大家好，我们小组此次project的题目是基于神经辐射场的简单场景编辑，主要做的是论文的一个简单的复现工作。

我们此次展示将会从以下几个方面展开。首先我们会介绍一下神经辐射场，也就是NeRF的原理和网络架构，然后是我们的场景编辑工作和一些问题可能的改进方法。

首先是NeRF的原理。作为最近两三年才兴起的一种三维重建方式，NeRF不像点云、网格等方法来显式表征场景，也就是那种，重建完成后就可以让我们实时看到场景长什么样子的方式，而是使用神经网络场对场景进行隐式表征，也就是说，NeRF是将学习到的场景信息保存在网络之中，当我们去查询网络时，才会将场景的信息输出。这里右下角展示的就是NeRF的一个重建效果，左边是rgb颜色图，右边是深度图，可以看到，它的场景重建质量很高，因此也可能被广泛应用到VR、AR等技术之中。我们此次的project，都是在NeRF的模型基础上完成，因此有必要先讲解一下NeRF的基本思想。

总的来说，NeRF的输入是空间中一个点的坐标以及这个点所处射线的方向。对训练集中的每一张图片，我们先根据图像的位姿来算出图像每一个像素点对应的射线，这里的这个射线可以理解成我们人眼看物体的那种视线。我们会在射线上采点，把点的坐标加上这条射线方向组成的5D向量输入网络，得到这个点对应的颜色RGB和体密度。Emm，这里的有比较明确的物理含义，表示的是一条射线在这个点处终止的微分概率。然后利用作者提出的体渲染方程，我们就可以将射线上点的颜色做一个积分求和，得到这条射线，也就是这个像素点的颜色。通过最小化合成出来颜色与ground truth之间的均方误差，我们就可以达到优化网络的目的。

然而，作者发现这种方法存在网络对图像高频变化不敏感的问题，所以又引入了一个位置编码来做高维映射。具体的使用方法就是在输入网络之前，对点的位置x和射线方向d做位置编码，最后得到一个60维和24维的向量。

接下来就是NeRF的网络结构。它的网络比较简单，只用到了一个MLP。因为体密度只与点的位置有关，颜色c与位置和射线方向都有关，因此，我们先使用8个256通道的全连接层处理点的坐标，得到体密度和一个256维的特征向量，再把这个特征向量和射线方向连接起来输入两个全连接层，得到3维的颜色输出。

Ok，NeRF的架构大概就是以上这些，我们接下来将会讲解我们在场景编辑方面的工作，主要是对这篇Object-NeRF做了一些简化和复现。总的来说，Object-NeRF提出了一种场景分解的方法，也就是使用一个分支scene branch来重建场景，用另一个分支object branch来重建单个物体。当然，图中展示的是Object-NeRF的原始版本，我们在此基础上做了简化，主要是去掉了体素特征的输入，然后只考虑单个物体的提取。这就使得scene和object这两个分支实际上和原始NeRF比较接近，只不过还是要有一些修改。首先，scene branch是对场景的全景学习，基本上就是一个普通的NeRF；而object branch需要把物体提取出来，因此需要变化对它的监督方式。

这里我们展示的就是object branch的损失函数，它除了考虑颜色这个损失以外，还引入了一个mask M(r)，加入了不透明度的损失。这里M(r) 的取值和像素点的所属关系有关。当射线对应的像素点不属于物体时，M(r) 就等于0，这时候颜色这一部分损失就等于0了，然后网络要做的就是缩小这条射线的不透明度，也就是让这条射线变透明，最后渲染出来的效果就是这个像素点完全是黑色；当射线对应的像素点属于物体时，M(r) 就等于1，这时候我们就要考虑这条射线的颜色，同时让它不透明，这就是这个损失函数的物理含义。

接下来就是我们实现的一个效果。上面这张图是我们使用原始NeRF对kitti-360进行测试的结果，然后下面这张图就是只考虑颜色约束，不考虑不透明度约束的渲染结果。可以看到，物体的渲染效果还是很好的，然后其他区域就比较混乱了。（点击）最后我们加上了不透明度约束，得到了下面这个结果。（点击）当然，当出现物体的遮挡时，会出现一些噪点，比较影响渲染质量。

然后接下来就开始涉及到我们场景编辑的工作了。首先讲一下我们的场景编辑的思想。因为普通NeRF渲染场景的时候，就是在每一个像素点对应的射线上采点、查询网络，然后用渲染方程合成颜色，那我们进行编辑的时候，采用的就是ray bending 的方法，就是把这条射线弯折，使它经过它原本不能经过的区域。这样的话，这条射线上点的颜色就会改变，从而影响渲染出来的像素点的颜色。（翻前页）在我们这个项目中，因为我们维护了一个object 分支，它只会渲染物体，然后其他区域都透明，那我们可以对object分支的射线做弯折，然后scene分支的射线保持不变。这样的话，如果一条射线原本只会射中街道，在object分支里面做了弯折以后，它将会经过这辆车。我们把scene和object两个分支得到的采样点的信息结合起来进行渲染，此时就会在未弯折射线对应的像素点处渲染出来这辆车的颜色。（翻页）

这是我们理想的情况，但观察我们刚刚渲染出来图像的深度图，上面这个是scene分支的结果，下面这个是object分支的结果，很明显两个分支的深度是不一样的，这样的话，场景的前后景相对关系就不一致了，采用ray bending的方法渲染出来的结果就会变成街道在前面，而我们复制过去的车在后面，显然是不行的。

为了将两个分支的深度统一，我们需要对他们进行联合优化。具体来说就是训练的时候对两个网络输入相同射线上的相同采样点，然后把它们各自的损失加起来，联合优化。

经过这样的处理以后，两个分支就耦合起来的，然后尺度也会统一，最后得到的PSNR和分开训练得到的差不多。

最后给大家放一下我们简单场景编辑的效果。首先上面这张图是object分支单独渲染的结果（点击），然后这是我们对object分支单独编辑的效果，下面这张图就是我们把object分支的车平移之后和scene分支返回的信息进行结合渲染出来的结果。

最后是我们对一些存在的问题提出的further solutions，可以再在我们现在这个结果的基础上再做一些完善。首先，我们可以看到场景的渲染质量还是可以提高的，这个可以通过在网络输入加上体素编码来改进。另外一个比较重要的是物体质量的提高。其实通过这里这个动图可以看到，物体只占画面很小一部分面积的时候，渲染出来是很花的。我们觉得这主要是因为我们训练时是随机取的射线的batch，这样当物体只在画面很小一部分出现时，其实有更多的射线都是取的非物体区域，这样其实很多信息都被浪费掉了，所以取batch的时候可以调整一下策略，确保物体区域有足够多的射线。然后剩下的一些就是功能上的拓展了，比如更复杂的编辑方式等等。

Ok，我们的展示就是这些，谢谢大家！