NeurIPS 2023 - 三维重建中的Neural SDF(Neural Implicit Surface)

本文介绍一下我们组在NeurIPS 2023发表的工作: StEik: Stabilizing the Optimization of Neural Signed Distance Functions and Finer Shape Representation。这是一篇关于Neural SDF基础理论的paper，所以这篇文章中我先给大家梳理了一下Neural SDF(Neural Implicit Surface)近几年的发展历程，再介绍我们的贡献。

论文链接：StEik: Stabilizing the Optimization of Neural Signed Distance Functions and Finer Shape Representation

代码链接：https://github.com/sunyx523/StEik

Introduction

近期基于体渲染的三维重建方法中，有项距离场Signed Distance Function(SDF)被广泛的用来表示三维表面，而SDF又被MLP隐式的定义。刚刚接触这个领域可能认为SDF近期才出现并应用到三维重建中，其实SDF的历史已经有几十年了，早在二十多年前SDF就被广泛的应用到图像分割和三维重建算法中（三维重建和图像分割其实可以看作同一个任务，三维重建其实是将空间分割为物体内和物体外两部分）。

这些算法都是基于calculus of variations来求解计算机视觉问题的，这是一种非常传统的算法，在二十年前是当时计算机视觉的主流算法，但国内的老师大多是深度学习火起来之后才开始进入cv领域的，对这种传统方法了解的不多。我老板Anthony Yezzi也算这个领域的小牛了，曾在01年拿过CVPR outstanding student paper，对SDF有很深的理解，毕竟做了几十年了，我也希望能将我所学到的分享给大家。之前SDF是被显式地储存在grid中，随着近几年深度学习的大火，人们开始用MLP隐式的表示SDF。Implicit Representation相比于Explicit Representation的优势很明显：

- MLP是连续的，可以查询任意位置的SDF值
- 可以通过PyTorch求得analytic gradient，避免numercial gradient引起的数值问题

Stereoscopic segmentation[1]

首先简单讲一下我老板很久之前的算法，也是三维重建的经典模型。在该方法中，物体的表面 S也是用一个SDF $\phi:\mathbb{R}^3\to\mathbb{R}$的零水平集来表示，不过此时SDF是显式的储存在一个grid之中。并且定义了foreground radiance function $\mathbf{f}:S\to\mathbb{R}$和background radiance function $\mathbf{g}:B\to\mathbb{R}$， $\mathbf{f}$和$\mathbf{g}$也都是显式的储存在grid之中。之后将这些三维空间中的表示投影到输入的图片视角，通过迭代 $\phi,\mathbf{f},\mathbf{g}$来缩小投影和输入图片的差值。基于表面渲染的三维重建和这个方法非常像，只不过SDF用SDF network来表示，radiance function用radiance network来表示，所以老板看到DVR和IDR后感叹，这不就是我二十年前做的东西么。

Neural Implicit Surface

在DeepSDF[2]中，作者首次尝试用MLP来隐式的表示SDF，通过缩小MLP的输出和ground truth SDF的差来让MLP拟合一个SDF：

$$f_\theta(\boldsymbol{x}) \approx SDF(\boldsymbol{x}), \forall \boldsymbol{x} \in \Omega$$

$$\mathcal{L}\left(f_\theta(\boldsymbol{x}), SDF(\boldsymbol{x})\right) = \left|\text{clamp}\left(f_\theta(\boldsymbol{x}), \delta\right) - \text{clamp}(SDF(\boldsymbol{x}), \delta)\right|$$

对于不在表面上的点，DeepSDF也需要知道它们的符号（物体内部/外部）和距离表面的距离。然而在很多应用中，很难得到一个ground truth SDF。SAL[3]提出了不需要ground truth SDF的训练方法，并且提出了几何初始化，即将MLP的输出初始化为一个球体。这个初始化非常重要，在以后的工作中被广泛用到，之前单独写了一篇文章讲解这种初始化：

sunsun：基于体渲染（NeRF）三维重建中的几何初始化

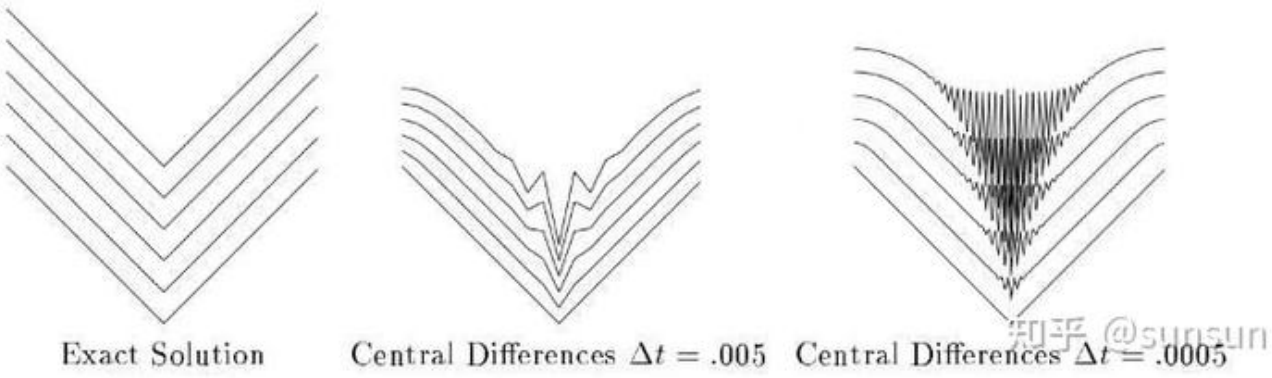
在此基础上SALD[4]在训练过程中引入了点云的normal监督，并大幅提高了重建的质量。

Eikonal Loss

在IGR[5]中，作者引入了基于Eikonal Equation的约束项，可以让学习到的函数接近一个SDF，这项约束可以在有无normal的情况下都可以使用。并且作者证明了仅仅利用这项约束加上约束表面点的SDF值为0，就可以学到理想的SDF，不需要使用SAL[6]中提出的复杂学习方法。这项约束后来成为著名的Eikonal Loss，是三维重建中必不可少的一项约束。

然而Eikonal Loss在基于variational method中并没有被广泛使用，取而代之的是reinitialization，即根据zero level set使用fast marching method重新生成一个SDF。其中一个原因如下图所示，因为SDF是用离散的grid表示的，求梯度时要用到finite difference scheme，在求拐点附近的梯度时，如果用简单的central difference scheme就会造成如下

问题



Fast marching method, Sethian

需要使用复杂的upwind scheme才能避免奇点的不连续性。Eikonal Loss还有另一个问题就是optimization过程不稳定（这个不稳定性我将在后面讲到），之前也有很多文章[7]研究了Eikonal equation的这种不稳定性，这也就是为什么在variational method中人们几十年来都没有使用Eikonal Loss。

然而为什么最近的Neural SDF中又开始使用Eikonal Loss了呢，这是因为MLP相比于voxel grid存在一种implicit reguarization，也即FFN中提到的spectral bias，可以稳定Eikonal Loss的优化过程。然而当使用更强表达能力的结构时，这种implicit reguarization也会减弱，不稳定性又会出现。为了克服spectral bias重建出更多细节，人们开始探索表达能力更强的网络，随之而来的便是Eikonal Loss带来的不稳定性。

DiGS[8]

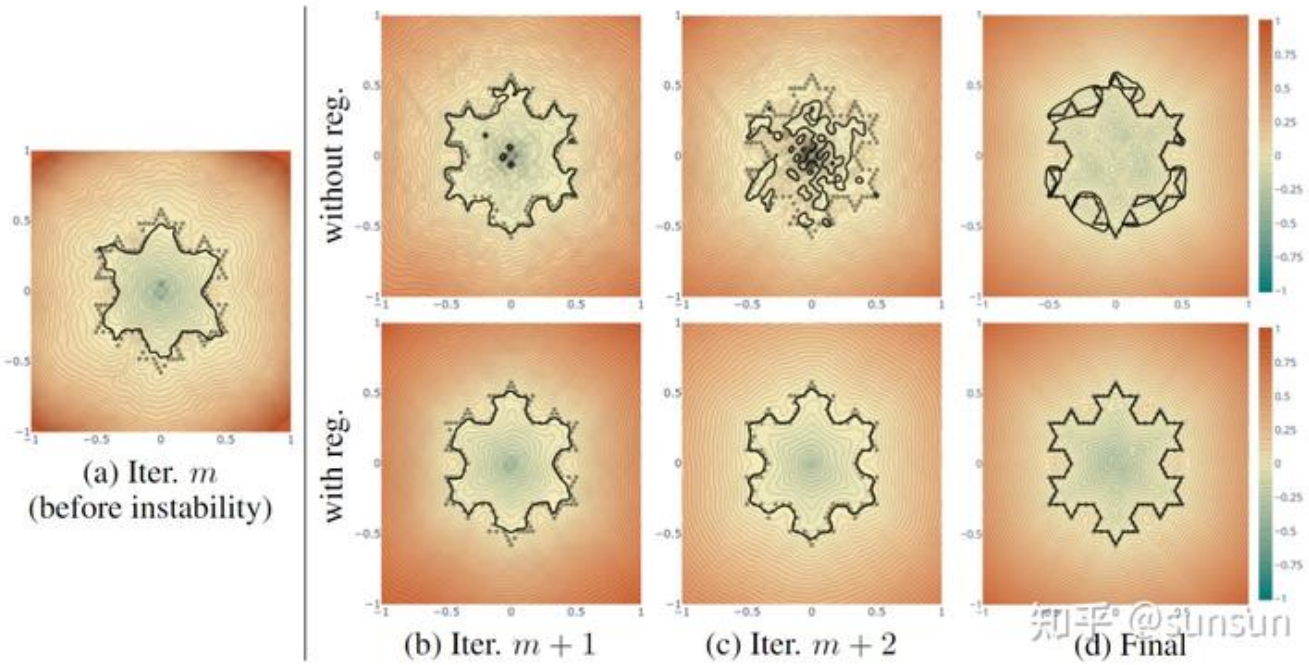
在这篇文章中作者尝试使用SIREN替换ReLU MLP来捕捉更多的几何细节。作者发现当SIREN网络在没有normal constrain（即输入ground truth normal，惩罚其和表面normal的差）的情况下重建效果很差，于是作者引入了二阶项约束，即梯度场的散度divergence，也是SDF的Laplacian：

$$L_{div} = \int_{\Omega \setminus \Omega_0} |\Delta \Phi(x;\theta)| dx = \int_{\Omega \setminus \Omega_0} |\nabla_x \cdot \nabla_x \Phi(x;\theta)| dx$$

作者发现使用二阶约束项后重建效果会变好。作者是根据观察SDF场，发现其二阶导数为零，才根据经验提出的这项约束。然而作者不知道这背后的理论原因，其实是因为normal constrain具有稳定Eikonal Loss的效果，所以当没有normal constrain的时候SIREN重建效果会变差，而二阶约束也具有稳定Eikonal的效果，所以加入二阶项也会让重建效果变好。

Stability analysis of Eikonal Loss

在这篇论文中我们证明了Neural SDF优化过程中存在不稳定性, 使用的是PDE stability analysis，是variational method中常用的一种稳定性分析方法，具体的证明推导过程过于理论，感兴趣的话可以阅读我们论文的原文。并且我们也证明了DiGS中提出的 L_{div}和normal constraint具有稳定Eikonal Loss的作用，这也解释了引入这两项约束为何能提升训练效果，但是同时引入的话效果不会比只引入其中一个高很多，因为这两项起到的作用是相似的。我们用了一个二维的简单例子来展示这种不稳定性：



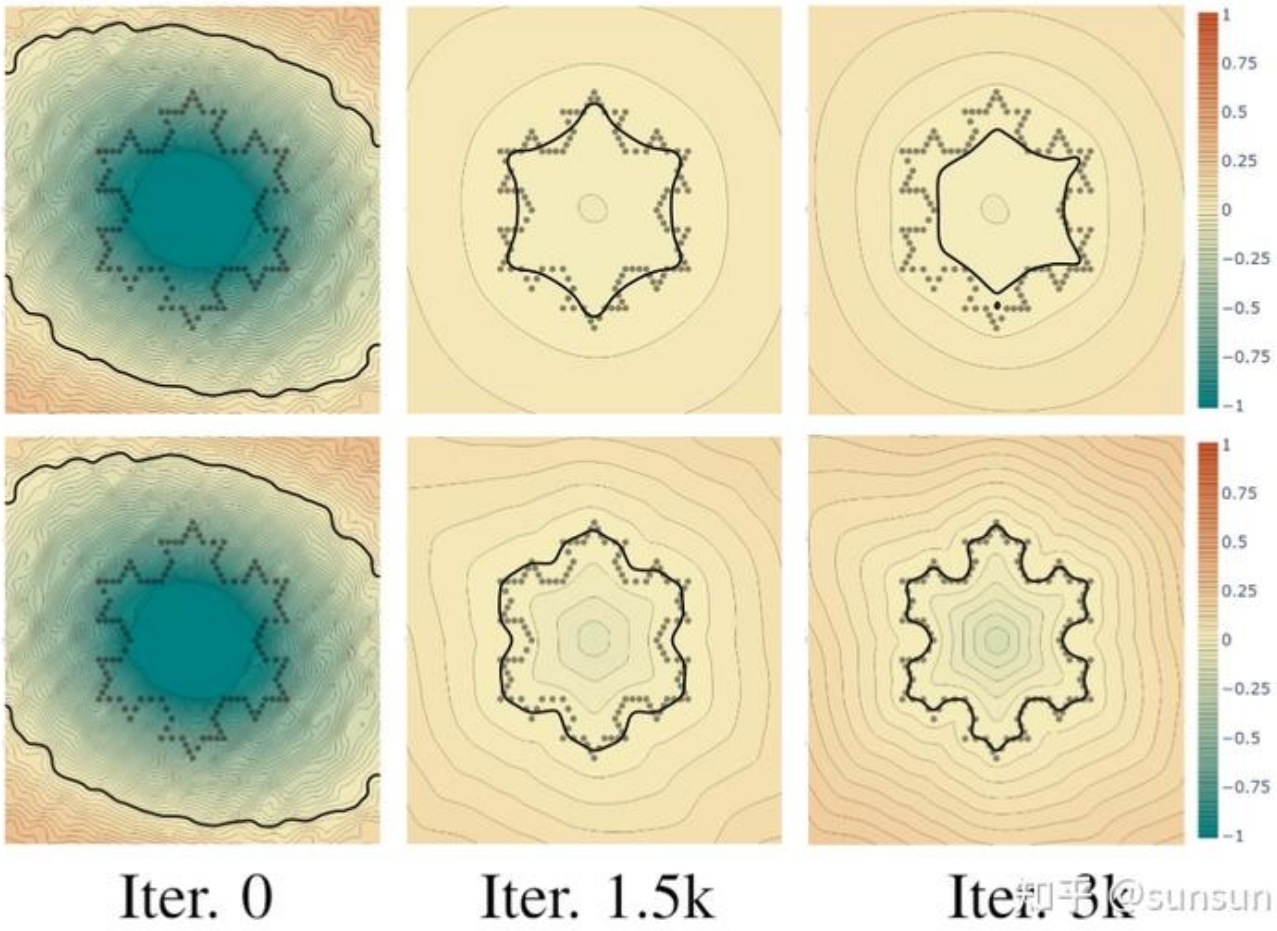
> Visual demonstration of the eikonal instability in the INR. (a) shows the level set at the iteration, m, before an instability of the non-regularized SDF optimization. [Top row]: shows the level set at various subsequent iterations (b)-(d) of the continued non-regularized SDF evolution. [Bottom row]: shows the level set at various iterations of the SDF evolution when adding our proposed regularization (directional divergence) after (a).

可以看到，在第一行Eikonal Loss的不稳定性导致的很糟糕的结果，第二行引入二阶约束后效果得到提升。

New Shape Regularization

前面提到DiGS中引入的 L_{div}能稳定Eikonal Loss的优化过程，得到更好的结果。然而我们发现，一个好的SDF其实只需要其法线方向上的二阶导数为0，如果在切线方向上的二阶导数为0的话，得到的SDF轮廓会非常平滑，不利于学习到一些细节。 L_{div}同时惩罚法线和切线方向上的二阶导数，所以DiGS将很难学习到一些几何细节。所以我们提出了一

个新的二阶约束，只惩罚法线方向上的二阶导数：
$L_{\mathrm{L.\,n.}}(u) = \int_{\Omega} |\nabla u(x)^T D^2 u(x) \cdot \nabla u(x)| dx$.在二维上的示例如下图所示，上面的是 L_{div}的实验结果，下面是我们新提出的 $L_{\mathrm{L.\,n.}}$，可以看到 L_{div}是over-smooth的。



> Illustration of the ability of our new regularization to capture fine-scale details of shape while still stabilizing the optimization. Without a penalty on the mean curvature, our directional divergence term restores the shape more quickly and captures fine details (bottom). On the other hand, the full divergence term (top) unnecessarily minimizes the mean curvature of the level sets, resulting in over-smoothness. Note the dark black lines represent the zero level set (lighter ones indicate other

> level sets). Ground truth is a snow-flake like shape (dotted gray). Note that both divergence terms prevent minimizing the eikonal loss too early, thereby avoiding sharp local minimizers.

New Shape Representation

MLP大多使用linear layer作为basic block，而一些文章中已经提到了ReLU based MLP中存在spectral bias，即更容易学习到低频信息。SIREN在这基础上提出了Sinosoidal activation，使网络能更快速地学习到高频信息。那能不能再进一步呢?

我们注意到activation其实是将domain分割成不同的区域，如果使用的是linear layer的话在每个区域里都接近于一个linear function，因为多层linear layer串联在一起还是一个linear layer，所以MLP表示的函数是 piecewise-linear的。我们意识到多层quadratic layer串联起来可以表示更高阶次的函数，比如两层quadratic layer串联起来就是一个四次函数。其实已经有很多文章研究过quadratic layer了，但大多应用于传统的机器学习/计算机视觉任务中，效果不是很惊艳，也没有获得很多关注，而我们首次引入quadratic layer用来表示Neural SDF。原始的二次函数 $x^T Ax + Bx + C$在pytorch中运算速度过慢，我们使用了最近一篇文章[9]中提出的高效的quadratic layer：

$(W_1\mathbf{x} + \mathbf{b_1}) \circ (W_2\mathbf{x} + \mathbf{b_2}) + W_3\mathbf{x}^2 + \mathbf{b_3}$其中 ∘表示的是element-wise product。我们使用的网络结构是用quadratic layer替换了SIREN中的linear layer。

Experiments

我们在Surface Reconstruction Benchmark，ShapeNet，Scene reconstuction数据集上测试了我们的实验结果，均达到了state of the art的结果。

Method	GT d_C	GT d_H	Scans $d_{\vec{C}}$	Scans $d_{\vec{H}}$
IGR wo n	1.38	16.33	0.25	2.96
SIREN wo n	0.42	7.67	0.08	**1.42**
SAL[1]	0.36	7.47	0.13	3.50
IGR+FF[17]	0.96	11.06	0.32	4.75
PHASE+FF[17]	0.22	4.96	**0.07**	1.56
DiGS[14]	0.19	3.52	0.08	1.47
Our StEik	**0.180**	**2.800**	0.096	1.454

Table 1: Quantitative results on the Surface Reconstruction Benchmark[28] using only point data (no normals).

Method	squared Chamfer ↓ mean	squared Chamfer ↓ median	squared Chamfer ↓ std	IoU ↑ mean	IoU ↑ median	IoU ↑ std
SIREN wo n	3.08e-4	2.58e-4	3.26e-4	0.3085	0.2952	0.2014
SAL[1]	1.14e-3	2.11e-4	3.63e-3	0.4030	0.3944	0.2722
DiGS[14]	1.32e-4	2.55e-5	4.73e-4	0.9390	0.9764	0.1262
Ablation (of Regularizations & Linear vs Quad Layers)						
Lin+$L_{\text{L. n.}}$	1.71e-4	1.23e-5	1.20e-3	0.9586	0.9809	0.0993
Qua+L_{div}	**5.45e-5**	1.05e-5	3.60e-4	0.9593	**0.9852**	0.1130
Our StEik (Qua+$L_{\text{L. n.}}$)	6.86e-5	**6.33e-6**	**3.34e-4**	**0.9671**	0.9841	**0.0878**

Table 2: Quantitative results on the ShapeNet[29] using only point data (no normals).

通过visual result可以很清楚的看到我们的方法可以恢复一些复杂精细的几何结构。

Figure 3: Example Visual results on ShapeNet [29]: We manifest the effectiveness of the new regularization and the new representation of Neural SDFs independently. Furthermore, the combination of two modules demonstrates an extra improvement. See supplement for more visual results.

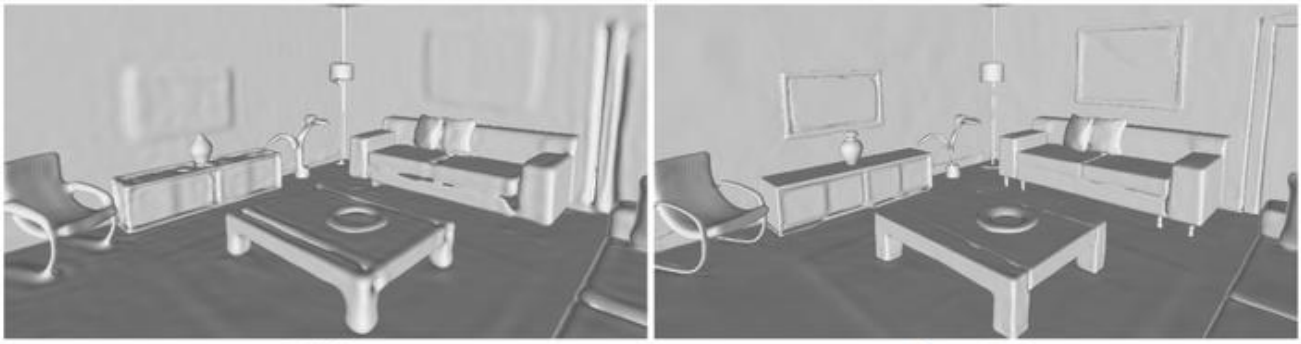


Figure 4: Visual results on the Scene Reconstruction Benchmark using only point data (no normals).

模型大小与速度：

Method	Structure	Runtime	Parameters
DiGS	5×256	37.86ms	0.26M
Lin+$L_{\text{L.n.}}$	5×256	32.52ms	0.26M
Qua+L_{div}	5×128	50.92ms	0.20M
Ours	5×128	42.20ms	0.20M
DiGS	8×512	63.28ms	1.84M
Lin+$L_{\text{L.n.}}$	8×512	50.90ms	1.84M
Qua+L_{div}	8×256	100.27ms	1.39M
Ours	8×256	80.62ms	1.39M

可以看到我们提出的 $L_{\text{L. n.}}(u)$计算速度是比 L_{div}要快的，因为我们使用了一种高效计算二阶导数的方式，感兴趣的可以直接看我们的源码。quadratic layer可以在参数量仅为linear layer的3/4的情况下达到更好的结果，然而计算速度要慢很多，这是因为pytorch中没有集成

quadratic layer模块，希望quadratic layer获得更多关注后也能集成到pytorch中。

参考

1. ^https://ieeexplore.ieee.org/document/937499
2. ^https://arxiv.org/abs/1901.05103
3. ^https://arxiv.org/abs/1911.10414
4. ^https://arxiv.org/abs/2006.05400
5. ^https://arxiv.org/abs/2002.10099
6. ^https://math.berkeley.edu/~sethian/2006/Papers/sethian.siam_fast.pdf
7. ^https://www.sciencedirect.com/science/article/abs/pii/016727899190197H
8. ^https://arxiv.org/abs/2106.10811
9. ^https://arxiv.org/abs/1808.00098