NeRF 的原理及其复现

B. Mildenhall, P. Srinivasan, Matthew Tancik, J. Barron, R. Ramamoorthi, Ren Ng

摘要

我们提出了一种方法，该方法通过使用一组稀疏的输入视图优化底层连续体积场景函数，从而获得用于合成复杂场景的新颖视图的最先进结果。我们的算法使用全连接（非卷积）深度网络表示场景，其输入是单个连续 5D 坐标（空间位置 (x, y, z) 和观察方向 (θ, φ)），输出是体积密度以及在该空间位置依赖于视图的发射辐射。我们通过沿相机光线查询 5D 坐标来合成视图，并使用经典的体积渲染技术将输出颜色和密度投影到图像中。因为体绘制是自然可微的，所以优化我们的表示所需的唯一输入是一组具有已知相机姿势的图像。我们描述了如何有效地优化神经辐射场以渲染具有复杂几何形状和外观的场景的逼真新视图，并展示了优于先前神经渲染和视图合成工作的结果。

关键词：场景表示；视图合成；体渲染；3D 深度学习

1 引言

本人的研究方向是就是 NeRF，这是一种隐式三维重建的方式。而所选的这篇论文则是 NeRF 的开山之作，之后所有 NeRF 相关的论文都是以这篇论文为基础，在某些方面进行改进和扩展。因此复现这篇论文的代码对我理解 NeRF 这个领域相关的论文将会有很大的帮助。

2 相关工作

计算机视觉最近一个有希望的方向是在 MLP 的权重中编码对象和场景，它直接从 3D 空间位置映射到形状的隐式表示，例如该位置的符号距离[1]。然而，到目前为止，这些方法无法再现具有复杂几何形状的真实场景，其保真度与使用离散表示（如三角形网格或体素网格）表示场景的技术相同。在本节中，我们回顾了这两条工作线并将它们与我们的方法进行对比，我们的方法增强了神经场景表示的能力，以产生用于渲染复杂现实场景的最先进结果。

2.1 神经 3D 形状表示

最近的工作通过优化将 xyz 坐标映射到带符号的距离函数[2]或占用场[3]的深度网络，研究了连续 3D 形状作为水平集的隐式表示。然而，这些模型受到访问地面真实 3D 几何的要求的限制，这些几何通常是从 ShapeNet[4]等合成 3D 形状数据集获得的。随后的工作通过制定可微渲染函数放宽了对地面真实 3D 形状的要求，这些函数允许仅使用 2D 图像优化神经隐式形状表示。尼迈耶等人[5]将表面表示为 3D 占用场，并使用数值方法为每条射线找到表面交点，然后使用隐式微分计算精确导数。然后将每个光线交叉位置作为神经 3D 纹理场的输入提供，该场预测该点的漫反射颜色。西茨曼等人。[6]使用不太直接的神经 3D 表示，在每个连续的 3D 坐标处简单地输出一个特征向量和 RGB 颜色，并提出一个可区分的渲染函数，该函数由一个循环神经网络组成，该网络沿着每条射线行进以确定表面的位置。尽管这些技术可以潜在地表示任意复杂和高分辨率的场景几何形状，但到目前为止，它们仅限于具有低几何复杂度的简单形状，从而产生过度平滑的渲染视图。我们展示了一种优化网络以编码 5D

辐射场（具有 2D 视图相关外观的 3D 体积）的替代策略可以表示更高分辨率的几何形状和外观，以呈现复杂场景的逼真新颖视图。

2.2 视角合成和基于图像的渲染

计算机图形学界通过从观察到的图像预测传统几何和外观表示，在逼真的新颖视图合成方面取得了重大进展。一类流行的方法使用基于网格的场景表示，具有漫射[7]或视图相关[8-9]外观。可微光栅化器[10-11]或路径追踪器[12]可以直接优化网格表示以使用梯度下降再现一组输入图像。然而，基于图像重投影的基于梯度的网格优化通常很困难，这可能是因为局部最小值或损失景观条件差。此外，该策略需要提供具有固定拓扑结构的模板网格作为优化前的初始化[12]，这通常不适用于不受约束的现实世界场景。另一类方法使用体积表示来专门解决从一组输入 RGB 图像合成高质量照片级真实感视图的任务。体积方法能够逼真地表示复杂的形状和材料，非常适合基于梯度的优化，并且往往比基于网格的方法产生更少的视觉干扰伪影。早期的体积方法使用观察到的图像直接为体素网格着色[13]。最近，一些方法[14]使用多个场景的大型数据集来训练深度网络，从一组输入图像中预测采样体积表示，然后使用 alpha 合成[15]射线以在测试时呈现新颖的视图。其他作品针对每个特定场景优化了卷积网络 (CNN) 和采样体素网格的组合，这样 CNN 可以补偿来自低分辨率体素网格的离散化伪影[16]或允许预测的体素网格根据输入时间而变化或动画控制。虽然这些体积技术在新颖的视图合成方面取得了令人印象深刻的结果，但由于它们的离散采样，它们扩展到更高分辨率图像的能力从根本上受到时间和空间复杂性差的限制——渲染更高分辨率的图像需要更精细的 3D 空间采样。我们通过在深度全连接神经网络的参数内编码连续体积来规避这个问题，这不仅比以前的三维重建方法产生更高质量的渲染，而且只需要那些采样体积表示的存储成本的一小部分。

3 本文方法

3.1 本文方法概述

根据从不同视角下拍摄的照片以及这些照片的相机位姿，可以生成以相机光心为起点，射向物体表面各像素点的光线。对于每条光线，都在这条光线上取一些采样点，将每个采样点的位置以及它所在光线的方向作为神经网络的输入，而输出则是每个采样点的颜色和体密度，根据同一条光线上不同采样点的体密度，对这些采样点的颜色进行一个加权平均操作，得到的就是物体表面像素点的颜色。当我们得到物体表面所有像素点的颜色之后，也就得到了这个物体的三维模型。整体流程如图 1所示：

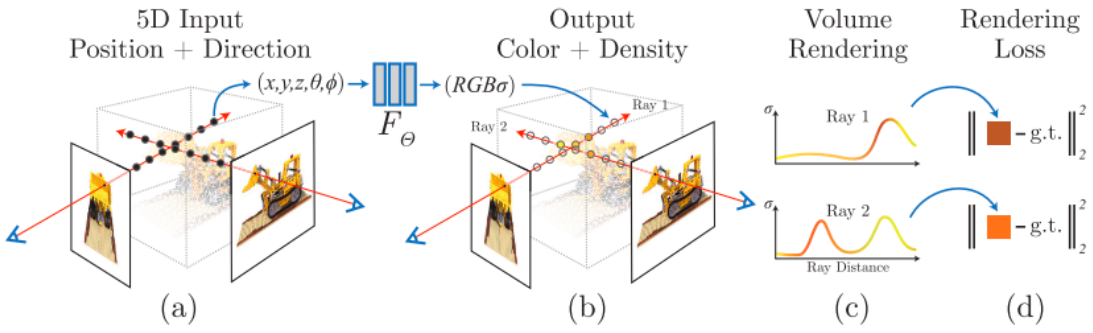

图 1: 整体流程图

3.2 数据加载模块

原始图片如果分辨率太高的话需要先进行一个下采样操作，降低分辨率，这样可以减少计算量；输入的图片是从不同视角下拍摄的，但是我们还需要生成一些新视角下的相机位姿，后续也对这些新视角下的光线进行渲染，这是为了使视角更密集，让最终得到的三维模型看起来更连贯。除此之外我们还需要定义一个世界坐标系原点，我们最终的光线方向以及坐标点位置都是以这个世界坐标系原点作为参考的。

3.3 生成光线模块

对于一张图片中的所有像素，每个像素都有一条光线穿过它，这条光线的原点就是相机光心，而光线的方向则可以由像素在图片中的位置以及相机的焦距来得到。在生成光线阶段，涉及到不同坐标系之间坐标的转换，比如相机坐标系到世界坐标系的转换。各坐标系间的转换关系如图 2所示。

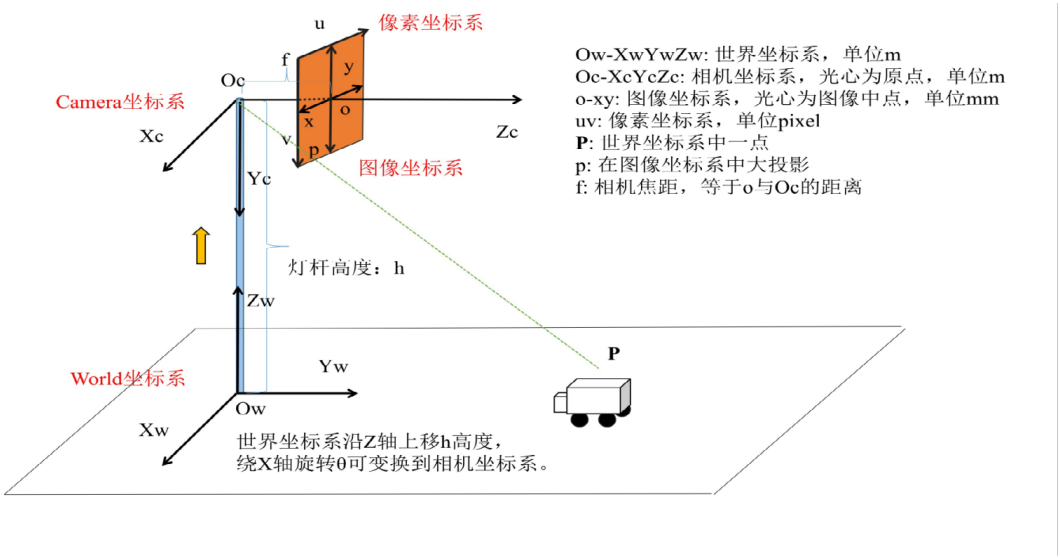

图 2: 各坐标系间的转换关系

3.4 神经网络模块

神经网络的结构如图 3所示。它的输入是光线上各个采样点的坐标以及光线方向，输出是对应采样点的颜色和体密度。其中体密度只和点的位置有关，因此它在光线方向输入神经网络之前就输出了，而颜色则和采样点的位置以及光线方向都有关系。

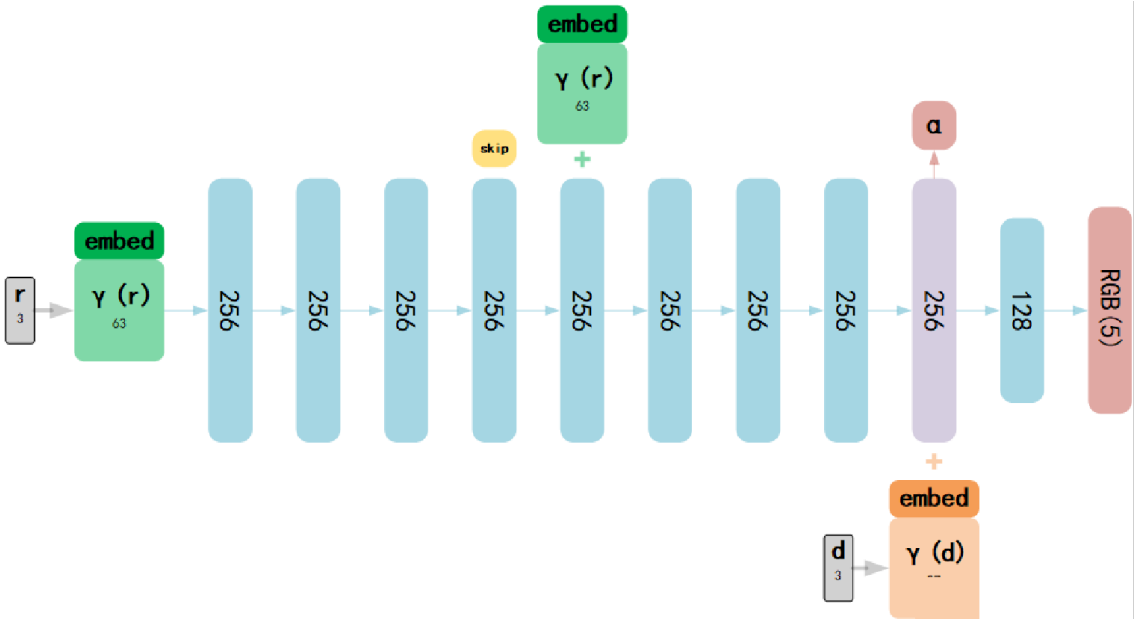

图 3: 神经网络结构图

3.5 体渲染模块

在生成光线之后，采样过程分为两步。第一步是粗采样，采样点比较稀疏，根据这些稀疏采样点的体密度可以更精细地确定物体表面像素点的位置范围。第二步是细采样，细采样就是在粗采样所确定的更精细的位置范围内进行一个密集的采样，得到各细采样点的颜色和密度，最终得到物体表面的颜色。

4 复现细节

4.1 与已有开源代码对比

本次复现的论文有好几个版本的开源代码，其中我引用了 yenchenlin 版本的代码，如图 4所示。

图 4: 引用的代码

阅读完该版本的开源代码后，发现写得真的是通俗易懂而且简洁。但是在仔细阅读的过程中，我发现它用的损失函数不是最好的，因此对它的损失函数进行了改进。它用的是 l2 损失函数，而我把它改为了 smooth l1 损失函数。l2 损失函数和 smooth l1 损失函数的表达式分别如图 5和图 6所示。

$$loss(x, y) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$$

图 5: l2 损失函数

$$loss(x,y) = \frac{1}{n}\sum_{i=1}^{n}\begin{cases} .5 * (y_i - f(x_i))^2, & if\ |y_i - f(x_i)| < 1 \\ |y_i - f(x_i)| - 0.5, & otherwise \end{cases}$$

图 6: smooth l1 损失函数

smooth l1 损失函数的优点如下：(1) 真实值和预测值差别较小时（绝对值差小于 1），梯度也会比较小（损失函数比普通 L1 loss 在此处更圆滑）。(2) 真实值和预测值差别较大时，梯度值足够小（普通 L2 loss 在这种位置梯度值就很大，容易梯度爆炸）。不同损失函数间的对比图如图 7所示。

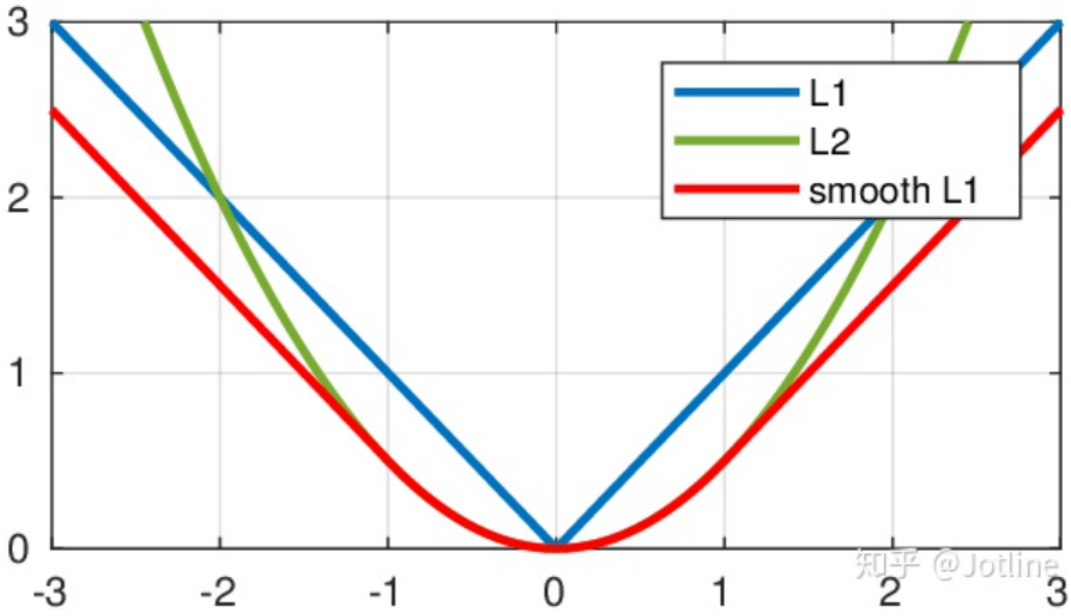

图 7: 不同损失函数的对比图

5 实验结果分析

对原始开源代码进行改进后，使用相同的数据集分别对改进前和改进后的代码进行了测试并对比了它们的训练结果。

改进之后发现 PSNR 有一定的提高。图 8为改进前的 PSNR，图 9为改进后的 PSNR。

```
[TRAIN] Iter: 197100 Loss: 0.0029270814266055822  PSNR: 31.955753326416016
[TRAIN] Iter: 197200 Loss: 0.002979918150231242  PSNR: 31.493391036987305
[TRAIN] Iter: 197300 Loss: 0.002497459761798382  PSNR: 33.05510330200195
[TRAIN] Iter: 197400 Loss: 0.002928572241216898  PSNR: 32.89718246459961
[TRAIN] Iter: 197500 Loss: 0.0027041391003876925  PSNR: 33.90538024902344
[TRAIN] Iter: 197600 Loss: 0.0033657681196928024  PSNR: 30.886119842529297
[TRAIN] Iter: 197700 Loss: 0.0021026243921369314  PSNR: 33.70112991333008
[TRAIN] Iter: 197800 Loss: 0.002073601819574833  PSNR: 33.288761138916016
[TRAIN] Iter: 197900 Loss: 0.0025783516466617584  PSNR: 31.47218894958496
[TRAIN] Iter: 198000 Loss: 0.0036639380268752575  PSNR: 32.13809585571289
[TRAIN] Iter: 198100 Loss: 0.0028536913450807333  PSNR: 34.858577728271484
[TRAIN] Iter: 198200 Loss: 0.0035988292656838894  PSNR: 31.677631378173828
[TRAIN] Iter: 198300 Loss: 0.004609433468431234  PSNR: 30.220083236694336
[TRAIN] Iter: 198400 Loss: 0.0037269508466124535  PSNR: 32.2530517578125
[TRAIN] Iter: 198500 Loss: 0.002760181901976466  PSNR: 33.08586883544922
[TRAIN] Iter: 198600 Loss: 0.0032552124466747046  PSNR: 32.202510833740234
[TRAIN] Iter: 198700 Loss: 0.002235875464975834  PSNR: 33.301246643066406
[TRAIN] Iter: 198800 Loss: 0.0022488771937787533  PSNR: 33.728389739990234
[TRAIN] Iter: 198900 Loss: 0.0020704909693449736  PSNR: 34.86151123046875
[TRAIN] Iter: 199000 Loss: 0.003736931597813964  PSNR: 28.3499813079834
[TRAIN] Iter: 199100 Loss: 0.0034432262182235718  PSNR: 30.050432205200195
[TRAIN] Iter: 199200 Loss: 0.0034996597096323967  PSNR: 30.180646896362305
[TRAIN] Iter: 199300 Loss: 0.004681784193962812  PSNR: 31.475528717041016
[TRAIN] Iter: 199400 Loss: 0.0020590582862496376  PSNR: 34.32799530029297
[TRAIN] Iter: 199500 Loss: 0.002692688722163439  PSNR: 32.205482482910156
[TRAIN] Iter: 199600 Loss: 0.0035289968363940716  PSNR: 30.310253143310547
[TRAIN] Iter: 199700 Loss: 0.0022698761895298958  PSNR: 32.62821578979492
[TRAIN] Iter: 199800 Loss: 0.002840711735188961  PSNR: 32.355384826660156
```

图 8: 改进前 PSNR

```
[TRAIN] Iter: 138300 Loss: 0.0012086732313036919  PSNR: 36.445804595947266
[TRAIN] Iter: 138400 Loss: 0.0015927978092804551  PSNR: 35.00798416137695
[TRAIN] Iter: 138500 Loss: 0.001388766337186098  PSNR: 35.4750862121582
[TRAIN] Iter: 138600 Loss: 0.0017474447377026081  PSNR: 34.15349197387695
[TRAIN] Iter: 138700 Loss: 0.0011903642443940043  PSNR: 35.600074768066406
[TRAIN] Iter: 138800 Loss: 0.0014813824091106653  PSNR: 35.87008285522461
[TRAIN] Iter: 138900 Loss: 0.0014936252226430595  PSNR: 36.2629508972168
[TRAIN] Iter: 139000 Loss: 0.0020041095558553934  PSNR: 32.90184783935547
[TRAIN] Iter: 139100 Loss: 0.0014628991484642029  PSNR: 35.517486572265625
[TRAIN] Iter: 139200 Loss: 0.0012061558663845062  PSNR: 34.348018646240234
[TRAIN] Iter: 139300 Loss: 0.0018430135678499937  PSNR: 33.72639465332031
[TRAIN] Iter: 139400 Loss: 0.0009510292438790202  PSNR: 36.012325286865234
[TRAIN] Iter: 139500 Loss: 0.0015842549037188292  PSNR: 34.67888641357422
[TRAIN] Iter: 139600 Loss: 0.0025565265677869324  PSNR: 32.61853790283203
[TRAIN] Iter: 139700 Loss: 0.0013808741932734847  PSNR: 34.64133071899414
[TRAIN] Iter: 139800 Loss: 0.0014932269696146254  PSNR: 35.83865737915039
[TRAIN] Iter: 139900 Loss: 0.0014243272598832846  PSNR: 35.994388580322266
 60%|
Saved checkpoints at ./logs/blender_paper_lego/140000.tar
[TRAIN] Iter: 140000 Loss: 0.0014261107426136732  PSNR: 34.272159576416016
[TRAIN] Iter: 140100 Loss: 0.0011826166883111  PSNR: 36.0952262878418
[TRAIN] Iter: 140200 Loss: 0.0018392833881080015  PSNR: 33.25590133666992
[TRAIN] Iter: 140300 Loss: 0.0015854444354772568  PSNR: 35.79493713378906
[TRAIN] Iter: 140400 Loss: 0.0016642652917653322  PSNR: 35.06460952758789
[TRAIN] Iter: 140500 Loss: 0.0013570500304922462  PSNR: 35.25355911254883
[TRAIN] Iter: 140600 Loss: 0.001544756698422134  PSNR: 35.40174865722656
[TRAIN] Iter: 140700 Loss: 0.0013994046021252870  PSNR: 35.7257194519043
[TRAIN] Iter: 140800 Loss: 0.0016139976214617491  PSNR: 34.240352630615234
```

图 9: 改进后 PSNR

改进前渲染出的三维模型截图如图 10所示，改进后的如图 11所示。

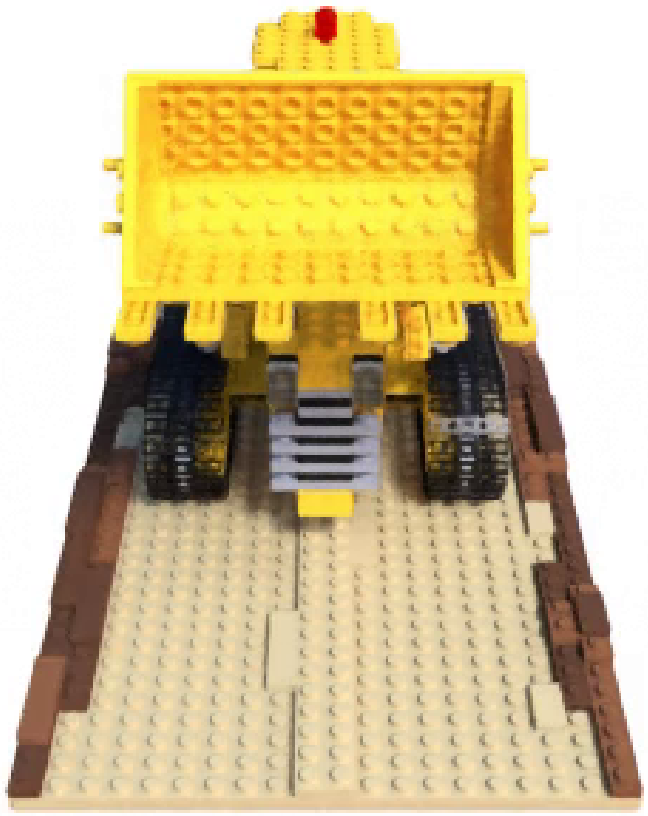

图 10: 改进前渲染出的三维模型

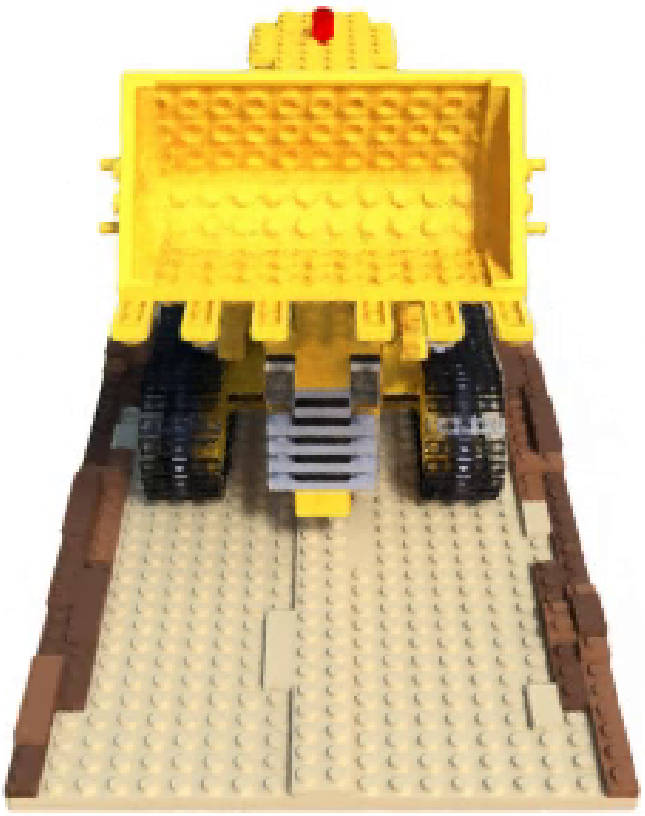

图 11: 改进后渲染出的三维模型

6 总结与展望

整篇文档主要介绍了 NeRF 这种隐式三维重建方式的实现原理以及过程，并记录了本人对 NeRF 已开源代码进行的一定的改进。实现过程中的不足之处主要有两点。第一，只在代码层面进行改进，对于 NeRF 这个技术本身则没有想到改进的点。第二，对代码的改进还是比较局限，局限在一些细节之处，没能对代码的整体框架进行大的改进。未来准备针对 NeRF 目前存在的许多不足之处，比如实时性，对动态物体的三维重建等方面进行深入研究，并尝试改进。

参考文献

[1] CURLESS B, LEVOY M. A volumetric method for building complex models from range images[J]. Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, 1996.

[2] PARK J J, FLORENCE P R, STRAUB J, et al. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation[J]. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019: 165-174.

[3] MESCHEDER L M, OECHSLE M, NIEMEYER M, et al. Occupancy Networks: Learning 3D Recon-

struction in Function Space[J]. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018: 4455-4465.

[4] CHANG A X, FUNKHOUSER T A, GUIBAS L J, et al. ShapeNet: An Information-Rich 3D Model Repository[J]. ArXiv, 2015, abs/1512.03012.

[5] NIEMEYER M, MESCHEDER L M, OECHSLE M, et al. Differentiable Volumetric Rendering: Learning Implicit 3D Representations Without 3D Supervision[J]. 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019: 3501-3512.

[6] SITZMANN V, ZOLLHOEFER M, WETZSTEIN G. Scene Representation Networks: Continuous 3D-Structure-Aware Neural Scene Representations[J]. ArXiv, 2019, abs/1906.01618.

[7] WAECHTER M, MOEHRLE N, GOESELE M. Let There Be Color! Large-Scale Texturing of 3D Reconstructions[C]//European Conference on Computer Vision. 2014.

[8] DEBEVEC P E, TAYLOR C J, MALIK J. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach[J]. Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, 1996.

[9] WOOD D N, AZUMA D I, ALDINGER K R, et al. Surface light fields for 3D photography[J]. Proceedings of the 27th annual conference on Computer graphics and interactive techniques, 2000.

[10] CHEN W, GAO J, LING H, et al. Learning to Predict 3D Objects with an Interpolation-based Differentiable Renderer[J]. ArXiv, 2019, abs/1908.01210.

[11] LOPER M, BLACK M J. OpenDR: An Approximate Differentiable Renderer[C]//European Conference on Computer Vision. 2014.

[12] LI T M, AITTALA M, DURAND F, et al. Differentiable Monte Carlo ray tracing through edge sampling [J]. ACM Transactions on Graphics (TOG), 2018, 37: 1-11.

[13] KUTULAKOS K N, SEITZ S M. A Theory of Shape by Space Carving[J]. International Journal of Computer Vision, 1999, 38: 199-218.

[14] FLYNN J, BROXTON M, DEBEVEC P E, et al. DeepView: View Synthesis With Learned Gradient Descent[J]. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019: 2362-2371.

[15] PORTER T K, DUFF T. Compositing digital images[J]. Proceedings of the 11th annual conference on Computer graphics and interactive techniques, 1984.

[16] SITZMANN V, THIES J, HEIDE F, et al. DeepVoxels: Learning Persistent 3D Feature Embeddings[J]. 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018: 2432-2441.