'params'

```
# scene/gaussian_model.py
def training_setup(self, training_args):
        self.percent_dense = training_args.percent_dense
        self.xyz_gradient_accum = torch.zeros((self.get_xyz.shape[0], 1),
device="cuda")
        self.denom = torch.zeros((self.get_xyz.shape[0], 1), device="cuda")

        l = [
            {'params': [self._xyz], 'lr': training_args.position_lr_init *
self.spatial_lr_scale, "name": "xyz"},
            {'params': [self._features_dc], 'lr': training_args.feature_lr,
"name": "f_dc"},
            {'params': [self._features_rest], 'lr': training_args.feature_lr /
20.0, "name": "f_rest"},
            {'params': [self._opacity], 'lr': training_args.opacity_lr, "name":
"opacity"},
            {'params': [self._scaling], 'lr': training_args.scaling_lr, "name":
"scaling"},
            {'params': [self._rotation], 'lr': training_args.rotation_lr, "name":
"rotation"}
        ]

        self.optimizer = torch.optim.Adam(l, lr=0.0, eps=1e-15)
        self.xyz_scheduler_args =
get_expon_lr_func(lr_init=training_args.position_lr_init*self.spatial_lr_scale,

lr_final=training_args.position_lr_final*self.spatial_lr_scale,

lr_delay_mult=training_args.position_lr_delay_mult,

max_steps=training_args.position_lr_max_steps)
```

- l: a list that use to store the information of optimizable parameter, one optimizable parameter will store in one dictionary
- 'params': key of the dictionary of one optimizable parameter. Will be initialized with the pravate attributes of `GaussianModel`, including:
    - _xyz: the 3D position / coordinates of 3D Gaussians
    - _features_dc: parameters of Spherical Harmonics (SH) value, direct component of SH value
    - _features_rest: parameters of Spherical Harmonics (SH) value, rest components of SH value
    - _opacity: opacity of 3D Gaussians
    - _scaling: scale of the Gaussian splatting (used to calculate the 2D projection on the camera plane of 3D Gassian)
    - _rotation: orientation of the Gaussian splatting (used to calculate the 2D projection on the camera plane of 3D Gassian)
- these parameters will be initialized in `create_from_pcd(self, pcd : BasicPointCloud, spatial_lr_scale : float)` in `scene/gaussian_model.py`

Understanding Spherical Harmonic

- like fourier series, use mix of function to express
- use spherical coordinate system:
    - Radial Distance (r): distance from the origin (center) of the coordinate system to the point in space
    - Polar Angle (θ): angle between the radial line (from the origin to the point) and the positive z-axis
    - Azimuthal Angle (ϕ): angle between the projection of the radial line onto the xy-plane (measured from the positive x-axis) and the positive x-axis
- rgb to SH:
    - red, green and blue need to be represent seperately: e.g., use 3 orders of SH (1+3+5=9 coefficients) to represents rgb, totally $9 \times 3 = 27$ coefficients.
- refer to (球谐函数介绍（Spherical Harmonics）)[https://puye.blog/posts/SH-Introduction-CN/]
- refer to (Table of spherical harmonics)[https://en.wikipedia.org/wiki/Table_of_spherical_harmonics]

Questions about Spherical Harmonic

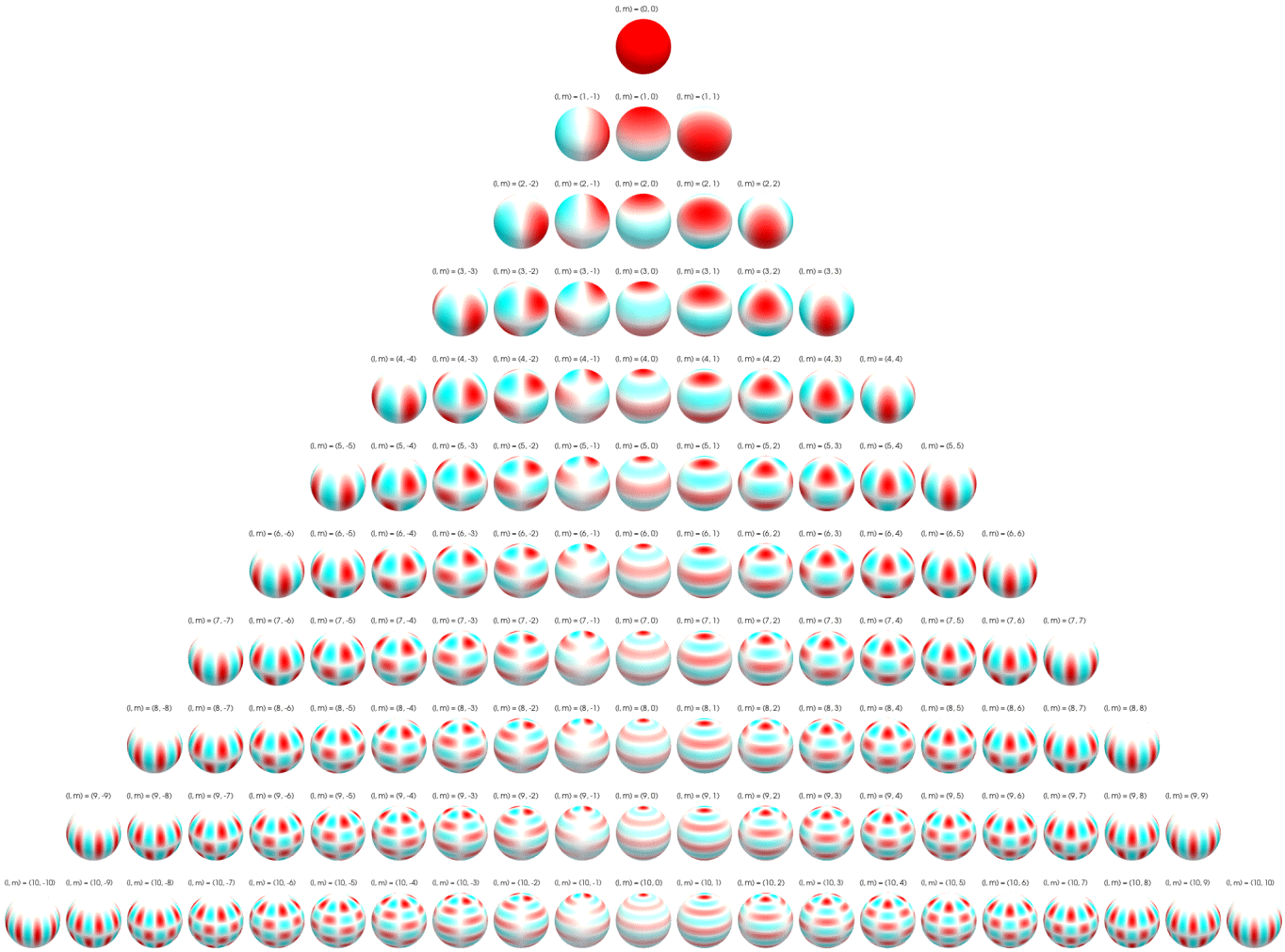
1. Why need to split direct component (features_dc) and rest component (features_rest) in training setup?
    - Is it because the 1st order / degree has significant difference between the rest of the order / degree?
2. In `utils\sh_utils.py`, the coefficients of SH from 1st order to 5th order (0 to 4 degree, but the code only support 0 - 3 degree currently) is defined. According to the Table of spherical harmonics, the 2nd order SH ($\mathbf{C_1}$) should contains 3 coefficients, but the code only has one, why is that?

```
# utils\sh_utils.py
C0 = 0.28209479177387814

# only one coefficient
C1 = 0.4886025119029199

C2 = [
    1.0925484305920792,
    -1.0925484305920792,
    0.31539156525252005,
    -1.0925484305920792,
    0.5462742152960396
]
C3 = [
    -0.5900435899266435,
    2.890611442640554,
    -0.4570457994644658,
    0.3731763325901154,
    -0.4570457994644658,
    1.445305721320277,
    -0.5900435899266435
]
C4 = [
    2.5033429417967046,
    -1.7701307697799304,
```

```
    0.9461746957575601,
    -0.6690465435572892,
    0.10578554691520431,
    -0.6690465435572892,
    0.47308734787878004,
    -1.7701307697799304,
    0.6258357354491761,
]
```



network_gui

```
# train.py
if network_gui.conn == None:
            network_gui.try_connect()
        while network_gui.conn != None:
            try:
                net_image_bytes = None
                custom_cam, do_training, pipe.convert_SHs_python,
pipe.compute_cov3D_python, keep_alive, scaling_modifer = network_gui.receive()
                if custom_cam != None:
                    net_image = render(custom_cam, gaussians, pipe, background,
scaling_modifer)["render"]
                    net_image_bytes = memoryview((torch.clamp(net_image, min=0,
```

```
max=1.0) * 255).byte().permute(1, 2, 0).contiguous().cpu().numpy())
                network_gui.send(net_image_bytes, dataset.source_path)
                if do_training and ((iteration < int(opt.iterations)) or not
keep_alive):
                    break
            except Exception as e:
                network_gui.conn = None
```

- network_gui: network graphical user interface, defined by `gaussian_renderer/network_gui.py`
    - used to show the rendered scene

xyz_scheduler_args

```
# scene/gaussian_model.py
self.xyz_scheduler_args =
get_expon_lr_func(lr_init=training_args.position_lr_init*self.spatial_lr_scale,

lr_final=training_args.position_lr_final*self.spatial_lr_scale,

lr_delay_mult=training_args.position_lr_delay_mult,

max_steps=training_args.position_lr_max_steps)
```

- xyz_scheduler_args: As shown above, xyz (3D position of 3D Gaussians) is an optimizable parameter, the learning rate of xyz is dynamically changing during the training process
    - self.xyz_scheduler_args is a fuction that generate the corresponding learning rate (lr) give certain iteration

My guess on resolution_scale / resolution_scales

```
# scene/__init__.py
for resolution_scale in resolution_scales:
            print("Loading Training Cameras")
            self.train_cameras[resolution_scale] =
cameraList_from_camInfos(scene_info.train_cameras, resolution_scale, args)
            print("Loading Test Cameras")
            self.test_cameras[resolution_scale] =
cameraList_from_camInfos(scene_info.test_cameras, resolution_scale, args)
```

- for the entire training process, default setup `resolution_scale=1.0` is used
- `resolution_scale` will affect the resolution of ground truth image (see `utils/camera_utils.py`):

```
# utils/camera_utils.py
def loadCam(args, id, cam_info, resolution_scale):
```

```
    orig_w, orig_h = cam_info.image.size

    if args.resolution in [1, 2, 4, 8]:
        resolution = round(orig_w/(resolution_scale * args.resolution)),
    round(orig_h/(resolution_scale * args.resolution))
    else:  # should be a type that converts to float
        if args.resolution == -1:
            if orig_w > 1600:
                global WARNED
                if not WARNED:
                    print("[ INFO ] Encountered quite large input images (>1.6K
    pixels width), rescaling to 1.6K.\n "
                          "If this is not desired, please explicitly specify '--
    resolution/-r' as 1")
                    WARNED = True
                global_down = orig_w / 1600
            else:
                global_down = 1
        else:
            global_down = orig_w / args.resolution

        scale = float(global_down) * float(resolution_scale)
        resolution = (int(orig_w / scale), int(orig_h / scale))

    resized_image_rgb = PILtoTorch(cam_info.image, resolution)

    gt_image = resized_image_rgb[:3, ...]
    loaded_mask = None

    if resized_image_rgb.shape[1] == 4:
        loaded_mask = resized_image_rgb[3:4, ...]

    return Camera(colmap_id=cam_info.uid, R=cam_info.R, T=cam_info.T,
                  FoVx=cam_info.FovX, FoVy=cam_info.FovY,
                  image=gt_image, gt_alpha_mask=loaded_mask,
                  image_name=cam_info.image_name, uid=id,
    data_device=args.data_device)
```

- more exactly, the gt image will be downsampled to certain size according to `resolution_scale`
- so other values of `resolution_scale` (besides `resolution_scale=1.0`) **might be** used when the gt images are too large to compute and