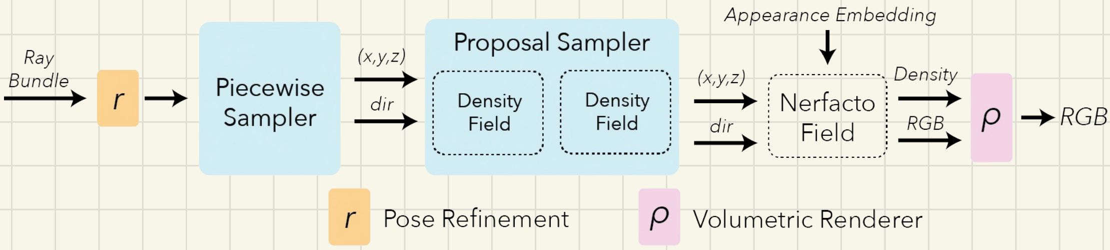


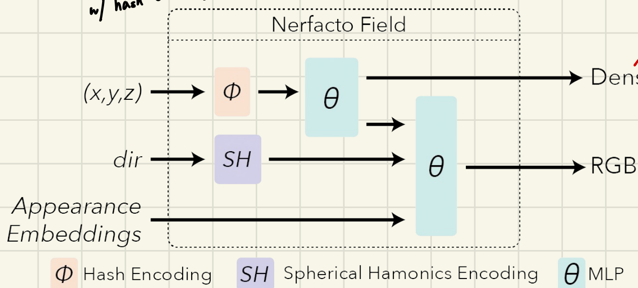
1) Pose Refinement – backpropagate loss gradients to the input camera pose calculations

2) Piecewise Sampler – half of samples are uniformly spaced out while other half are sampled at varying step sizes

3) Proposal Sampler – use density function to consolidate samples to regions of space that contribute the most to the render

fully-used MLP w/ hash encodings

4) Nerfacto Field



→ the sigma values we are interested in!

Code Structure (Single GPU core → gpu-20)

Training

train.py
- entrypoint()
  ↳ main()
    ↳ launch()
      ↳ train_loop()

trainer.py
- setup(): extract images, set up pipeline, optimizers, checkpoints, load configurations file
  ↳ train() → initialize viewer state
    ↳ train_iteration(): at given step, extract loss, loss_dict, and metric_dict
      ↳ update viewer state
      ↳ save checkpoint, close Event Logger / print out final results

In-Depth Look at train_iteration()

↳ called at every step of the training process

- call zero_grad() on optimizers for camera & proposal networks
- get_train_loss_batch() → base_pipeline.py
  - ↳ extract next batch of data from train dataloader
    - base_datamanager.py → extract 4096 rays in the batch using pixel sampling
  - ↳ pass bundle of rays through Nerfacto model (get_outputs() in nerfacto.py)
    - calls get_density() and get_outputs() from TCNNNerfactoField in nerfacto_field.py
      - ↳ computes densities w/ some activation after passing through an MLP
      - computes RGB values after passing SH encoding, density embedding, & appearance embedding through MLP
    - compute weights using densities from raysamples (get_weights() in rays.py)
      - ↳ extract alpha values as $1 - \text{torch.exp}(-\sigma \cdot d)$ → we can model alpha directly using softplus here
      - ↳ extract transmittance values using $(\sigma \cdot d)$ → eventually passed through torch.exp( )
    - performs accumulation (of depths, RGBs, and opacities)
      - renderers.py & vol_rendering.py → used to accumulate data for each ray into a singular rendered view
      - calls nerfacc.accumulate_along_rays() or simply torch.sum() to perform the accumulation!

- reduces loss_dict into single accumulated loss value
- scale gradient backwards (for precision purposes)
- optimizer.step()
- update gradient scaler
- update learning rate scheduler