Contents

1

3DGS 原理与实现

代码仓库与版本：54c035f7834b564019656c3e3fcc3646292f727d.

1.1 高斯椭球表示、优化与代码实现

本节代码位于 scene/gaussian_model.py 文件中.

1.1.1 椭球定义

一个高斯球由以下几个核心属性定义：

1. 位置：_xyz. 表示每个高斯球在三维空间中的中心点位置.
2. 协方差：描述高斯球的形状和方向. 为了保证矩阵在优化过程中始终为正定，假设协方差矩阵 Σ 可以由一个缩放矩阵 S 和一个旋转矩阵 R 分解得到，即 $\Sigma = RSS^TR^T$. 在代码中，为了存储方便，其存储的是一个三维向量与一个四元数.
   - 缩放：_scaling. 一个三维向量，即缩放矩阵 S 的对角线元素.
   - 旋转：_rotation. 一个四元数，即旋转矩阵 R 的四元数表示.
3. 颜色：表示高斯球的颜色属性. 使用 Y_{lm} (l 阶 m 次球谐函数) 来表示颜色，这样高斯球的颜色可以随着观察角度的变化而变化，从而模拟更复杂的光照和材质效果. 简单来说，假设视角是 (θ, φ)，那么颜色可以表示为球面 $\mathbb{S}^2$ 上的函数 $C(\theta, \varphi)$，其可以展开为球谐函数的线性组合：

$$C(\theta,\varphi) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} c_{lm} Y_{lm}(\theta,\varphi).$$

在实际应用中，通常只考虑低阶的球谐函数 (例如 $l = 0, 1, 2$)，以减少计算复杂度. 在 3DGS 代码中取的是最高 3 阶. 对于 $0, 1, \ldots, l$ 阶的球谐函数，共有 $1 + 3 + \cdots + (2l + 1) = (l + 1)^2$ 个球谐系数 c_{lm} 需要优化. 在代码中，这些系数被分开存储在两个属性中.

- 漫反射系数：_feature_dc. 也即球谐函数的零阶系数 c_{00}，这代表了高斯球的基础颜色，不随视角变化. 这是因为 $Y_{00} = 1/\sqrt{4\pi}$ 是常数.
- 高阶系数：_feature_rest. 也即剩下的从 1 阶到 l 阶的球谐系数.

所以，对于一个颜色通道，若最高阶数为 l，则总共有 $(l+1)^2$ 个球谐系数. 对于 RGB 三个颜色通道，总共有 $3(l+1)^2$ 个系数需要存储.

4. 不透明度：`_opacity`. 表示高斯球的透明度属性.

总的来说，对于一个高斯球，需要存储 $3+3+4+3(l+1)^2+1=11+3(l+1)^2$ 个参数. 若 $l=3$，也即最高计算 3 阶球谐系数，则总共有 $11+3\times 16=59$ 个参数.

由于上述参数有些需要一定的数学约束，例如四元数的四个参数必须满足单位长度约束，缩放参数必须为正等等，因此在代码实现中，3DGS 使用无约束的参数进行优化，然后通过一些变换将其映射到有约束的参数空间中. 例如，四元数参数通过归一化来确保其单位长度，缩放参数通过指数映射来确保其为正值等等. 具体来说，代码中定义了以下参数变换函数：

```
def setup_functions(self):
    def build_covariance_from_scaling_rotation(scaling, scaling_modifier, rotation):
        L = build_scaling_rotation(scaling_modifier * scaling, rotation)
        actual_covariance = L @ L.transpose(1, 2)
        symm = strip_symmetric(actual_covariance)
        return symm

    self.scaling_activation = torch.exp
    self.scaling_inverse_activation = torch.log
    self.covariance_activation = build_covariance_from_scaling_rotation
    self.opacity_activation = torch.sigmoid
    self.inverse_opacity_activation = inverse_sigmoid
    self.rotation_activation = torch.nn.functional.normalize
```

可以看到，`scaling_activation` 定义为指数函数，用于确保缩放参数属性后为正值；`covariance_activation` 定义为通过缩放和旋转构建协方差矩阵的函数 (即 $\Sigma = RSS^TR^T$)；`opacity_activation` 定义为 Sigmoid 函数，确保不透明度在 $[0,1]$ 范围内；`rotation_activation` 定义为归一化函数，确保四元数属性变换后为单位四元数. 在返回实际属性时，3DGS 会调用这些激活函数将无约束参数映射到有约束的属性空间中，例如：

```
@property
def get_scaling(self):
    return self.scaling_activation(self._scaling)
```

1.1.2 场景创建与 I/O

首先从一个初始点云创建高斯模型，这是训练开始的第一步，具体代码实现如下：

```
def create_from_pcd(self, pcd : BasicPointCloud, cam_infos : int, spatial_lr_scale :
↪  float):
    self.spatial_lr_scale = spatial_lr_scale
    fused_point_cloud = torch.tensor(np.asarray(pcd.points)).float().cuda()
```

```
    fused_color = RGB2SH(torch.tensor(np.asarray(pcd.colors)).float().cuda())
    features = torch.zeros((fused_color.shape[0], 3, (self.max_sh_degree + 1) **
    ↪  2)).float().cuda()
    features[:, :3, 0 ] = fused_color
    features[:, 3:, 1:] = 0.0

    print("Number of points at initialisation : ", fused_point_cloud.shape[0])

    dist2 =
    ↪  torch.clamp_min(distCUDA2(torch.from_numpy(np.asarray(pcd.points)).float().cuda()),
    ↪  0.0000001)
    scales = torch.log(torch.sqrt(dist2))[...,None].repeat(1, 3)
    rots = torch.zeros((fused_point_cloud.shape[0], 4), device="cuda")
    rots[:, 0] = 1

    opacities = self.inverse_opacity_activation(0.1 *
    ↪  torch.ones((fused_point_cloud.shape[0], 1), dtype=torch.float,
    ↪  device="cuda"))

    self._xyz = nn.Parameter(fused_point_cloud.requires_grad_(True))
    self._features_dc = nn.Parameter(features[:,:,0:1].transpose(1,
    ↪  2).contiguous().requires_grad_(True))
    self._features_rest = nn.Parameter(features[:,:,1:].transpose(1,
    ↪  2).contiguous().requires_grad_(True))
    self._scaling = nn.Parameter(scales.requires_grad_(True))
    self._rotation = nn.Parameter(rots.requires_grad_(True))
    self._opacity = nn.Parameter(opacities.requires_grad_(True))
    self.max_radii2D = torch.zeros((self.get_xyz.shape[0]), device="cuda")
    self.exposure_mapping = {cam_info.image_name: idx for idx, cam_info in
    ↪  enumerate(cam_infos)}
    self.pretrained_exposures = None
    exposure = torch.eye(3, 4, device="cuda")[None].repeat(len(cam_infos), 1, 1)
    self._exposure = nn.Parameter(exposure.requires_grad_(True))
```

首先将点云的坐标和颜色转换成 `Tensor` 类型，初始的零阶球谐系数就是对应的 RGB 颜色，高阶球谐系数全为零. 然后通过计算每个点到其最近邻点的距离 (`distCUDA2()` 函数) 来估算一个初始大小，避免高斯球过大或过小. 初始化旋转为单位四元数. 初始化不透明度为一个较小的值 (0.1)，以确保初始模型较为透明. 最后将这些初始参数封装为 `nn.Parameter` 对象，以便在训练过程中进行优化.

高斯模型可以输出为 `.ply` 格式的点云文件，里面包含高斯球的所有属性，代码如下：

```
def save_ply(self, path):
    mkdir_p(os.path.dirname(path))
```

```
    xyz = self._xyz.detach().cpu().numpy()
    normals = np.zeros_like(xyz)
    f_dc = self._features_dc.detach().transpose(1,
    ↪  2).flatten(start_dim=1).contiguous().cpu().numpy()
    f_rest = self._features_rest.detach().transpose(1,
    ↪  2).flatten(start_dim=1).contiguous().cpu().numpy()
    opacities = self._opacity.detach().cpu().numpy()
    scale = self._scaling.detach().cpu().numpy()
    rotation = self._rotation.detach().cpu().numpy()

    dtype_full = [(attribute, 'f4') for attribute in
    ↪  self.construct_list_of_attributes()]

    elements = np.empty(xyz.shape[0], dtype=dtype_full)
    attributes = np.concatenate((xyz, normals, f_dc, f_rest, opacities, scale,
    ↪  rotation), axis=1)
    elements[:] = list(map(tuple, attributes))
    el = PlyElement.describe(elements, 'vertex')
    PlyData([el]).write(path)
```

这个函数将高斯球的各个属性提取出来，转换为 NumPy 数组，然后按照 PLY 文件格式的要求进行组织和存储．每个点的属性标签由 `construct_list_of_attributes()` 函数生成，首先是六个位置属性 `'x','y','z','nx','ny','nz'`，然后是颜色属性 `'f_dc_i','f_rest_i'`，最后是透明度和形状属性 `'opacity','scale_i','rot_i'`．

当然，也可以从上述格式的 PLY 文件中加载高斯模型，代码位于 `load_ply()` 函数中，这里不再赘述．

1.1.3 训练与优化

接下来我们介绍高斯模型的训练与优化过程，代码位于 `train.py` 文件中．代码的主要流程都在 `training()` 函数中，我们逐步进行解读．首先初始化各项设置：

```
first_iter = 0
tb_writer = prepare_output_and_logger(dataset)
gaussians = GaussianModel(dataset.sh_degree, opt.optimizer_type)
scene = Scene(dataset, gaussians)
gaussians.training_setup(opt)
```

其中 `scene` 对象维护着相机和高斯模型，并提供渲染接口．

1.2 Splatting 渲染原理与实现

1.2.1 前向传播渲染