## camera\_util

### get\_points\_from\_angles 函数解释

这个函数用于从给定的距离、仰角和方位角计算点的坐标。可以处理单个浮点数或整数的距离，也可以处理张量形式的距离。让我们分步骤分析：

· **函数参数**

* · distance: 距离，可能是浮点数、整数或张量。
* elevation: 仰角。
* azimuth: 方位角。
* degrees: 一个布尔值，表示仰角和方位角是否以度数表示，默认为 True。

· **处理单个距离值**

* · 如果 distance 是浮点数或整数：
  + 如果 degrees 为 True，将仰角和方位角从度数转换为弧度（使用 math.radians）。
  + 计算并返回相应的坐标 (x, y, z)。

· **处理张量形式的距离值**

* · 如果 degrees 为 True，将仰角和方位角从度数转换为弧度（使用简单的数学运算 math.pi / 180）。
* 计算并返回相应的坐标张量 (x, y, z)，使用 torch.stack 将计算结果组合成一个张量，并转置以得到正确的形状。

### perspective 函数解释

这个函数用于计算顶点的透视变形。

· **函数参数**

* · vertices: 一个张量，表示一组顶点的坐标，应该有三个维度（批次大小、顶点数量、坐标数量）。
* angle: 透视角度，默认值为 30。

· **检查顶点张量的维度**

* · 如果 vertices 不是三维张量，则抛出一个错误。

· **计算透视变形**

* · 获取张量所在的设备（CPU 或 GPU）。
* 将角度从度数转换为弧度，并转换为张量。
* 计算视角宽度（即 tan(angle)）。
* 分别计算顶点的 x 和 y 坐标相对于 z 坐标和视角宽度的缩放。
* 使用 torch.stack 将计算结果组合成新的顶点张量。

### rotate\_vertices 函数解释

这个函数用于将顶点进行“Look at”变换，也就是根据相机的位置和方向旋转顶点。让我们逐步分析：

· **函数参数**

* · vertices: 一个张量，表示一组顶点的坐标，应该有三个维度（批次大小、顶点数量、坐标数量）。
* eye: 相机的位置。
* at: 相机的目标位置，默认为 [0, 0, 0]。
* up: 相机的上方向，默认为 [0, -1, 0]。

· **检查顶点张量的维度**

* · 如果 vertices 不是三维张量，则抛出一个错误。

· **处理输入参数**

* · 将 at、up 和 eye 转换为张量，并确保它们在同一设备上。

· **批量处理**

* · 获取批次大小。
* 如果 eye、at 或 up 只有一个维度，将其扩展为批次大小。

· **创建新坐标轴**

* · 计算 z 轴：从 eye 到 at 的单位向量。
* 计算 x 轴：up 与 z 轴的叉积的单位向量。
* 计算 y 轴：z 轴与 x 轴的叉积的单位向量。

· **创建旋转矩阵**

* · 将 x、y 和 z 轴组合成旋转矩阵。

· **应用旋转**

* · 将顶点减去 eye 的位置。
* 使用旋转矩阵进行旋转。

· **返回结果**

* · 返回旋转后的顶点、相机位置和旋转矩阵。

## datasets\_split\_view0\_dataloader

### 图像转换解释

* transforms.Resize(224): 将图像大小调整为 224x224 像素。
* transforms.ToTensor(): 将图像转换为张量。
* transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]): 使用给定的均值和标准差归一化图像张量。

### 类别映射解释

* class\_ids\_map: 映射 ShapeNet 类别 ID 到类别名称。

ShapeNet 数据集类解释

· \_\_init\_\_ **方法**

* · 初始化数据集实例，加载数据列表。
* 设置仰角和距离。

· \_\_len\_\_ **方法**

* · 返回数据集的长度，这里设置为一个非常大的值 99999999。

· \_\_getitem\_\_ **方法**

* · 根据索引获取数据。
* 读取图像和相关数据。
* 计算视点。
* 返回图像、视点、掩码、张量和精细图像数据。

### get\_eval\_img 函数解释

· **读取图像**

* · 使用 OpenCV 读取图像文件。

· **转换图像**

* · 将图像转换为浮点数并归一化到 [0, 1] 范围。
* 将图像转换为张量，并调整形状以适应模型输入。

· **计算视点**

* · 使用 camera\_util.get\_points\_from\_angles 函数计算视点。

· **返回结果**

* · 返回图像、掩码、张量和视点。

主程序解释

· **设置类别 ID**

* · 指定类别 ID 为 '03001627'（即椅子）。

· **创建数据集实例**

* · 使用 ShapeNet 类创建数据集实例。

· **创建数据加载器**

* · 使用 DataLoader 创建数据加载器，设置批次大小为 64，使用 16 个工作线程，不打乱数据。

· **处理数据批次**

* · 迭代数据批次并打印每个批次中各个部分的数据形状。
* 在第一个批次后退出程序。

## datasets\_split\_view0\_dataloader\_random

ShapeNet 数据集类解释

· \_\_init\_\_ **方法**

* · 初始化数据集实例，加载数据列表。
* 如果没有指定类别 ID，则加载所有类别的训练数据列表；否则，加载指定类别的训练数据列表。
* 设置仰角和距离，以及视角角度列表。

· \_\_len\_\_ **方法**

* · 返回数据集的长度，这里设置为一个非常大的值 99999999。

· \_\_getitem\_\_ **方法**

* · 根据随机索引获取数据。
* 读取图像和相关数据。
* 计算视点。
* 返回图像、视点、掩码、张量和精细图像数据。

主程序解释

· **创建数据集实例**

* · 使用 ShapeNet 类创建数据集实例，不指定类别 ID 和数据集名称。

· **创建数据加载器**

* · 使用 DataLoader 创建数据加载器，设置批次大小为 4，使用 4 个工作线程，不打乱数据。

· **处理数据批次**

* · 迭代数据批次并打印每个批次中各个部分的数据形状（这部分打印代码被注释掉了）。
* 在第一个批次后退出程序。

## datasets\_split\_view0\_dataloader\_testsplit

具体修改的解释

· **数据目录路径**

* · 更新了 DATA\_DIR 为新的路径 './data/test/viewangle\_%d\_224\_split\_test/'。

· **数据集文件路径**

* · 在 \_\_init\_\_ 方法中，打开对应的类别和数据集名称的文件 './data/data\_list/%s\_%s\_cat.txt'。

· **数据集长度**

* · 在 \_\_len\_\_ 方法中，返回数据列表的实际长度。

· **图像读取和处理**

* · 在 \_\_getitem\_\_ 方法中，根据索引读取图像，并转换为浮点数，归一化到 [0, 1] 范围。

## datasets\_split\_view0\_dataloader\_single-view-testsplit

· ShapeNetSingle **数据集类**

* · 在 \_\_init\_\_ 方法中，数据集实例化时会读取一个文件，这个文件包含单张图像的路径信息。
* 在 \_\_getitem\_\_ 方法中，根据索引从数据列表中读取单张图像，并进行处理。

· **主程序**

* · 创建 ShapeNetSingle 数据集实例，指定类别 ID 和数据集名称为 'train'。
* 使用 DataLoader 创建数据加载器，设置批次大小为 64，使用 16 个工作线程，不打乱数据。
* 迭代数据批次，并打印每个批次中图像、视点、掩码和张量的形状。

## Losses

这些损失函数主要用于 3D 模型的评估和优化，涉及到 IoU 损失、Chamfer 距离、平滑损失、对称性损失、聚类正则化等。这些函数结合了 Pytorch3D 中的邻近点计算功能（如 knn\_points 和 knn\_gather）来实现复杂的几何损失计算。通过这些损失函数，模型可以更好地学习三维结构和几何特性。

iou 和 iou\_loss

· **目的**：计算预测结果与目标之间的交并比（IoU），并将其转换为损失。

· **工作原理**：通过计算预测和目标之间的交集与并集之比，得到IoU，然后通过 1 - IoU 得到损失值。

· **应用场景**：用于衡量二值分割任务中预测结果与真实结果的重叠程度。

multiview\_iou\_loss 和 multiview\_iou\_fine\_loss

· multiview\_iou\_loss 计算多个视角的 IoU 损失，假设输入是四个不同视角的预测结果和对应的目标。

· multiview\_iou\_fine\_loss 类似，但在计算 IoU 前先应用了细化掩码。

· **目的**：计算多视角下的IoU损失，用于评估模型在多个视角下的表现。

· **工作原理**：对多个视角的预测结果与目标进行IoU计算，并取平均值作为损失。

· **应用场景**：多视角3D重建任务，用于确保模型在不同视角下的预测结果与真实结果的一致性。

split\_iou\_loss 和 multiview\_proj\_iou\_loss

· split\_iou\_loss 计算细化和表面预测的 IoU 损失，分别针对两个不同的目标。

· multiview\_proj\_iou\_loss 计算多视角投影的 IoU 损失。

· CD\_loss 计算 Chamfer 距离损失和半径损失。

· **目的**：计算两个点云之间的Chamfer距离以及球体半径之间的差异。

· **工作原理**：使用Chamfer距离衡量两个点云之间的相似度，并结合最近邻半径差异作为损失。

· **应用场景**：3D点云匹配与比较，用于评估模型生成的点云与目标点云之间的距离。

· regularize\_bias 计算偏置正则化损失。

· regularize\_radii 计算半径的正则化损失。

· **目的**：对球体中心和半径进行正则化，防止过拟合。

· **工作原理**：

* regularize\_bias：通过最近邻点之间的距离进行正则化，防止球体中心过度偏移。
* regularize\_radii：对球体半径进行正则化，防止半径过大或过小。

· **应用场景**：3D模型的几何正则化，确保模型生成的球体在空间上的合理分布和尺寸。

· edge\_loss 是一个占位符函数，未实现。

nearest\_smooth\_loss 和 smooth\_y\_loss

· nearest\_smooth\_loss 计算邻近平滑损失。

· smooth\_y\_loss 计算 y 方向上的平滑损失。

· **目的**：对球体的中心和半径进行平滑，确保相邻球体之间的平滑过渡。

· **工作原理**：

* nearest\_smooth\_loss：计算每个球体与其邻居之间的位置和半径差异。
* smooth\_y\_loss：专注于 y 轴方向上的平滑，确保相邻球体在 y 轴上的位置和半径差异在阈值内。

· **应用场景**：3D模型的平滑优化，确保生成的球体在空间上平滑过渡。

smooth\_vector\_loss 和 neighbors

· smooth\_vector\_loss 计算向量平滑损失。

· neighbors 获取邻近中心和半径的平均值。

· **目的**：计算球体中心和半径的平滑损失，并获取相邻球体的信息。

· **工作原理**：

* smooth\_vector\_loss：计算球体中心与其邻居之间的向量差异，并确保差异在阈值内。
* neighbors：获取相邻球体的中心和半径平均值。

· **应用场景**：3D模型的平滑优化，确保生成的球体在空间上平滑过渡。

· smooth\_loss\_1 计算平滑损失。

· tangent\_loss 计算切线损失。

· cluster\_reg\_loss 计算聚类正则化损失。

· symmetry\_loss 计算对称性损失。

· nearest\_r\_loss 计算最近半径损失。

· repulsion\_loss 计算排斥损失。

· repulsion\_loss\_2 计算另一种排斥损失。

egde\_laplacian\_loss 和 edge\_flatten\_loss

· egde\_laplacian\_loss 计算边拉普拉斯损失。

· edge\_flatten\_loss 计算边平滑损失。

· face\_regularization 计算面的正则化损失。

· face\_regularization\_min 计算最小半径正则化损失。

edge\_regularization\_near

· 计算边缘上的球体半径差的平方并取平均作为损失。

· knn\_gather 用于从 spheres 中提取球体的半径。

cd\_2d

· 计算二维点云之间的 Chamfer 距离。

· 将输入和预测值归一化到 [0, 1] 范围。

· chamfer\_distance 用于计算两个点云集合之间的距离。

chamfer\_1 和 chamfer\_2

· chamfer\_1 和 chamfer\_2 都是计算 Chamfer 距离的函数，但实现略有不同。

· 使用 knn\_points 计算最近邻点，并计算点之间的欧几里得距离。

cd\_2d\_1

结合 chamfer\_1 计算四种组合的 Chamfer 距离并取平均。

texture\_loss

计算预测和目标 RGB 值之间的平方误差损失，并求和作为纹理损失。

template\_loss

计算模板的法向量差异，并返回其平均值作为损失

kNNLoss 和 kNNSphereLoss

kNNLoss 和 kNNSphereLoss 计算最近邻点云的距离方差损失，使用最远点采样来选择种子点。

· **目的**：计算点云的k近邻损失。

· **工作原理**：

* 使用最远点采样选择种子点。
* 计算每个点到种子点的距离，并计算这些距离的方差作为损失。

· **应用场景**：点云的空间分布优化，确保点云在空间上的均匀分布。

SphereInterLoss

计算球体之间的相互作用损失，包括最近邻球体之间的距离和半径差异。

farthest\_point\_sample

实现最远点采样算法，选择点云中离当前最远点最远的点作为采样点。

square\_distance

计算源点和目标点之间的欧几里得平方距离。

query\_ball\_point

· 在指定半径内查询点，返回点的索引或密度矩阵。

· 如果 density\_only 为 True，只返回密度矩阵，否则返回组内点的索引。

kNNRepulsionLoss

· 计算k近邻点的排斥损失。

· 使用最远点采样选择种子点，并计算每个点到种子点的距离。

· 使用高斯核计算权重，并计算排斥力。

get\_fine\_loss

· 计算细化模型的总损失，包括多视角IoU损失、排斥损失、球体损失和最近邻半径损失。

· 使用配置文件中的权重调整各部分损失，并记录损失值。

get\_surf\_loss

类似于 get\_fine\_loss，计算表面模型的总损失

get\_joint\_loss

类似于 get\_fine\_loss 和 get\_surf\_loss，但针对联合模型，计算联合损失。

get\_finetune\_loss

计算和记录模型在微调阶段的损失。它结合了多视角IoU损失、细化损失和最近半径损失，并根据配置文件中的设置来确定是否计算和记录这些损失。通过这些步骤，函数能够帮助优化和评估模型在微调过程中的表现。

· **目的**：计算细化、表面和联合模型的总损失。

· **工作原理**：

* 包括多视角IoU损失、排斥损失、球体损失和最近邻半径损失。
* 根据配置文件中的权重调整各部分损失，并记录损失值。

· **应用场景**：用于细化、表面和联合模型的损失计算和优化。

## models\_merge\_circle\_resnet\_1113

Encoder 类

**功能**：编码器类将输入图像编码为高维特征向量。

**实现**：

* 三层卷积层，每层后接一个批量归一化层和 ReLU 激活函数。
* 三层全连接层将卷积后的特征图展平并进一步映射到指定的输出维度。
* 输入 dim\_in=4 表示输入图像有4个通道，输出 dim\_out=512 表示输出特征向量的维度为512。

Decoder 类

**功能**：解码器类将编码后的特征向量解码为3D球体的中心和半径，生成3D模型。

**实现**：

* 加载模板网格和球体网格，并注册为缓冲区。
* 两层全连接层将输入特征向量映射到球体中心和半径。
* 根据视角调整球体的位置。
* transfer\_circle 方法：根据视角调整球体位置。
* drop\_out 方法：按比例选择球体，用于丢弃部分球体。

Decoder\_DIS 类

**功能**：生成球体位移，用于微调球体半径。

**实现**：

* 使用 PointNet 编码器编码输入点云。
* 三层全连接层生成球体位移。

Model 类

**功能**：集成编码器和解码器，用于3D重建任务。

**实现**：

* 初始化包括两个编码器和解码器，以及一个用于生成位移的解码器。
* model\_param 和 model\_param\_finetune 方法返回模型参数。
* reconstruct 方法重建3D模型，返回顶点、面片、中心和半径等信息。
* reconstruct\_surf 方法重建表面模型。
* reconstruct\_semantic 方法根据语义重建模型。
* r\_finetune 方法微调球体半径。

初始化方法

· **初始化两个编码器和解码器**：

* encoder\_fine 和 decoder\_fine 处理高细节部分。
* encoder\_surf 和 decoder\_surf 处理表面部分。
* decoder\_displacements 用于生成位移，用于微调球体半径。

· **加载网格数据**：通过 utils.load\_obj\_spherenum 方法加载圆柱网格。

参数获取方法

· model\_param：返回所有模型参数，用于训练。

· model\_param\_finetune：返回微调阶段的模型参数。

重建方法

· reconstruct：对输入图像进行3D重建，返回顶点、面片、中心和半径等信息。

· reconstruct\_surf 和 reconstruct\_semantic：分别用于重建表面模型和语义模型。

微调方法

* r\_finetune：微调球体半径，通过计算位移和最近邻球体调整半径。

#### 辅助方法

* compute\_edges\_vf\_1：计算顶点和面片，通过最近邻球体生成边缘。
* get\_nearest\_edges：获取最近邻边缘。

#### 渲染方法

· render\_multiview：渲染多视图输入，生成3D模型并返回其相关信息。

· render\_multiview\_w\_fine 和 render\_multiview\_semantic：分别用于渲染包含细节的多视图和语义分割的多视图。

· pred\_spheres 和 render\_multiview\_split：分别用于预测球体和渲染分割的多视图。

· render\_neighbor：

* 功能：基于输入图像和视点渲染带有邻居信息的3D模型。
* 输入：image\_a, image\_b（输入图像），viewpoint\_a, viewpoint\_b（视点），neighbor\_num（邻居数量）。
* 输出：包含球体中心、半径、邻居索引等信息的字典。

· edge\_prediction：

* 功能：预测模型中的边缘。
* 输入：球体的中心和半径，k（邻居数量）。
* 输出：包含顶点和面的信息，以及邻居索引。

· get\_neighbors：

* 功能：获取给定中心和半径的球体的邻居信息。
* 输入：球体的中心和半径，k（邻居数量）。
* 输出：包含最近邻居的球体信息和邻居索引。

· compute\_edges\_vf：

* 功能：计算球体模型中的边缘顶点和面。
* 输入：球体的中心、半径和最近邻居信息。
* 输出：包含顶点和面的信息。

· render\_multiview\_progressive：

* 功能：基于输入图像和视点进行渐进式多视图渲染。
* 输入：image\_a, image\_b（输入图像），viewpoint\_a, viewpoint\_b（视点），current\_epoch（当前训练轮数），max\_epoch（最大训练轮数）。
* 输出：包含球体中心、半径、顶点和面的信息。

· render\_multiview\_split\_fine：

* 功能：基于输入图像和视点进行多视图渲染，输出精细模型。
* 输入：image\_a（输入图像），viewpoint\_a（视点）。
* 输出：包含精细模型的顶点和面信息。

· render\_multiview\_edges：

* 功能：基于输入图像和视点进行多视图渲染，输出边缘模型。
* 输入：image\_a, image\_b（输入图像），viewpoint\_a, viewpoint\_b（视点），aimuth\_a, aimuth\_b（方位）。
* 输出：包含球体模型的边缘信息。

· render\_multiview\_r\_finetune：

* 功能：基于输入图像和视点对球体模型进行微调。
* 输入：image\_a, image\_b（输入图像），viewpoint\_a, viewpoint\_b（视点），centers, radii（球体的中心和半径）。
* 输出：包含微调后的球体半径和模型顶点、面信息。

· evaluate：

* 功能：评估输入图像和视点生成的3D模型，输出球体中心、半径和微调后的球体信息。
* 输入：images（输入图像），viewpoints（视点）。
* 输出：包含球体中心、半径和微调后的球体信息。

· evaluate\_overview：

* 功能：评估输入图像和视点生成的3D模型，输出包含球体中心、半径和微调后的球体信息的概览。
* 输入：images（输入图像），viewpoints（视点）。
* 输出：包含球体中心、半径和微调后的球体信息。

· evaluate\_split：

* 功能：评估输入图像和视点生成的3D模型，输出精细模型和表面模型的信息。
* 输入：images（输入图像），viewpoints（视点）。
* 输出：包含精细模型和表面模型的顶点、面、球体中心和半径信息。

· render\_neighbor：

* · 功能：基于输入图像和视点渲染带有邻居信息的3D模型。
* 输入：image\_a, image\_b（输入图像），viewpoint\_a, viewpoint\_b（视点），neighbor\_num（邻居数量）。
* 输出：包含球体中心、半径、邻居索引等信息的字典。

· edge\_prediction：

* · 功能：预测模型中的边缘。
* 输入：球体的中心和半径，k（邻居数量）。
* 输出：包含顶点和面的信息，以及邻居索引。

· get\_neighbors：

* · 功能：获取给定中心和半径的球体的邻居信息。
* 输入：球体的中心和半径，k（邻居数量）。
* 输出：包含最近邻居的球体信息和邻居索引。

· compute\_edges\_vf：

* · 功能：计算球体模型中的边缘顶点和面。
* 输入：球体的中心、半径和最近邻居信息。
* 输出：包含顶点和面的信息。

· render\_multiview\_progressive：

* · 功能：基于输入图像和视点进行渐进式多视图渲染。
* 输入：image\_a, image\_b（输入图像），viewpoint\_a, viewpoint\_b（视点），current\_epoch（当前训练轮数），max\_epoch（最大训练轮数）。
* 输出：包含球体中心、半径、顶点和面的信息。

· render\_multiview\_split\_fine：

* · 功能：基于输入图像和视点进行多视图渲染，输出精细模型。
* 输入：image\_a（输入图像），viewpoint\_a（视点）。
* 输出：包含精细模型的顶点和面信息。

· render\_multiview\_edges：

* · 功能：基于输入图像和视点进行多视图渲染，输出边缘模型。
* 输入：image\_a, image\_b（输入图像），viewpoint\_a, viewpoint\_b（视点），aimuth\_a, aimuth\_b（方位）。
* 输出：包含球体模型的边缘信息。

· render\_multiview\_r\_finetune：

* · 功能：基于输入图像和视点对球体模型进行微调。
* 输入：image\_a, image\_b（输入图像），viewpoint\_a, viewpoint\_b（视点），centers, radii（球体的中心和半径）。
* 输出：包含微调后的球体半径和模型顶点、面信息。

· evaluate：

* · 功能：评估输入图像和视点生成的3D模型，输出球体中心、半径和微调后的球体信息。
* 输入：images（输入图像），viewpoints（视点）。
* 输出：包含球体中心、半径和微调后的球体信息。

· evaluate\_overview：

* · 功能：评估输入图像和视点生成的3D模型，输出包含球体中心、半径和微调后的球体信息的概览。
* 输入：images（输入图像），viewpoints（视点）。
* 输出：包含球体中心、半径和微调后的球体信息。

· evaluate\_split：

* · 功能：评估输入图像和视点生成的3D模型，输出精细模型和表面模型的信息。
* 输入：images（输入图像），viewpoints（视点）。
* 输出：包含精细模型和表面模型的顶点、面、球体中心和半径信息。

· evaluate\_render\_edge：

* · 功能：评估并渲染输入图像和视点生成的3D模型的边缘信息。
* 输入：images（输入图像），viewpoints（视点）。
* 输出：包含球体中心、半径、微调后的球体信息以及边缘顶点和面信息。

· evaluate\_joint：

* · 功能：联合评估输入图像和视点生成的3D模型。
* 输入：images（输入图像），viewpoints（视点）。
* 输出：包含球体中心、半径、顶点和面信息。

· evaluate\_surf：

* · 功能：评估并渲染表面模型。
* 输入：image\_a（输入图像），viewpoint\_a（视点）。
* 输出：包含表面模型的顶点、面、球体中心和半径信息。

· transfer\_2squares：

* · 功能：将球体转换为正方体，用于边缘生成。
* 输入：top\_spheres, bottom\_spheres（顶点和底部球体）。
* 输出：包含新生成的顶点和面信息。

· evaluate\_iou：

* · 功能：评估输入图像和体素数据的IOU（交并比）。
* 输入：images（输入图像），voxels（体素数据）。
* 输出：IOU、顶点、面、球体中心和半径信息。

· evaluate\_edge：

* · 功能：评估并渲染输入图像生成的3D模型的边缘信息。
* 输入：images（输入图像），viewpoints（视点）。
* 输出：新生成的顶点和面信息，球体中心和半径。

· predict\_spheres：

* · 功能：预测输入图像生成的球体信息。
* 输入：images（输入图像）。
* 输出：包含顶点、面、球体中心和半径信息。

· predict\_spheres\_render：

* · 功能：预测并渲染输入图像生成的球体信息。
* 输入：images（输入图像），viewpoints（视点）。
* 输出：包含顶点、面、球体中心和半径信息。

· forward：

* · 功能：根据不同任务选择相应的渲染或评估方法。
* 输入：images, viewpoints, azimuths, voxels, centers, radii, task, current\_epoch, max\_epoch, neighbor\_num（任务相关的输入参数）。
* 输出：根据任务类型返回相应的输出结果。

## pointnet\_utils

实现一个PointNet编码器，能够对3D点云数据进行特征提取和变换。这种架构广泛应用于3D物体识别和分类任务中。

定义 STN3d 类

* STN3d 继承了 nn.Module 类，用于构建神经网络模型。
* \_\_init\_\_ 方法初始化了STN3d类的一些层，包括卷积层、全连接层和批量归一化层。
  + self.conv1 是一个一维卷积层，输入通道数为 channel，输出通道数为64，卷积核大小为1。
  + self.conv2 类似，但输入通道数为64，输出为128。
  + self.conv3 输入128，输出1024。
  + self.fc1 是全连接层，输入维度为1024，输出维度为512。
  + self.fc2 输入512，输出256。
  + self.fc3 输入256，输出9（因为我们要生成一个3x3的矩阵，3x3=9）。
  + self.relu 是ReLU激活函数。
  + self.bn1 到 self.bn5 是批量归一化层，用于卷积层和全连接层的输出进行归一化。

### 定义 forward 方法

* forward 方法定义了模型的前向传播过程。
  + 首先获取输入 x 的批次大小 batchsize。
  + 输入通过三个卷积层，每个卷积层后接一个批量归一化层和ReLU激活函数。
  + torch.max(x, 2, keepdim=True)[0] 沿着第二个维度取最大值，这相当于全局最大池化。
  + x.view(-1, 1024) 将 x 变形为大小为 (batchsize, 1024) 的张量。
  + 通过两个全连接层，每个全连接层后接批量归一化层和ReLU激活函数。
  + x = self.fc3(x) 最终通过第三个全连接层，得到大小为 (batchsize, 9) 的输出。
  + 创建一个恒等矩阵 iden，并重复 batchsize 次。
  + 如果 x 在CUDA上运行，将 iden 也放到CUDA上。
  + 将输出 x 和恒等矩阵 iden 相加。
  + 最终将 x 变形为大小为 (batchsize, 3, 3) 的张量，并返回。

### 定义 STNkd 类

* STNkd 类与 STN3d 类类似，但 k 默认为64，fc3 层输出为 k\*k，用于生成 k\*k 的矩阵。

### 定义 PointNetEncoder 类

* PointNetEncoder 类定义了PointNet编码器。
  + self.stn 是 STN3d 类的实例。
  + 初始化了卷积层和批量归一化层。
  + global\_feat 和 feature\_transform 是两个布尔参数，用于控制是否输出全局特征和是否使用特征变换。
  + 如果 feature\_transform 为 True，初始化 STNkd 类的实例。

### 定义 forward 方法

* forward 方法定义了前向传播过程。
  + 获取输入 x 的批次大小 B、维度 D 和点的数量 N。
  + 通过 self.stn 计算转换矩阵 trans。
  + 对 x 进行维度交换。
  + 如果 D 大于3，将特征部分分离出来。
  + 通过 torch.bmm 进行批量矩阵乘法。
  + 如果 D 大于3，将特征部分拼接回去。
  + 对 x 进行维度交换，并通过卷积层和激活函数。
  + 如果 feature\_transform 为 True，通过 self.fstn 计算特征转换矩阵 trans\_feat，并进行批量矩阵乘法。
  + 否则，将 trans\_feat 设置为 None。
  + 获取点的特征 pointfeat。
  + 通过第二个和第三个卷积层，最后进行全局最大池化。
  + 如果 global\_feat 为 True，返回全局特征 x、转换矩阵 trans 和特征转换矩阵 trans\_feat。
  + 否则，将全局特征扩展回去并与点特征拼接，返回结果。

### 7. 定义特征变换正则化函数

该函数用于计算特征变换矩阵的正则化损失。

* 获取转换矩阵 trans 的维度 d。
* 创建单位矩阵 I。
* 如果 trans 在CUDA上运行，将 I 也放到CUDA上。
* 计算转换矩阵与其转置的乘积与单位矩阵的差值的范数，并取平均值作为损失返回。

## Prepare

定义 prepare 函数

这个函数用于准备实验所需的文件夹结构。

* 使用 os.path.join 生成输出目录的路径 directory\_output，路径包含了 args.experiment\_id。
* 使用 os.makedirs 创建这个目录，如果目录已经存在则不报错（exist\_ok=True）。
* 同样的方式创建 image\_output 目录，用于保存图片。
* 将生成的目录路径存入 cfg 字典中，并返回更新后的 cfg。

定义 create\_model 函数

这个函数用于创建模型。

* 从 cfg 中读取模型参数，如 FINE\_NUM, SURF\_NUM, MAX\_RADII, 和 DP\_RATIO。
* 使用这些参数创建模型 model\_joint，调用的是 models\_merge.Model。
* 将模型移动到GPU（调用 .cuda()）。
* 返回创建好的模型。

定义 set\_loss 函数

这个函数用于定义并返回损失函数。

* 创建一个空字典 losses。
* 实例化三个损失函数 kNNRepulsionLoss, kNNLoss, 和 SphereInterLoss，并将它们添加到 losses 字典中。
* 返回包含这些损失函数的字典 losses。

## Render

· 使用Kaolin库生成一个透视投影矩阵，并将其移动到GPU上。

· fovyangle=30 / 180.0 \* np.pi 将30度转换为弧度。

· ratio=1.0 指定纵横比为1。

render\_prepare 函数

准备渲染所需的颜色数据。

* meshcolors\_render 和 meshcolors\_render\_drop 是用numpy创建的全1数组，并转换为PyTorch张量，移动到GPU上。
* 这些张量分别用于渲染所有顶点和被丢弃顶点的颜色。

render\_prepare\_edge 函数

类似于 render\_prepare 函数，但为边缘准备颜色数据

remder\_mask 函数

根据 drop 参数决定渲染全部顶点还是丢弃顶点。

* 使用 rotate\_vertices 旋转顶点。
* 使用 render\_vertex\_colors 渲染顶点颜色。
* 返回渲染的掩码图像。

render\_demo 函数

渲染和保存演示图像和对象文件。

* 生成文件路径 demo\_path。
* 渲染旋转后的顶点并生成掩码图像 render\_f。
* 将球体转换为顶点和面，并保存为对象文件。
* 使用OpenCV保存渲染的掩码图像。

render\_batch\_demo 函数

类似于 render\_demo 函数，但用于批处理渲染。

## resnet

定义 conv3x3 函数

conv3x3 函数定义了一个3x3的卷积层，带有填充（padding），用于保持输入和输出的空间尺寸一致。

定义 BasicBlock 类

· BasicBlock 是ResNet的基本模块，包含两个3x3卷积层。

· expansion 表示通道数的扩展比例，BasicBlock 中为1。

· \_\_init\_\_ 方法初始化卷积层、批量归一化层和ReLU激活函数。

· forward 方法定义前向传播，包含跳跃连接（residual connection）。

定义 Bottleneck 类

· Bottleneck 是ResNet的瓶颈模块，包含1x1、3x3和1x1卷积层。

· expansion 表示通道数的扩展比例，在 Bottleneck 中为4。

· \_\_init\_\_ 和 forward 方法类似于 BasicBlock，但增加了一个1x1卷积层用于减少和恢复通道数。

定义 ResNet 类

· ResNet 类定义了ResNet模型的架构。

· \_\_init\_\_ 方法初始化了模型的各层，包括卷积层、批量归一化层、激活函数、池化层和全连接层。

· \_make\_layer 方法用于创建ResNet的每一层。

· forward 方法定义了前向传播过程。

定义ResNet模型构造函数

· 这些函数用于创建不同版本的ResNet模型。

· pretrained 参数决定是否加载预训练权重。

· \*\*kwargs 用于传递其他参数。

## test

使用 argparse 库解析命令行参数，这些参数将用于控制实验的配置和路径。

· 创建并初始化模型，将其移动到GPU。

· 加载预训练模型的状态字典，并将模型设置为评估模式。

· 加载ShapeNet数据集，使用DataLoader创建数据加载器。

· 创建用于保存结果的目录。

使用Kaolin库生成透视投影矩阵，并将其移动到GPU。

定义 test 函数

· test 函数是核心评估逻辑。

· 定义了一些计时器和损失函数。

· 对每个类别（CLASS\_IDS\_CHAIR中定义）进行循环。

· 为每个类别创建输出目录。

· 对每个批次的数据进行处理。

· 将数据转换为CUDA张量。

· 计算输出结果。

· 保存和渲染模型。

定义 numpy2cuda 函数

将批次数据从numpy数组转换为CUDA张量。

## Train

使用 argparse 库解析命令行参数。

设置随机种子以确保实验的可重复性。

· 使用 prepare 模块中的函数准备配置和创建模型。

· 复制配置文件到输出目录。

· 加载训练数据集并创建数据加载器。

· adjust\_learning\_rate 函数根据不同方法调整学习率。

· set\_lr 函数直接设置学习率。

· numpy2cuda 函数将批量数据从numpy数组转换为CUDA张量。

· adjust\_sigma 函数暂时没有具体实现。

### Train函数

初始化

· SummaryWriter 用于记录训练过程中的数据，供TensorBoard使用。

· time.time() 记录当前时间，用于计算时间间隔。

· AverageMeter 是一个自定义工具类，用于计算和存储指标的平均值和当前值。

· optimizer 使用Adam优化器，并指定学习率。

细粒度训练（train\_fine）

· 检查 cfg['train']['FINETUNE'] 是否为 False，若是则进行细粒度训练。

· 初始化一些计数器和参数。

· render\_prepare 函数准备渲染图像所需的颜色信息。

· 使用 DataLoader 迭代训练数据。

· 调用 numpy2cuda 函数将数据转换为CUDA张量。

· 调整学习率。

· 调用模型的前向传播函数，并指定任务为 train\_fine。

· 渲染图像，并计算损失。

· 更新损失，并进行反向传播和优化。

· 定期保存和打印训练状态。

· 达到指定迭代次数后保存模型。

表面训练（train\_surf）

· 与细粒度训练类似，但训练任务变为 train\_surf。

· 准备渲染图像的颜色信息。

· 使用 DataLoader 迭代训练数据。

· 调用 numpy2cuda 函数将数据转换为CUDA张量。

· 调整学习率。

· 调用模型的前向传播函数，并指定任务为 train\_surf。

· 渲染图像，并计算损失。

· 更新损失，并进行反向传播和优化。

· 定期保存和打印训练状态。

· 达到指定迭代次数后保存模型。

联合训练（train\_w\_fine）

· 联合训练阶段结合了细粒度和表面训练。

· 准备渲染图像的颜色信息。

· 使用 DataLoader 迭代训练数据。

· 调用 numpy2cuda 函数将数据转换为CUDA张量。

· 调整学习率。

· 调用模型的前向传播函数，并指定任务为 train\_w\_fine。

· 渲染图像，并计算损失。

· 更新损失，并进行反向传播和优化。

· 定期保存和打印训练状态。

· 达到指定迭代次数后保存模型。

微调训练（finetune\_r）

· 微调阶段，使用不同的优化器和学习率。

· 准备渲染图像的颜色信息。

· 使用 DataLoader 迭代训练数据。

· 调用 numpy2cuda 函数将数据转换为CUDA张量。

· 调整学习率。

· 调用模型的前向传播函数，并指定任务为 finetune\_r。

· 渲染图像，并计算损失。

· 更新损失，并进行反向传播和优化。

· 定期保存和打印训练状态。

· 达到指定迭代次数后保存模型。

train 函数分为四个主要阶段：细粒度训练、表面训练、联合训练和微调训练。每个阶段都涉及数据准备、模型前向传播、损失计算、反向传播和优化。定期保存和打印训练状态，以便于监控和分析训练过程。

## Utils

加载配置文件

load\_config 函数用于加载YAML配置文件。它读取指定路径的配置文件，并将其内容解析为Python字典。

AverageMeter 类

AverageMeter 类用于计算和存储值的平均值和当前值。它有三个方法：

* \_\_init\_\_：初始化各个参数。
* reset：重置所有参数。
* update：更新当前值、总和、计数和平均值。

Mesh 类

Mesh 类用于存储网格的顶点和面。构造函数将输入的顶点和面从 numpy 数组转换为 torch 张量。

图像转换函数

· img\_cvt 函数将图像张量转换为 uint8 类型的 numpy 数组，并进行格式转换。

· img\_cvt\_merge 函数将两个图像掩码合并。对于渲染掩码中非零的像素，将相应位置的 gt 掩码像素设置为蓝色。

写入和加载函数

· write\_ma 函数将球体数据保存为文本文件。

· write\_cam\_info 函数目前是空的，用于写入相机信息。

· load\_obj\_spherenum 函数从OBJ文件加载网格，并根据指定的球体数量复制顶点和面。

球体转换函数

transfer\_spheres 函数将球体数据转换为网格顶点和面。它根据球体数据调整顶点的位置和大小。

· export\_ma\_off 函数用于将球体和面数据导出为OFF文件格式。

· save\_2d\_png 函数用于保存2D PNG图像。

· load\_template 函数用于加载模板文件，并根据指定数量生成中心和面数据。

· save\_obj 函数用于将顶点和面数据保存为OBJ文件格式

## utils\_functions

### 帮助函数

#### normalize\_meshes\_np

* 这个函数对网格顶点进行归一化操作。它先计算顶点的最大值和最小值，然后将顶点平移到中心，再进行缩放。

#### normalize\_meshes

这个函数使用 torch 对网格顶点进行归一化操作。它先计算每个批次的最大值和最小值，然后将顶点平移到中心，再进行缩放。

### 渲染函数

#### render\_vertex\_colors

· 这个函数用于渲染顶点颜色到图像中。

· vertices\_camera 是相机坐标系中的顶点。

· faces 是网格的面。

· vertex\_colors 是顶点的颜色。

· camera\_proj 是相机投影矩阵。

· height 和 width 是图像的高度和宽度。

· 函数先计算每个面的顶点颜色，然后计算顶点在图像平面上的坐标。

· 使用 dibr\_rasterization\_kaolin 函数进行光栅化，得到渲染结果 imfeat、概率图 improb 和面索引 imfaceidx。

· 返回渲染的图像、概率图和面法线。

### 损失函数

#### calculate\_iou\_loss

· 这个函数用于计算预测掩码和真实掩码之间的损失。

· gt\_mask\_bx1xhxw 是真实掩码。

· pred\_mask\_bx1xhxw 是预测掩码。

· lossname 指定损失类型，可以是 iou 或 l1。

· eps 是一个小值，用于避免除零错误。

· 如果损失类型是 iou，计算交并比（IoU），并返回1减去平均IoU作为损失。

· 如果损失类型是 l1，计算预测掩码和真实掩码之间的L1损失。

## utils\_hole\_fill

### 文件名获取函数 get\_filename

#### 功能和作用

* 读取一个数据集文件，解析并返回源文件和目标文件的名字列表。

#### 输入和输出

* 输入：数据集文件路径（包含源路径和目标路径）。
* 输出：源文件名列表和目标文件名列表。

#### 应用场景

* 适用于需要处理包含文件路径的数据集文件，并提取文件名以进行进一步处理的情况。

### 绘图函数

#### plot\_sample

##### 功能和作用

* 绘制3D散点图和连线图，根据误差值设置连线颜色。

##### 输入和输出

* 输入：
  + spheres：球体的位置数组。
  + edges：边的数组。
  + errors：误差值数组。
  + filename：保存图像的文件名。
  + thred：误差阈值。
* 输出：无直接输出，生成并保存图像文件。

##### 应用场景

* 用于3D散点图和连线图的可视化，特别是当需要根据误差值突出显示某些连线时。

#### plot\_ma

##### 功能和作用

* 绘制3D散点图和连线图。

##### 输入和输出

* 输入：
  + spheres：球体的位置数组。
  + edges：边的数组。
  + filename：保存图像的文件名。
  + rot：是否进行旋转。
* 输出：无直接输出，生成并保存图像文件。

##### 应用场景

* 用于3D散点图和连线图的可视化，不涉及误差值的处理。

#### plot\_sample\_face

##### 功能和作用

* 绘制3D散点图和多边形面图，根据误差值设置面颜色。

##### 输入和输出

* 输入：
  + spheres：球体的位置数组。
  + faces：面的数组。
  + errors：误差值数组。
  + filename：保存图像的文件名。
  + thred：误差阈值。
* 输出：无直接输出，生成并保存图像文件。

##### 应用场景

* 用于3D散点图和多边形面的可视化，特别是当需要根据误差值突出显示某些面时。

### 加载和处理3D数据的函数

#### load\_MAT

##### 功能和作用

* 从文件中加载球体、边和面的数据。

##### 输入和输出

* 输入：MAT文件路径。
* 输出：球体、边和面的数组。

##### 应用场景

* 用于从文件加载3D形状的数据，以便后续处理和可视化。

#### load\_surf\_MAT

##### 功能和作用

* 从文件中加载球体、边和面的数据，并区分细粒度和粗粒度数据。

##### 输入和输出

* 输入：MAT文件路径。
* 输出：球体、边、面以及细粒度的边和面数组。

##### 应用场景

* 用于从文件加载3D形状的数据，并区分细粒度和粗粒度数据，以便进行更细致的处理。

### 保存3D数据的函数

#### write\_ma

##### 功能和作用

* 将球体、边和面的数据保存为MA文件格式。

##### 输入和输出

* 输入：
  + filename\_ma：保存的文件名。
  + spheres：球体的位置数组。
  + edges：边的数组。
  + faces：面的数组。
* 输出：无直接输出，生成并保存MA文件。

##### 应用场景

* 用于将处理后的3D数据保存为MA文件，以便后续使用和分析。

#### write\_ma\_check

##### 功能和作用

* 检查并过滤无效的球体、边和面，然后保存为MA文件。

##### 输入和输出

* 输入：
  + filename\_ma：保存的文件名。
  + spheres：球体的位置数组。
  + edges：边的数组。
  + faces：面的数组。
* 输出：无直接输出，生成并保存MA文件。

##### 应用场景

* 用于在保存3D数据之前进行检查和过滤，确保数据的有效性。