, they sampled a large number of points with their corresponding SDF. They sampled more points around the object's surface to capture a detailed representation (source).

### Discussion

I tried to reuse code from this repo, but it required quite a lot of time to set up it, and moreover showed not good results. So I decided to leave it for future work. But it was useful for understanding how to test and sample SDF points non-uniformly near the surface. I used that functionality from this mesh_to_sdf lib for testing, which is based on the DeepSDF paper.

## SIREN-like NN on Tensorflow

### Why have I chosen this model?

In general, SIRENs have good performance, capable of detailed 3D representation, and keep these properties for the derivatives of a target signal.

Stanford researchers demonstrated in the paper that SIREN is not only capable of representing details in the signals better than ReLU MLPs or positional encoding strategies proposed in concurrent work, but that these properties also uniquely apply to the derivatives — which is critical for many applications.

SIREN is a simple neural network architecture for implicit neural representations that uses the sine as a periodic activation function. The researchers found that any derivative of a

SIREN is itself a SIREN, as the derivative of the sine is a cosine. Therefore, the derivatives of a SIREN inherit the properties of SIRENs, which enables the researchers to supervise any derivative of SIREN with complicated signals.

Additionally, the researchers demonstrate SIRENs performance across a range of applications that include representing natural signals by fitting their values directly for images, audio, or videos; and solving problems rooted in physics that impose constraints on first order derivatives or second-order derivatives (source).

## Results

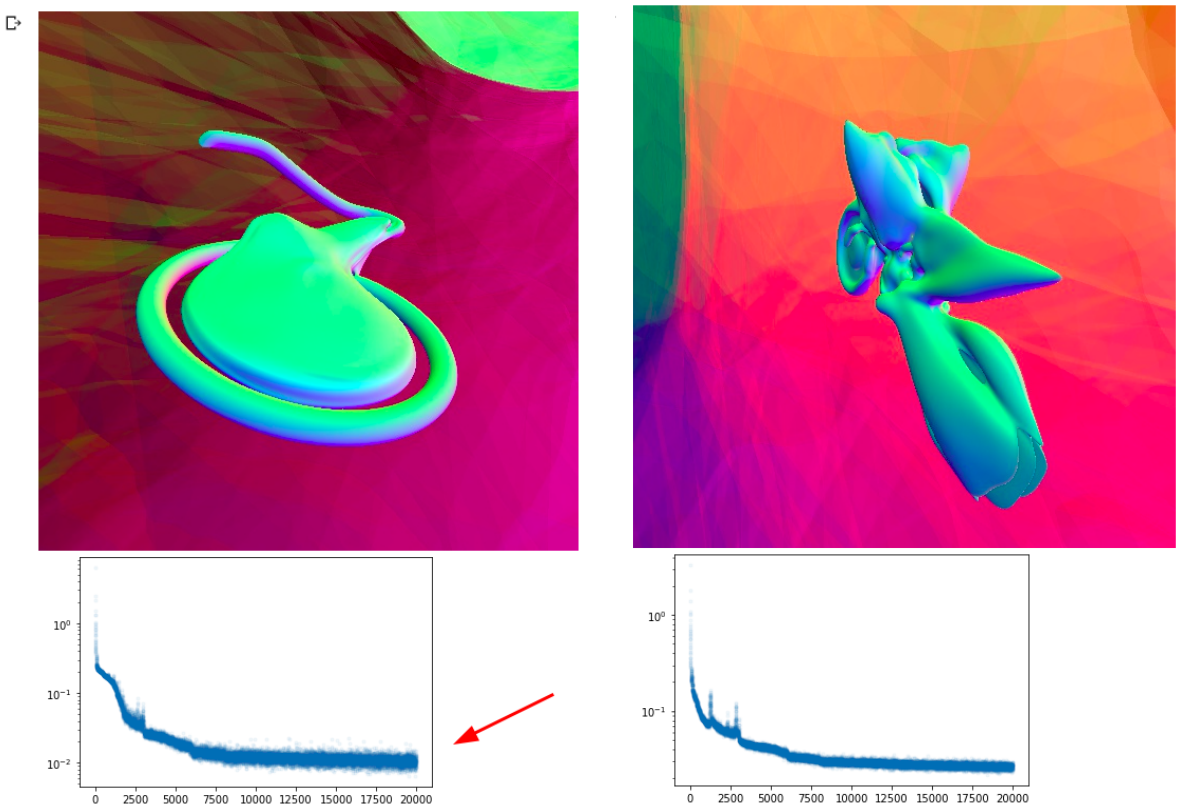
```
def print_lst_avg(title, lst):
  print(f'Average {title} based on {len(lst)} objects: ', round(sum(lst) / len(lst), 4))

print_lst_avg('iteration time (in seconds)', iter_times)
print_lst_avg('memory usage (in Kb)', model_memory_sizes)
print_lst_avg('occupancy F1 random', f1_random_vals)
print_lst_avg('occupancy F1 near surface', f1_near_surface_vals)
```

```
Average iteration time (in seconds) based 5 objects:  0.0043
Average memory usage (in Kb) based 5 objects:  202.5611
Average occupancy F1 random based 5 objects:  0.87
Average occupancy F1 near surface based 5 objects:  0.5218
```

In notebook1, you can find the final results based on 5 objects (you can check comments in the notebook). I reused some code from this Colab notebook. I did not have enough time and resources on Google Colab to test the model with all 50 objects from the dataset. However, the pipeline is elastic to any number of objects. Also, I wanted to play more with other approaches.

## Discussion

To sum up, all requirements are met except occupancy F1 near the surface. To achieve that, I need to add more epochs in the model training. However, we can still notice that more iterations produce more detailed results in this case. Also, it is a good idea to fine-tune the model for each object. In general, this approach shows the best results among others that I tested.




## Original version of SIREN on PyTorch

### Why have I chosen this model?

Reasons are pretty the same as in the case of SIREN-like NN on Tensorflow. But here I wanted to play with an original version of SIREN using [siren-pytorch](siren-pytorch).

### Results

```
def print_lst_avg(title, lst):
  print(f'Average {title} based {len(lst)} objects: ', round(sum(lst) / len(lst), 4))

print_lst_avg('iteration time (in seconds)', iter_times)
print_lst_avg('memory usage (in Kb)', model_memory_sizes)
print_lst_avg('occupancy F1 random', f1_random_vals)
print_lst_avg('occupancy F1 near surface', f1_near_surface_vals)
```

```
Average iteration time (in seconds) based 5 objects:  0.1844
Average memory usage (in Kb) based 5 objects:  912.001
Average occupancy F1 random based 5 objects:  0.5894
Average occupancy F1 near surface based 5 objects:  0.5617
```

In notebook2, you can find the final results based on 5 objects (you can check comments in the notebook). I reused an approach from the official SIREN Colab notebook.

## Discussion

To sum up, all requirements are met except occupancy F1 near the surface. Here, the model showed better occupancy F1 near the surface than the model from the notebook1. I just need to add more epochs to the model training to achieve better metrics. However, more crucially, I developed the correct model training stage. It took some time to create a good loss function. But now the loss constantly decreases over time.

```
  2%|          | 48/3000 [00:05<01:44, 28.37it/s]
Step 50, loss_near_surface: 0.21346572041511536, loss_random: 0.9188302159309387

  3%|          | 99/3000 [00:09<01:30, 31.96it/s]
Step 100, loss_near_surface: 0.07352349907159805, loss_random: 0.7744975090026855

  5%|          | 147/3000 [00:12<01:38, 28.86it/s]
Step 150, loss_near_surface: 0.04370163381099701, loss_random: 0.6362028121948242

  7%|          | 198/3000 [00:16<01:32, 30.23it/s]
Step 200, loss_near_surface: 0.030734173953533173, loss_random: 0.5191296935081482

  8%|          | 249/3000 [00:20<01:29, 30.84it/s]
Step 250, loss_near_surface: 0.02162740007042885, loss_random: 0.45983827114105225

 10%|          | 299/3000 [00:24<01:39, 27.16it/s]
Step 300, loss_near_surface: 0.01805521547794342, loss_random: 0.425927996635437

 12%|          | 348/3000 [00:27<01:29, 29.79it/s]
Step 350, loss_near_surface: 0.01458708941936493, loss_random: 0.39427751302719116

 13%|          | 400/3000 [00:31<01:24, 30.77it/s]
Step 400, loss_near_surface: 0.01186657790094614, loss_random: 0.36263883113861084
```