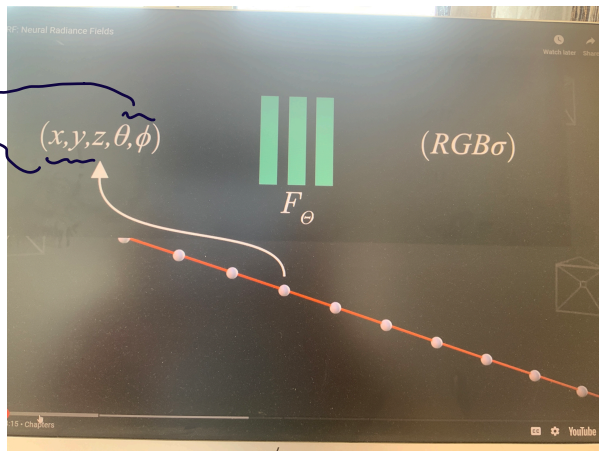
NeRF: neural radiance fields

Set of input images into a scene
e.g. Lego Bulldozer

→ optimize a volumetric representation of the scene as a vector valued function

Defined for any continuous 5D coordinate



parameterize scene representation as a fully connected deep network that takes each 5D coord. and outputs the:

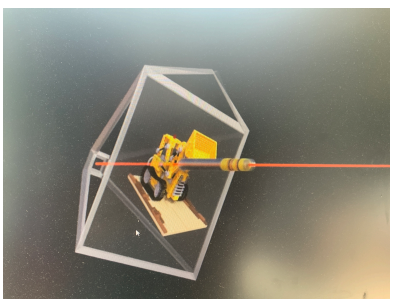
- volume density
- view dependent RGB radiance

Then volume rendering techiques can be used:

- composite these ouput values along a camera ray to render any pixel

This rendering is fully differentiable

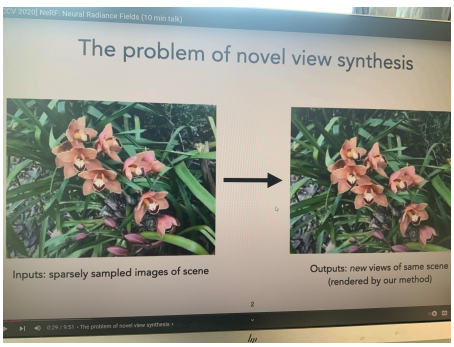
⇒ Scene representation can be optimized by minimizing the error of rendering all camera rays for a collection of standard rgb images



Notes from Technical Talk:

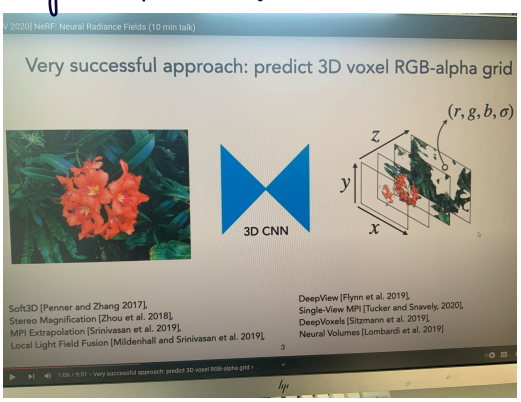
- Focus: View Synthesis

Multiple Input Imgs, along w/ cam. poses
↳ render
photo realistic new views of the scene



Very successful approach:
set of input images ⊕ procedure: e.g. 3D CNN
- to predict a discrete volumetric representation like a voxel grid

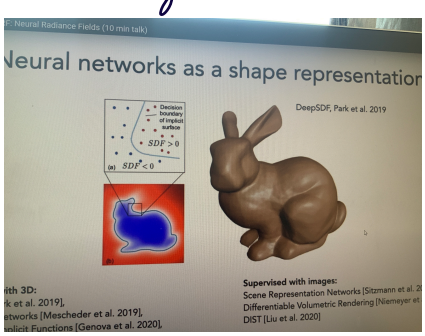
↳ then render new views by simply compositing along rays
⊕ diffable
⊕ optimizable



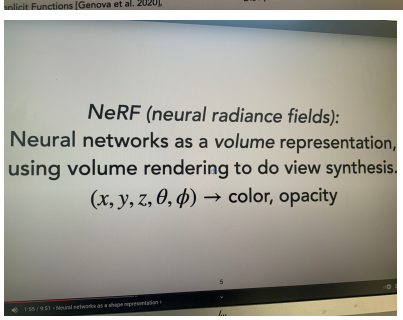
Instead of discretized grid, represent shapes as continuous functions

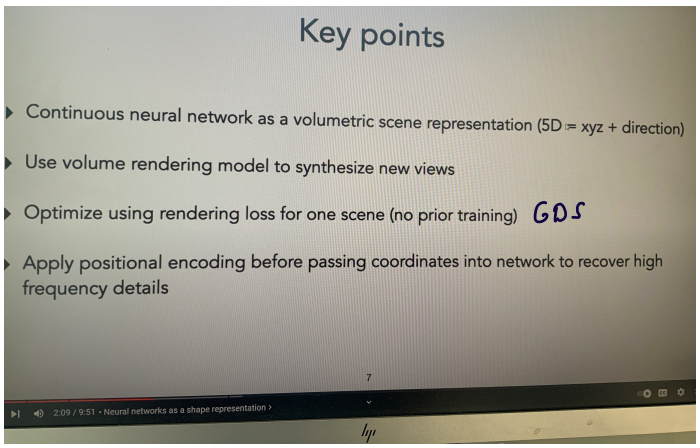
Main Idea: Represent a shape surface as the level set of a fully connected neural network on 3D space

since the entire representation is just a network weights the compression benefit can be huge vs. voxel grids



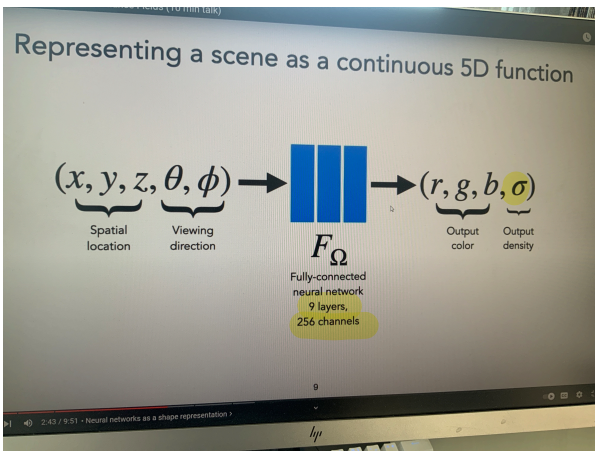
In Nerf:
- avoid limitations of discrete grids
- instead use a neural network to encode a cont. and volumetric representation of a scene



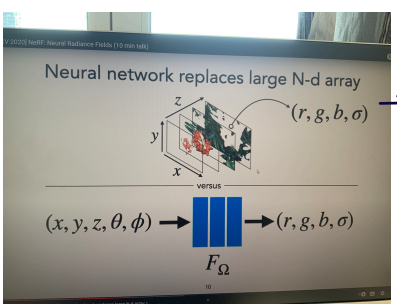


GDS

(1)



The network outputs the volume density sigma particles at that location
+ RGB : radiance emitted at $p(x,y,z)$ and along that input direction $\vec{v}(x,y,z)$

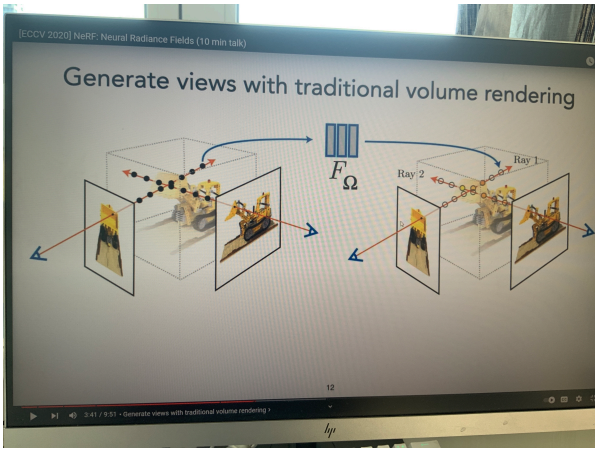


→ voxel grid representation

→ fitting a cont. function approximator to the volume w/o instantiating a grid of individual samples
⇒ allows to add more input dimensions w/o incurring any extra storage

Imagine: save 5D func. as a grid sample array
discritizing extra additional dimensions are expensive
→ In Nerf : almost free by concatinating more nums to the network input

(2)



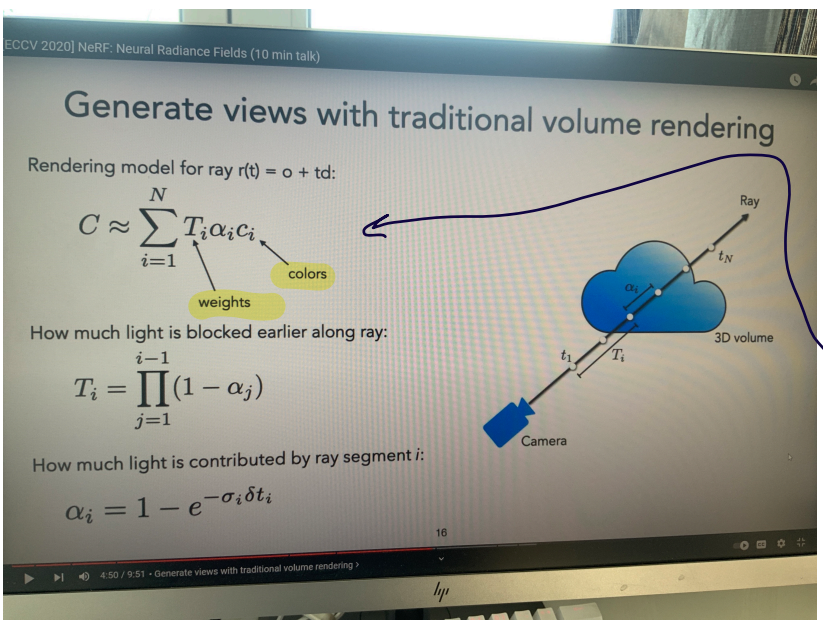
To render:
- we have to estimate how much light of each color makes it out of the volume along each cam. ray

To render each view:
Query the network at a bunch of discrete points along the ray (black dots on the left)

Network helps us corresponding view dependent colors and volume densities (on right)

→ simply composite these values along each ray to compute the final signal output

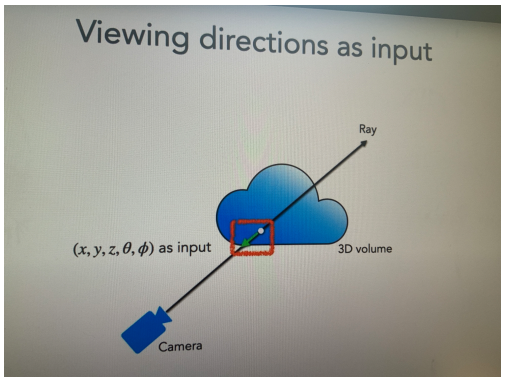
(Closer look)



estimate 1D integral along ray
This is done by querying the MLP at a bunch of samples between starting and ending distances (t_1, t_n)
- Use quadrate estimate, to estimate integral val.
- sum of contribution from each segment of the ray

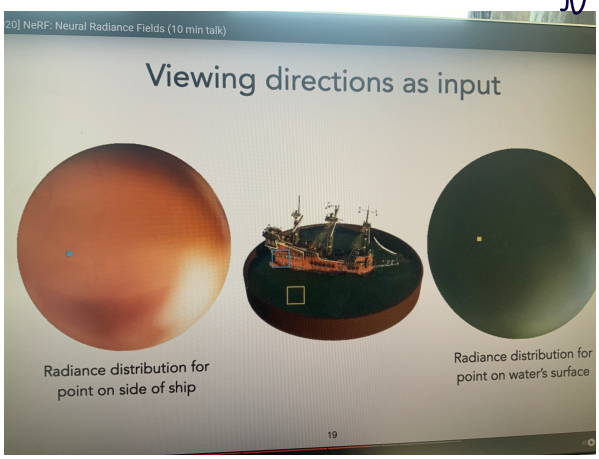
T_i : estimate of transmission : how much light is blocked before segment

α_i : amount of light emitted by the segment
- func. of segment length estimated volume density σ (sigma)
& same as opacity of the segment



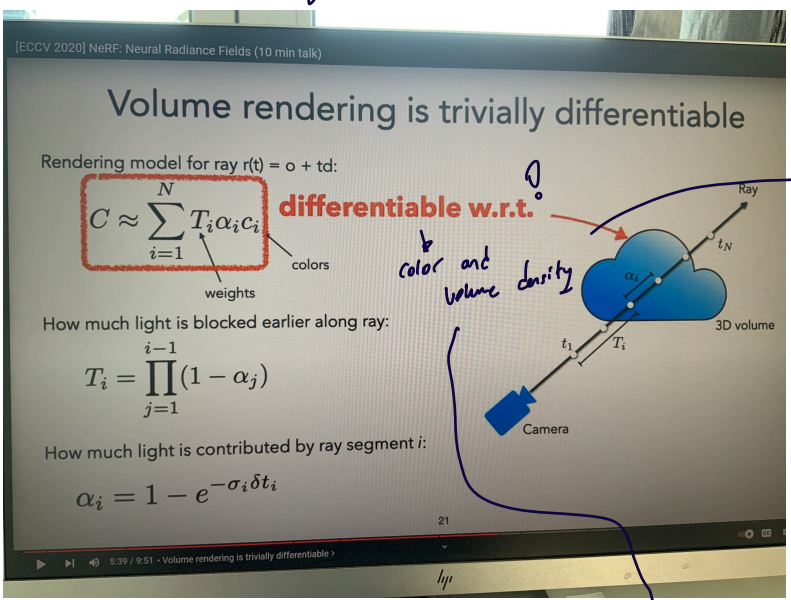
NeRF allows of color of any 3D point to vary as a function of viewing direction, as well as 3D pos.

Changing view dir. (θ, ϕ), view dependent effects can be visualized by network

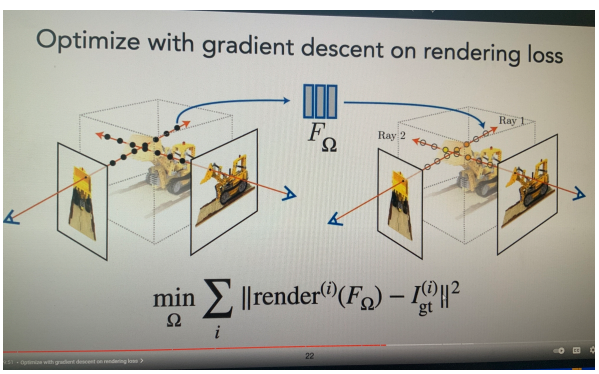


Specular highlight increases at grazing angles ✓

(3) Optimize using rendering loss for one scene (no prior training)



⇒ diffable. w/ respect to the parameters of the network that is outputting these vals.



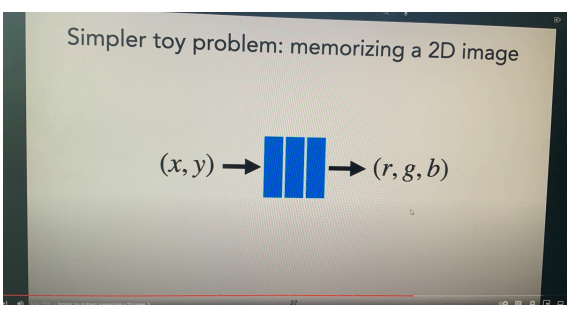
This lets us optimize the network params by using stochastic gradient descent to minimize the error rerendering all input imgs.

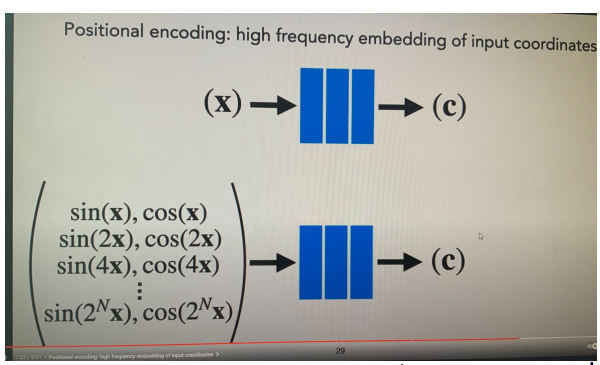
Over the training, simple multi view consistency encourages the network allocate high volume density and accurate colors to the locations where the surfaces actually exist in the underlying scene

(4)

Problem: How to get MLP to represent high frequency functions?

Example:





Fix: Map each input to higher dim. space encoding w/ set of sinusoids that exponentially increasing frequency

→ Apply this transformation to each scalar coord. before passing into the network

Note: Completly deterministic mapping, w/ no learn params. and incurs negligable extra storage to compute cost