HW SDF

Introduction

The implicit neural SDFs (signed distance functions) are used to parameterize 3D meshes. Their key benefits include *fixed size* regardless of resolution. They are used for simulation, path planning, 3D modeling and videogames. We implement the methods from two prominent papers:

- SIRENs (Sitzmann et al.)
- NG-LOD (Takikawa et al.)
  SIRENs were revolutionary at their time because they improved fitting of images and SDFs by utilizing specific architecture and initialization scheme.
  NG-LOD improved the inference speed of SDFs by utilizing specific architecture (based on sparse octree of feature vectors) enabling SDF inference at interactive rates.

Related works

The common MLPs with ReLU activations proved to be hard to optimize for fitting high frequency functions with low dimensional domains (in particular, SDFs). Tancik et al. showed that utilizing positional encoding before feeding the coordinates to MLP improved the performance. However, the method required the parameter sweep to adjust for frequency of the signal. Later on, Herz et al. mitigated the issue by proposing spatially-adaptive progressive encoding.
Sitzmann et al. replaced ReLUs with the element-wise application of sine and proposed a well-suited initialization scheme that enabled building large networks. The gradient with respect to input is also a SIREN. They showed that SIRENs preserve gradients of the signal and can be trained on gradients for indirectly fitting the signal.
Takikawa et al. worked on SDFs and developed NGLOD - an architecture based on sparse voxel octree that was fast in terms of inference, enabled real-time inference for rendering. Additionally, they enabled various levels of details for rendering. Moreover, their model was small enough due to sparsity of the voxel octree.
Muller et al. introduced hashing to improve training and inference speed and also used a slightly different implementation of NGLOD: the one that uses concatenation instead of linear interpolation.

Method

We implement the commonly used approach. It is to minimize some loss L penalizing the discrepancy between the predicted signed distance Φ_θ (neural network) and true signed distance. The loss can include additional regularization terms or terms imposing structure inherent to real signed distance function.

SIRENs

We utilize the architecture and initialization scheme proposed by Sitzmann et al.

Objective

We minimize the loss defined as:

$$L = \int_\Omega \||\nabla_{\mathbf{x}} \Phi(\mathbf{x})| - 1\| d\mathbf{x} + \int_{\Omega_0} \|\Phi(\mathbf{x})\| + (1 - \langle \nabla_{\mathbf{x}} \Phi(\mathbf{x}, \mathbf{n}(\mathbf{x})), \rangle) d\mathbf{x} + \int_{\Omega \setminus \Omega_0} \psi(\Phi(\mathbf{x})) d\mathbf{x}$$

where Ω is a set of all points, Ω_0 is a set of on-surface points, $\psi(\mathbf{x}) = \exp(-\alpha|\Phi(\mathbf{x})|), \alpha \gg 1$ penalizes off-surface points for creating SDF values close to 0. $\mathbf{n}(\mathbf{x}) = \nabla f(\mathbf{x})$ is the gradient of ground truth SDF f at point $\mathbf{x}$.

Architecture

We optimize the loss with respect to the parameters θ of $\Phi(\mathbf{x}) = \Phi_\theta(\mathbf{x})$.

$$\Phi(\mathbf{x}) = \mathbf{W}_n(\phi_{n-1} \circ \phi_{n-2} \circ \cdots \circ \phi_0)(\mathbf{x}) + \mathbf{b}_n$$

$$\phi_i(\mathbf{x}_i) = \omega_0 \cdot \sin(\mathbf{W}_i \mathbf{x}_i + \mathbf{b}_i)$$

Initialization

Elements of $\mathbf{W}_0$ are initialized from uniform distribution $U(-\frac{1}{n_0}, \frac{1}{n_0})$, where n_0 is the number of input features to the network (3 in case of points in 3D).
Elements of $\mathbf{W}_i, i \in 1 \dots n$ are initialized from uniform distribution $U(-\sqrt{\frac{6}{n_i \omega_0^2}}, \sqrt{\frac{6}{n_i \omega_0^2}})$, where n_0 is the number of input features to $(i+1)$th layer.

Dataset

Dataset is approximately 10^5 points sampled from the surface of a mesh using Poison disk sampling. Points are centered and normalized to range $[-1, 1]$ preserving the aspect ratio.

Training details

We minimize the objective using mini-batch gradient descent. We optimize with batches consisting of 5000 points sampled from surface and 5000 points sampled randomly from a bounding box. On each iteration, we minimize the aforementioned loss but with terms scaled by

respective factors 3000, 100, 100 and 5. We use Adam optimizer with initial learning rate of 10^{-4} for calculating gradient statistics. We divide learning rate by 10 when the loss plateaus.

NGLOD

We utilize the architecture proposed by Takikawa et al.

Objective

We minimize the MSE loss defined as:

$$L = \mathbb{E}_{\mathbf{x}}(\|\Phi(\mathbf{x}) - f(\mathbf{x})\|^2)$$

where $\Phi(\mathbf{x}) = \Phi_\theta(\mathbf{x})$ is the signed distance predicted by a neural network and $f(\mathbf{x})$ is the ground truth distance.

Architecture

We architecture is as proposed by Takikawa et al. It is based on an octree. We define K to be the depth of an octree. The output of a neural net is defined as $W_1 \max(W_0 \sum_{i=0}^{K-1} Z_i(\mathbf{x}), 0)$, where $Z_i(\mathbf{x})$ is a trilinearly interpolated feature vector from feature vectors of the 3D grid of level i.

Dataset

We sample $5 \cdot 10^5$ points on every epoch from the bounding volume of a mesh. The sampling algorithm is as proposed by Takikawa et al. In particular, we sample 10^5 points uniformly from the bounding box, $2 \cdot 10^5$ points from the points near surface and $2 \cdot 10^5$ points from the mesh. The points are centered and normalized to range $[-1, 1]$ with preserved aspect ratio before feeding to neural network.

Experiments

We train each of the two models on 50 triangular meshes. For each model and each mesh we calculate the average of two F1 scores: (calculated for points in the bounding box and near surface). The points near surface are obtained by sampling the points on a surface (Poisson-disk sampling) and then perturbing them (adding noise with $\sigma = 0.01$). Additionally, we measure the running time and storage volume of each model.

SIRENs

We build a Siren with four linear layers of sizes 256 and final layer projecting to 1. The last layer is not followed by an application of sine.

F1 scores

Mesh	F1 in bounding box	F1 near surface
0	0.93	0.86
1	1.0	0.99
2	1.0	1.0
3	1.0	0.99
4	0.99	0.98
5	0.99	0.97
6	1.0	0.99
7	0.87	0.84
8	0.99	0.98
9	0.07	0.36
10	0.98	0.97
11	1.0	1.0
12	1.0	0.99
13	0.99	0.96
14	1.0	0.99
15	0.69	0.66
16	0.5	0.7
17	0.83	0.73
18	0.63	0.63
19	0.96	0.95
20	0.81	0.75
21	1.0	0.98
22	0.99	0.96
23	0.99	0.97
24	1.0	0.99
25	1.0	0.99
26	0.87	0.81
27	0.96	0.94
28	0.58	0.55
29	1.0	0.99
30	1.0	0.98

Mesh	F1 in bounding box	F1 near surface
31	0.98	0.96
32	0.61	0.6
33	0.82	0.74
34	0.99	0.96
35	1.0	0.99
36	0.64	0.67
37	1.0	0.99
38	1.0	0.99
39	1.0	0.99
40	1.0	0.99
41	0.99	0.97
42	0.99	0.96
43	0.64	0.58
44	0.71	0.99
45	1.0	0.99
46	0.94	0.94
47	0.99	0.97
48	1.0	0.98
49	0.99	0.98
Average	**0.9**	**0.89**

Memory

The occupied memory of the $.pth$ file with model parameters is **0.77 MB** and does not vary from mesh to mesh. The model has approximately 200 thousand parameters which correspond to **0.8 megabytes** (assuming FP32)

Running time

Single point*	Batch of 3000 points
~89ns	~703us

*Time for single point was measured with batch of 50000 points.
The time was measured on RTX 2080ti.

NGLOD

We build the model with 3 levels of octree and an MLP with single hidden 128-way linear layer. We do not utilize sparsity (i.e. we utilize features from every level during inference).
Note: further works showed that we do not necessarily need the MLP at the end, but we observe a significant decrease in F1-score when removing the network.

F1 scores

mesh	F1 in bounding box	F1 near surface
0	0.97	0.9
1	0.93	0.88
2	1.0	0.98
3	1.0	0.98
4	0.97	0.93
5	0.99	0.93
6	0.95	0.85
7	0.97	0.96
8	0.92	0.76
9	0.9	0.69
10	0.93	0.78
11	1.0	0.97
12	1.0	0.97
13	0.99	0.96
14	1.0	0.95
15	0.97	0.91
16	0.96	0.91
17	0.98	0.95
18	0.99	0.93
19	0.97	0.94
20	0.96	0.91
21	1.0	0.96
22	1.0	0.98
23	1.0	0.98
24	1.0	0.95
25	0.99	0.96

mesh	F1 in bounding box	F1 near surface
26	0.99	0.94
27	0.99	0.95
28	0.98	0.94
29	1.0	0.96
30	1.0	0.98
31	0.98	0.96
32	0.99	0.96
33	0.99	0.93
34	0.99	0.98
35	0.97	0.95
36	0.96	0.89
37	1.0	0.97
38	0.99	0.98
39	0.98	0.94
40	0.99	0.97
41	0.99	0.95
42	1.0	0.97
43	0.97	0.92
44	0.97	0.98
45	0.99	0.97
46	0.98	0.94
47	1.0	0.97
48	0.99	0.94
49	0.98	0.97
Average	**0.98**	**0.94**

Running time

Single point*	Batch of 3000 points
~7ns	~310us

*Time for single point was measured with batch of 50000 points.
The time was measured on RTX 2080ti.

Memory

Occupied space of a $.pth$ file is approximately **0.5 MB** and does not vary from mesh to mesh. The model has approximately 140 thousand parameters with correspond to **0.56 MB** of memory (assuming FP32).

Possible improvements

Most probably we can improve the results making the octree sparse (do not allocate a voxel when it is outside the mesh). Maybe the improvement in memory and inference speed can be achieved by using sparse tensors.

References

Fourier features (Tancik et al.): https://arxiv.org/pdf/2006.10739.pdf
SIRENs (Sitzman et al.): https://arxiv.org/pdf/2006.09661.pdf
Spline encoding: https://arxiv.org/abs/2106.01553
SAPE (Hertz et al.): https://arxiv.org/abs/2104.09125
NG-LOD (Takikawa et al.): https://research.nvidia.com/labs/toronto-ai/nglod/assets/nglod.pdf
INGP (Muller et al.): https://nvlabs.github.io/instant-ngp/