Tracing the Data Flow of View-Dependent Color Representation

- `scene/gaussian_model.py`
  - hash grid(originally from instant-ngp)
    - use two tcnn components
      - $c_n(d) = f(contract(p_n),d;\theta)$, $\theta$ probably stands for the mlp head parameters
      1. recolor (hash grid) encoding
         - input: xyz (3D coordinates)
         - output: a feature vector that contains spatial information
      2. direction encoding
    - tcnn: Tiny CUDA Neural Networks
    - max SH degree is set to 0

```
# __init__()
self.recolor = tcnn.Encoding(
             n_input_dims=3,
             encoding_config={
                "otype": "HashGrid",
                "n_levels": 16,
                "n_features_per_level": 2,
                "log2_hashmap_size": model.max_hashmap,
                "base_resolution": 16,
                "per_level_scale": 1.447,
            },
    )
self.direction_encoding = tcnn.Encoding(
    n_input_dims=3,
    encoding_config={
        "otype": "SphericalHarmonics",
        "degree": 3
    },
    )
self.mlp_head = tcnn.Network(
        n_input_dims=
(self.direction_encoding.n_output_dims+self.recolor.n_output_dims),
        n_output_dims=3,
        network_config={
            "otype": "FullyFusedMLP",
            "activation": "ReLU",
            "output_activation": "None",
            "n_neurons": 64,
            "n_hidden_layers": 2,
        },
    )
```

- initialization:

```
# precompute()
def precompute(self):
    xyz = self.contract_to_unisphere(self.get_xyz.half(), 
torch.tensor([-1.0, -1.0, -1.0, 1.0, 1.0, 1.0], device='cuda'))
    self._feature = self.recolor(xyz)
    torch.cuda.empty_cache()

# contract_to_unisphere()
def contract_to_unisphere(self,
    x: torch.Tensor,
    aabb: torch.Tensor,
    ord: int = 2,
    eps: float = 1e-6,
    derivative: bool = False,
):
    aabb_min, aabb_max = torch.split(aabb, 3, dim=-1)
    x = (x - aabb_min) / (aabb_max - aabb_min)
    x = x * 2 - 1  # aabb is at [-1, 1]
    mag = torch.linalg.norm(x, ord=ord, dim=-1, keepdim=True)
    mask = mag.squeeze(-1) > 1

    if derivative:
        dev = (2 * mag - 1) / mag**2 + 2 * x**2 * (
            1 / mag**3 - (2 * mag - 1) / mag**4
        )
        dev[~mask] = 1.0
        dev = torch.clamp(dev, min=eps)
        return dev
    else:
        x[mask] = (2 - 1 / mag[mask]) * (x[mask] / mag[mask])
        x = x / 4 + 0.5  # [-inf, inf] is at [0, 1]
```

- `gaussian_renderer/__init__.py`

```
# render()
xyz = pc.contract_to_unisphere(means3D.clone().detach(), torch.tensor([-1.0, -1.0, 
-1.0, 1.0, 1.0, 1.0], device='cuda'))
dir_pp = (means3D - viewpoint_camera.camera_center.repeat(means3D.shape[0], 1))
dir_pp = dir_pp/dir_pp.norm(dim=1, keepdim=True)
shs = pc.mlp_head(torch.cat([pc.recolor(xyz), pc.direction_encoding(dir_pp)], 
dim=-1)).unsqueeze(1)
```

- xyz = pc.contract_to_unisphere(): Contract the unbounded positions $p \in R^{N \times 3}$ to the bounded range
- dir_pp: Compute the 3D view direction $d \in R^3$ for each gaussian based on the camera center point