Implicit Representations Report

Severyn Peleshko

June 2023

Ukrainian Catholic University
APPLIED SCIENCES FACULTY

1 Introduction

In this report, I am going to explore an effective method for representing three-dimensional (3D) shapes with the help of a signed distance function (SDF). The representation of 3D objects is a fundamental task for such spheres as computer graphics, and computer vision, so it's very important to find an effective and compact way to represent our shapes.

There are several ways in which we can represent different geometric objects, for example, mesh-based models or voxel grids. However, those representations have certain problems, among which in the first place are memory requirements, discretization artifacts, and others.

The signed distance function approach offers an alternative quite-powerful representation method. It allows us to represent a surface as a function that assigns a signed distance value to each point, indicating the distance from that point to the surface. This representation provides several advantages, including compactness and the ability to handle complex shapes and implicit surfaces.

2 Data exploration

During this research, I worked with a dataset that contained 50 mesh files, that are used to represent separate objects. Each mesh consisted of two main components: vertices and faces, which uniquely define this object in the 3D space.

The mesh's vertices correspond to the separate points, which in my case were the coordinates in three-dimensional space. Altogether they define the shape and structure of the object. By joining the corresponding vertices, we can get the faces of the mesh, which represent the polygons that enclose a specific region of the object. But despite this, each object was additionally provided with the faces list, which can be used during some calculation and visualization tasks.

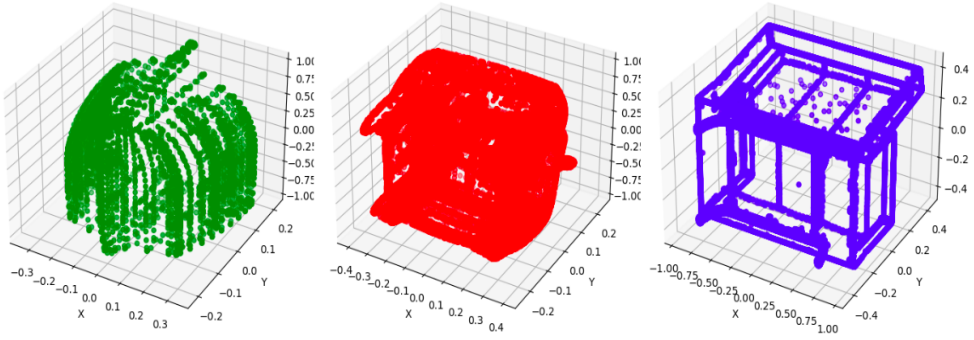

Figure 1: Mesh vertices

In the above figure, we can observe the vertices of the mesh objects in the dataset. While the figure below demonstrates us the corresponding objects' faces and meshes. Exploring the vertices can give us insights into the specific points that define the 3D mesh shape. Conversely, analyzing the faces allows us to comprehend the connectivity and structure of the objects.

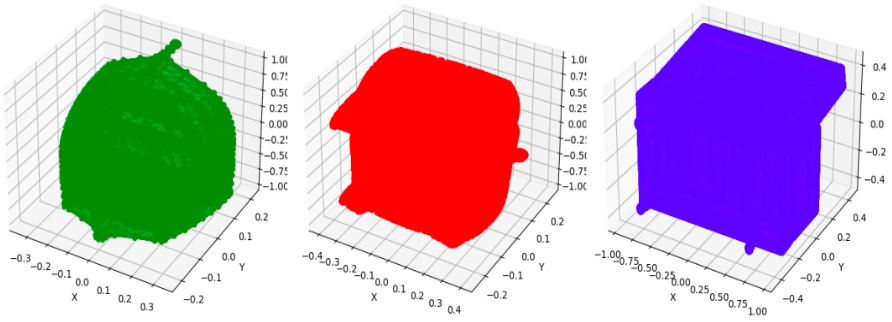

Figure 2: Mesh faces

3 Approach

In order to accomplish this task, I conducted some research, on the ways how to calculate the SDF values efficiently and the promising ones were to try to use a neural network model, which will be fine-tuned or trained from the start in order to receive the bunch of 3D points and automatically produce the corresponding SDF values for them. For example, if the point lies on the object's contour the SDF should be equal to zero, while at the same time, the presence of a point inside the object leads to a positive SDF value and, accordingly, outside will result in a negative one.

The possible options were some multi-layer perceptrons and SIREN(Sinusoidal Representation Networks) models. Actually, I stopped on the last one(SIREN model), which accordingly to its article, shows quite promising results in the task of representing complex 3D scenes, shapes, and objects, so can be efficiently applied to my task of estimating the SDF values.

3.1 Data preprocessing

In order to produce some meaningful results and therefore to train my SIREN model, I needed well-prepared data. To accomplish this I generated the dataset, which mainly consisted of two essential parts, in the first place it's the vertices' of the mesh object itself, which actually can be used to represent the shape of this object, so their SDF would be equal to zero, and the second one it's randomly sampled points from the 3D space in which is located our object. In such a way, for each object, I generated some point data, which included contour points, that were initially given, and some arbitrary ones inside or outside the object. The amount of such random sampled points is actually the hyperparameter and can be tunned through some experiments, in my case after trying some values, I stopped on the 5000 points.

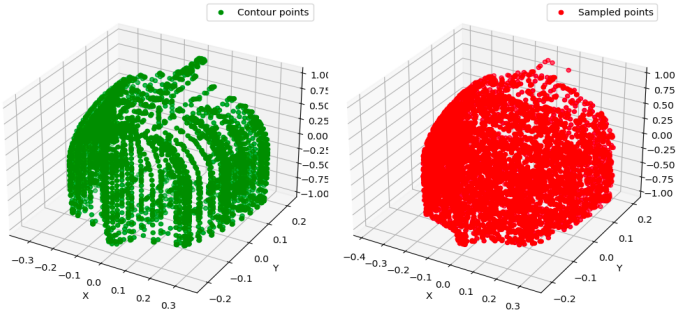

Figure 3: Contour vertices vs sampled ones, mesh object №1

For example, the figure above illustrates us the example of the points set received after the data preprocessing stage. Here on the right, we can observe the sampled points, while on the left are contour ones.

After that, I generated the corresponding SDF values for my points set, because to train the SIREN model I will use a supervised approach. Speaking more precise about the process of SDF calculation, it's important to mention that for those vertices on the contour I assigned SDF equal to zero, because they actually define the object's shape, while for sampled points I used an already implemented function from the python-package pysdf, that calculates SDF automatically for the bunch of points.

3.2 Model architecture

So, as was mentioned earlier as a model architecture I have chosen the SIREN model, which is a quite-powerful tool for a large variety of tasks, especially for those one related to implicit representations, where the network, for example, learns to map points in a 3D space to a corresponding value, such as a signed distance function (SDF) or color intensity.

The model itself was introduced in a research paper named "Implicit Neural Representations with Periodic Activation Functions". And its key feature is the usage of sinusoidal activation functions instead of commonly used activation functions like ReLU or sigmoid. The sinusoidal activations allow the model to produce smooth and continuous outputs across the entire input space. This property is very beneficial for an accurate representation of 3D geometry, as it allows us to produce good approximations of objects' shapes. Also accordingly to the paper authors are recommending to use the next lost function for SIRENS's SDF approximation task:

$$\mathcal{L}_{\text{sdf}} = \int_{\Omega} \left\| \left|\nabla_{\mathbf{x}} \Phi(\mathbf{x})\right| - 1 \right\| d\mathbf{x} + \int_{\Omega_0} \|\Phi(\mathbf{x})\| + \left(1 - \langle \nabla_{\mathbf{x}} \Phi(\mathbf{x}), \mathbf{n}(\mathbf{x}) \rangle\right) d\mathbf{x} + \int_{\Omega \setminus \Omega_0} \psi\big(\Phi(\mathbf{x})\big) d\mathbf{x},$$

Figure 4: Loss formula

To introduce some small clarifications, I will mention, that by Ω we represent our whole domain of points, and by Ω_0 we denote those ones with the zero SDF value. Also, the last term is used to penalize off-surface points for creating SDF values close to 0 and it stays for the next expression:

$$\psi(x) = \exp(-\alpha \cdot |\Phi(x)|), a \gg 1$$

Among the main SIREN advantages is its ability to handle large-scale scenes and complex objects' geometry. It can efficiently represent intricate shapes and capture even small details. Moreover, SIREN can be easily trained using gradient-based optimization methods, making them a great tool for various computer graphics and computer vision tasks.

4 Evaluation

To evaluate the model's performance for different mesh objects, I initially split my dataset into three parts: training, development(validation), and test sets. On the first two, I actually was training and validating my model architecture, and based on the last one I was evaluating my model's overall performance, by calculating the F1 occupancy scores, RMSE, and MAE metrics. The important remark here would be that those metrics and models themselves were calculated and trained separately for each mesh object and then based on metrics values were calculated some averaged scores and compared.

4.1 Separate mesh object evaluation

In this section would be discussed a comprehensive analysis of the evaluation stage of SIREN's performance on a separate mesh object, specifically focusing on object №1. Bellow you could observe the shapes of the sets, which were obtained after initial data preprocessing and split:

```
{'X_train': (7717, 3),
 'Y_train': (7717, 1),
 'X_val': (858, 3),
 'Y_val': (858, 1),
 'X_test': (953, 3),
 'Y_test': (953, 1}
```

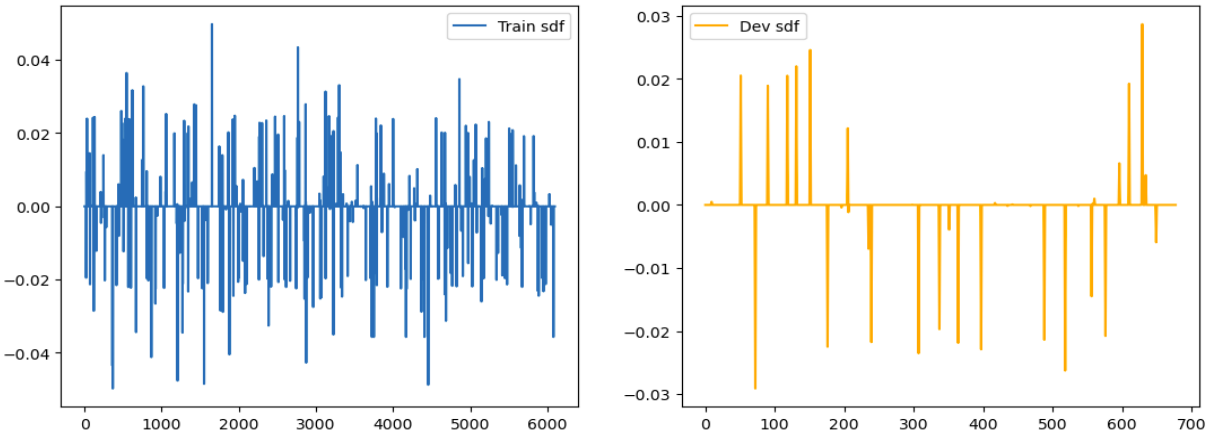

Figure 5: SDF split

The above plots illustrate us the distributions of ground-truth signed distance function (SDF) values for the development and training sets respectively. In the figure below you could observe the model's overall performance, being more precise we can see how changed loss, RMSE, and MAE metrics during the training stage for both validation and train sets.

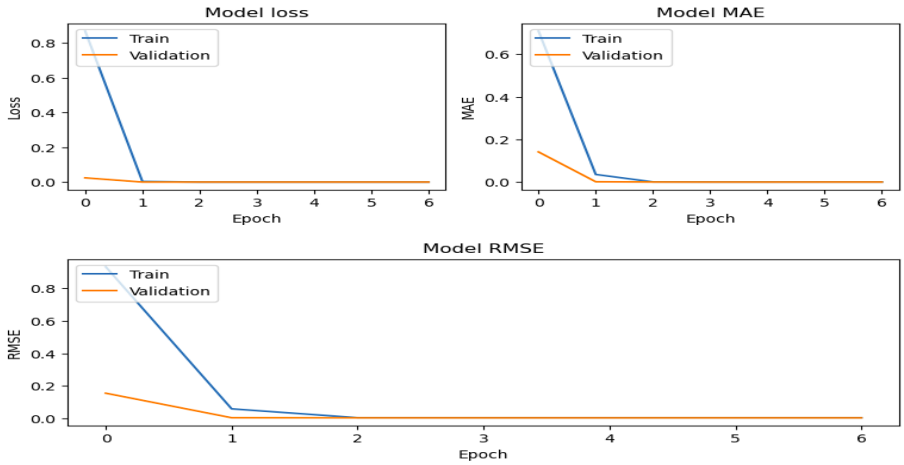

Figure 6: Training stage

At first glance, we can say that the results are quite good and the model shows excellent performance, but actually, this is not the case, in reality, performance is worse and therefore those metrics are not very representative. The possible reason is that the range of our SDFs is not very wide and variable, so as a result, many SDF values differ from their actual values on some small amount which leads to very small losses, but at the same time, it's very important for us to produce precise results, because even slight deviations from the ground-truth SDF values can significantly impact the resulting shape.

To obtain a more accurate evaluation of the model's performance, I also computed two occupancy F-scores. The first one stays for points near the surface (with a noise standard deviation of 1e-2), and measures the model's ability to accurately represent the object's boundary. The second F-score is for points inside the bounding volume and assesses the model's performance in capturing the interior regions of the object. Their values could be observed below:

```
{'Occupancy F1 for points near the surface': 0.64,
 'Occupancy F1 for points in the bounding volume': 0.26}
```

So now we see that model faces some troubles while approximating SDF values for the object's points, especially for those inside the bounding volume, which actually is bad because it won't allow us to accurately reproduce the object's shape without having its explicit representation. Towards the end of this section, I will mention, that during the modeling stage, I tried different model architectures, by combining SIREN layers with sequential dense or dropout ones, as well as working with the SIREN model itself. Also, I experimented with various hyperparameters, for example, with the optimizers, the number of hidden layers, neurons, and others.

4.2 Multiple objects' evaluation

So in order to evaluate the model's overall performance on different objects I trained my SIREN architecture for each mesh object separately and then measured its performance, by calculating occupancy F-score metrics. Above you could observe the averaged metrics' values:

```
{'Occupancy F1 for points near the surface': 0.51,
 'Occupancy F1 for points in the bounding volume': 0.25}
```

So as in the previous section, the model faces serious troubles while approximating SDF values for different points. So we can't state that the model is accurate and in the future, it won't allow us to precisely reproduce objects' shapes.

5 Conclusion

So summing up, it's important to mention, that as a result of this assignment was produced a SIREN model, that was trained on objects' meshes and is used to approximate the SDF values for a point in the 3D space. The received model satisfies the memory and time constraints, but unfortunately, it has some troubles with the performance quality. Actually, this is the problem because due to this limitation, we won't be able to accurately reproduce an object's shape in 3D space through the SDF implicit representation. The possible solution to this problem would be to somehow enhance the model's architecture and tune hyperparameter values, for example, by increasing the number of hidden layers, changing the neuron amount, or totally changing the architecture from SIREN to another one. The other promising approach would be to change the loss function to those one specified in the paper, which I also mentioned in Figure number 4, because the MSE or MAE loss may not be well-suit for this task, for the reasons I described earlier.

6 References

1. Implicit Neural Representations with Periodic Activation Functions
2. GitHub with the code