KAIST

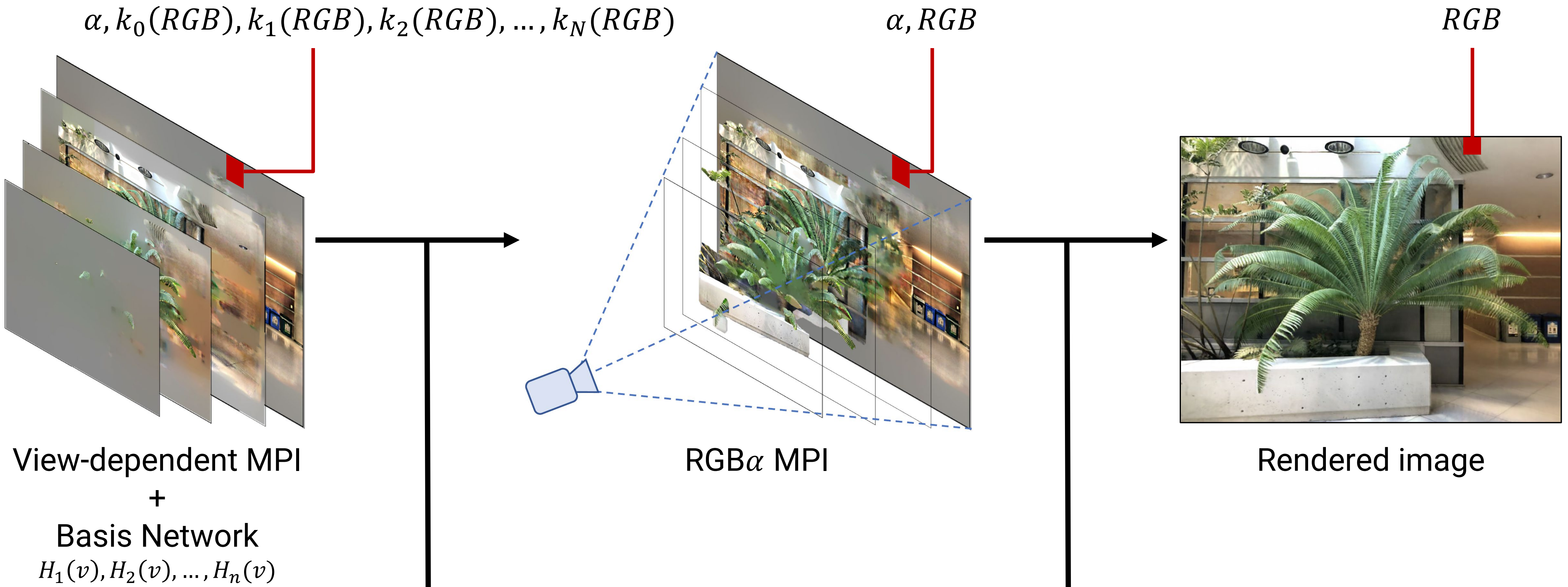
Norm-NeX

Team 1

20214609 Jaemin Cho

20223664 Dongyoung Choi

Original NeX

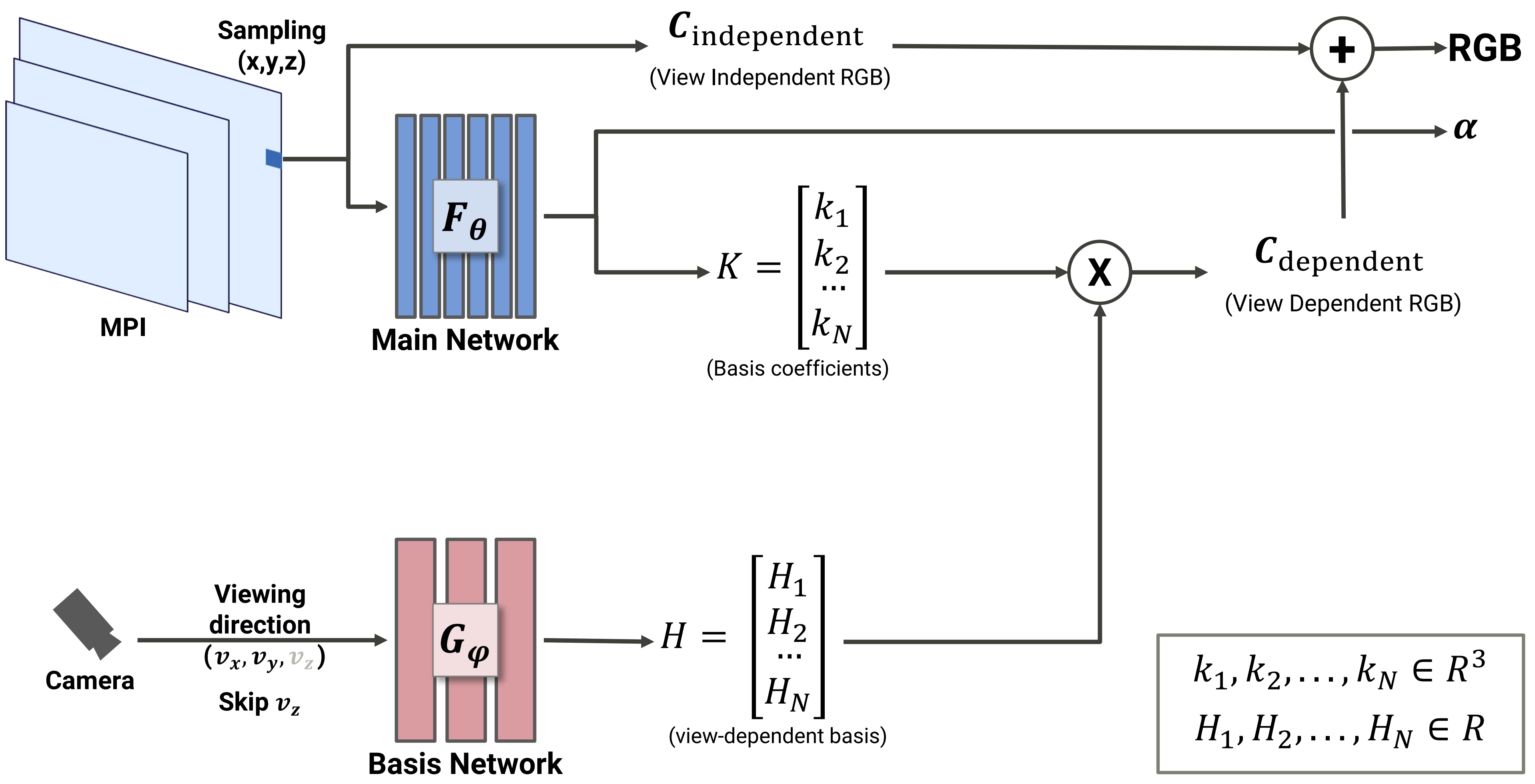


$\alpha, k_0(RGB), k_1(RGB), k_2(RGB), \ldots, k_N(RGB)$
α, RGB
RGB
View-dependent MPI
+
Basis Network
$H_1(v), H_2(v), \ldots, H_n(v)$
RGBα MPI
Rendered image

$$C^P = k_0 + \sum_{n=1}^{N} k_n^P \times H_n(v)$$

$$C = \sum_{d=1}^{D} w_d \times C^P \quad \left[(cf)\ w_d = \alpha_d \prod_{i=1}^{d-1} (1 - \alpha_i)\right]$$

Original NeX



$$k_1, k_2, \ldots, k_N \in R^3$$
$$H_1, H_2, \ldots, H_N \in R$$

Original NeX

- The rendering quality is moderately good, but not perfect and components have no meaning

Ground truth

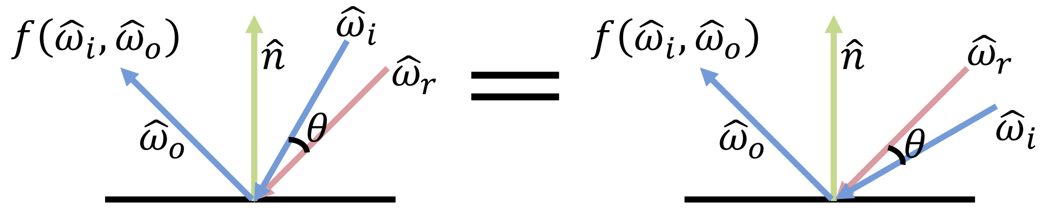
Rendered image

Ideas to improve on weaknesses

- Reflection of viewing direction
- Reparameterization to the meaningful components(normal, diffuse, specular, etc)
- Compute normal vectors
- RGB ↔ YCbCr

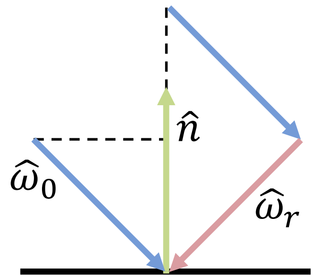
Ref-NeRF: reflection of viewing direction

- BRDF : rotationally-symmetric about reflected view direction
  - $f(\hat{\omega}_i, \hat{\omega}_o) = p(\hat{\omega}_r \cdot \hat{\omega}_i)\ for\ some\ function\ p$
  - Neglecting interreflections and self-occlusions



$\hat{\omega}_o$	: viewing direction
$\hat{\omega}_r$	: reflection of viewing direction
$\hat{\omega}_i$	: input radiance
$f(\hat{\omega}_i, \hat{\omega}_o)$	: output radiance to viewing direction

- We can calculate reflection of viewing direction through viewing direction and normal vector
  - $\hat{\omega}_r = 2(\hat{\omega}_0 \cdot \hat{n})\hat{n} - \hat{\omega}_0$
  - We need accurate normal vector!

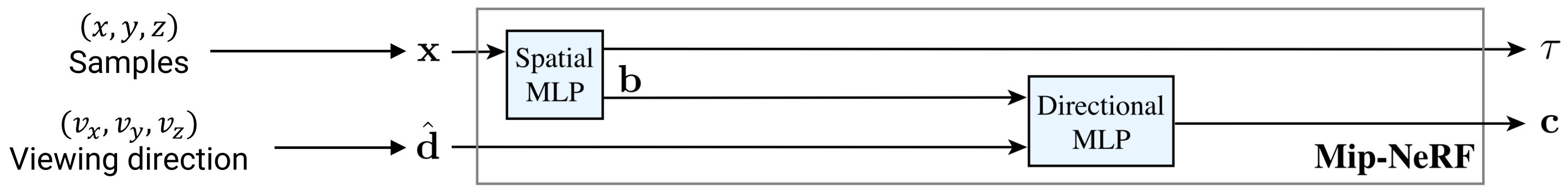


Ref-NeRF: reparameterization for meaningful components

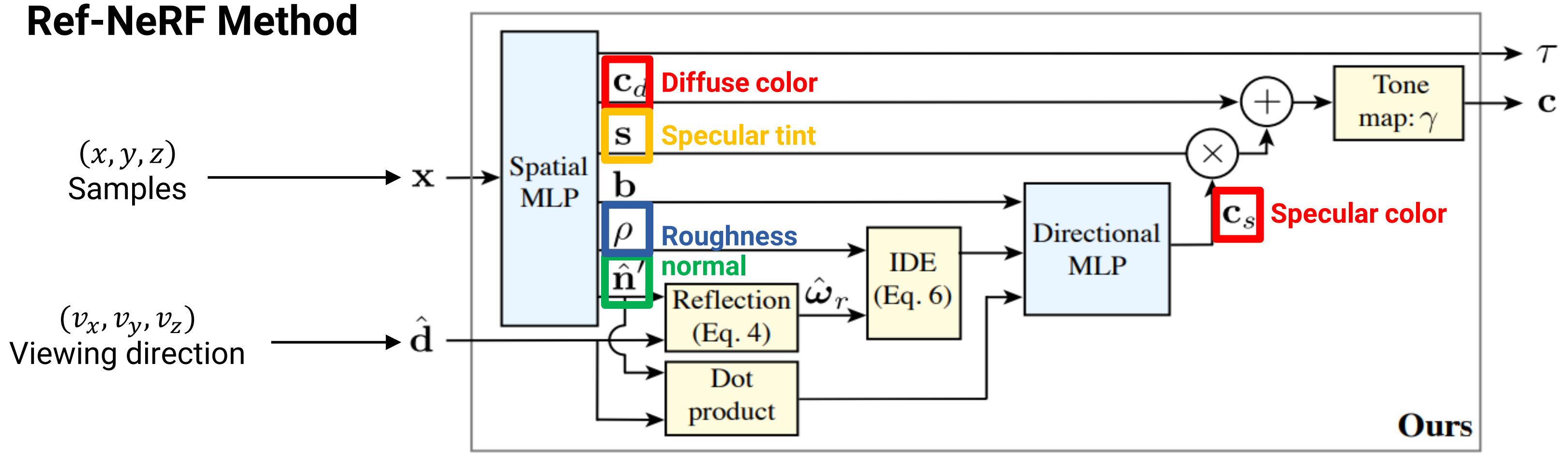
- Unlike traditional NeRF, Ref-NeRF calculate final RGB color by using meaningful outputs
  - (nomal vector, roughness, diffuse color, specular color, specular tint)
- When synthesizing novel view scene, these meaningful components enables more reasonable prediction

Original Method

τ : density (similar meaning to α)

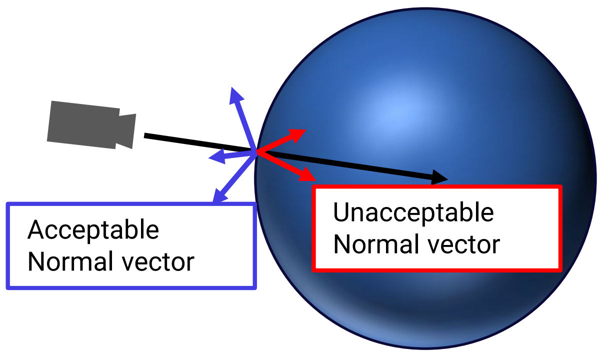


Ref-NeRF Method



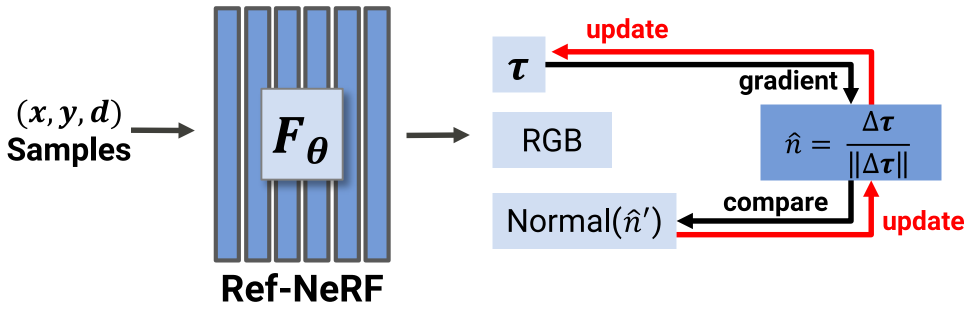
Ref-NeRF: compute normal vectors

- Calculate normal vector by two methods
  - Predicted by network ($\hat{n}'$: predict normal vector)
  - Gradient of density ($\hat{n}$: ground truth normal vector)
- Normal vector of the object seen on the camera should be facing the camera
  - Reduce reverse object



- Accurate normal vector ↔ Accurate alpha ↔ Accurate reflect direction

Learning normal vector in Network



G.T. normal: $\hat{n}(x) = \frac{\nabla density(x)}{\|\nabla density(x)\|}$

$$R_1 = \sum_i w_i \| \boxed{\hat{n}_i} - \boxed{\hat{n}_i'} \|$$

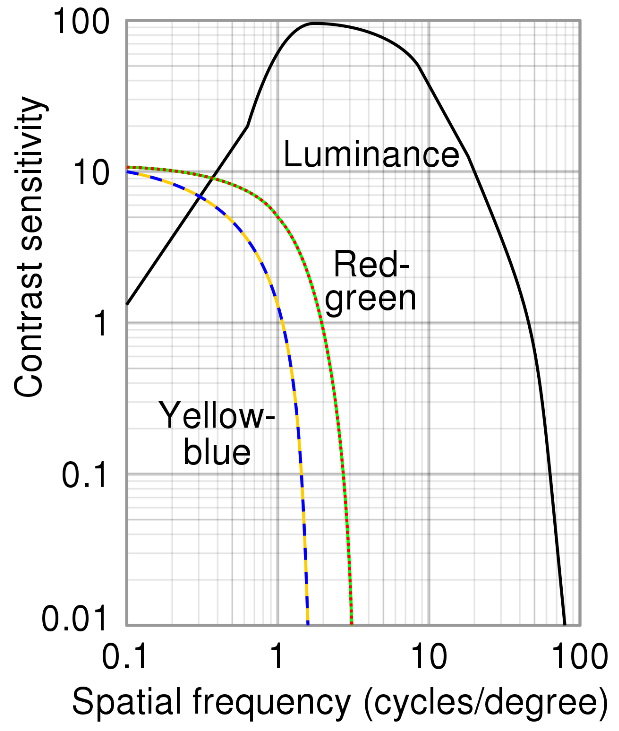
Predict normal vector

$$R_2 = \sum_i w_i max(0, \boxed{\hat{n}_i'} \; d)^2$$

YCbCr

- If we append meaningful components(normal vector, roughness, etc.) to output of our network, its size is much larger than original NeX.
- Then, learning is slow and it is difficult to get high quality rendering result.

- Human eyes are more sensitive to luminance than chromaticity
- As is often used for image compression, we used YCbCr instead of RGB for specular components.



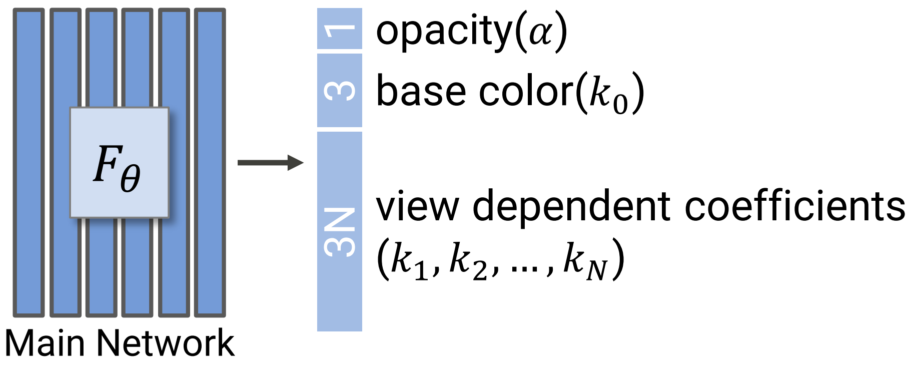
Our goals

- Convert RGB to YCbCr
- Decompose radiance to diffuse and specular components
- Calculate accurate normal vector and alpha
- Synthesize novel view with sharp highlights

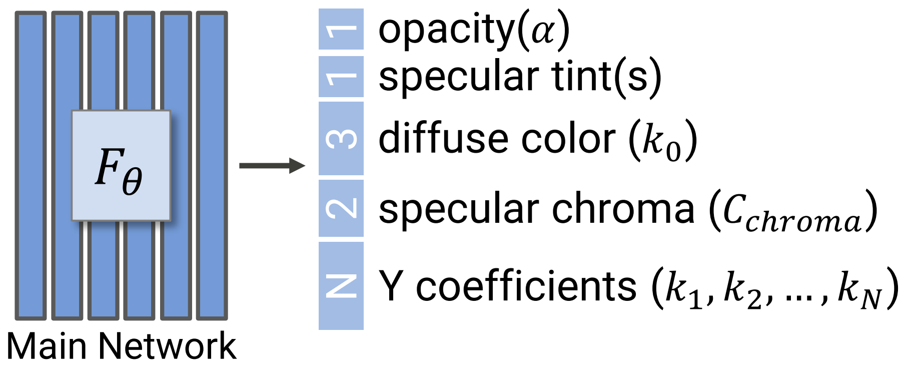
RGB → YCbCr

- Original NeX
  - Output of main network : 1+3+3N (N=8) = 28
  - Each pixel of view dependent MPI : $\alpha, k_0(RGB), k_1(RGB), k_2(RGB), \dots, k_N(RGB)$
  - RGB color of each pixel : $C^P = k_0 + \sum_{n=1}^{N} k_n^P \times H_n(v)$
- YCbCr
  - Output of main network : 1+1+3+2+N (N=8) = 15
  - Each pixel of view dependent MPI : $\alpha, s, k_0(RGB), chroma(CbCr), k_1(Y), k_2(Y), \dots, k_N(Y)$
  - RGB color of each pixel : $C^P = k_0 + chroma \times \sum_{n=1}^{N} k_n^P \times H_n(v)$

Original NeX



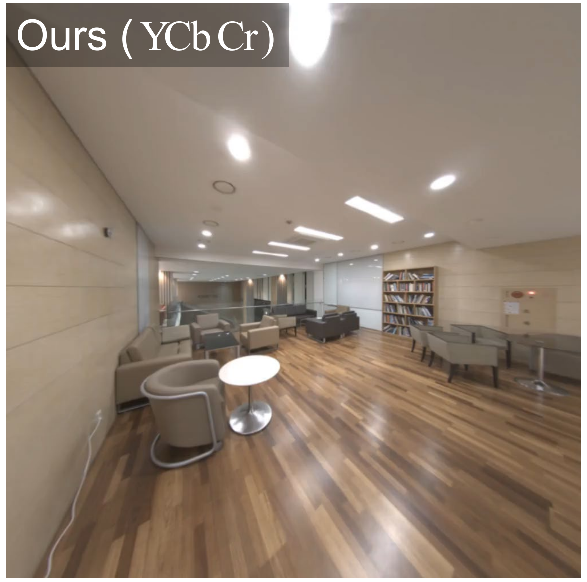
YCbCr



RGB → YCbCr

- The rendering quality of MPI using YCbCr was similar to original NeX

Original NeX (RGB)



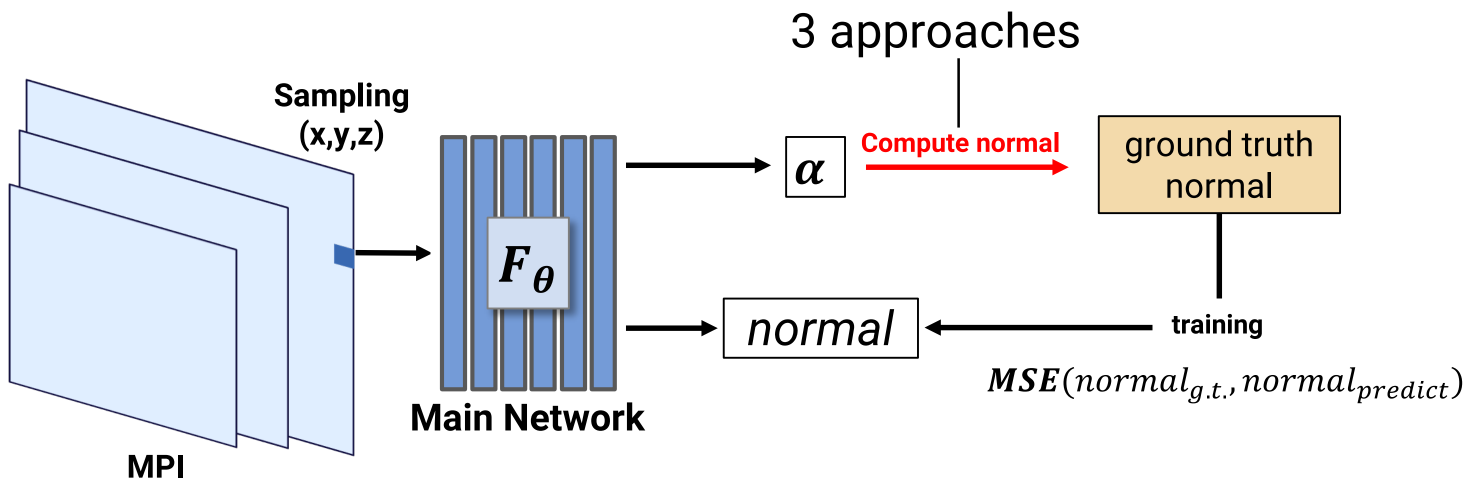
	PSNR ↑	SSIM ↑	LPIPS ↓
NeX	<u>30.7291</u>	<u>0.9855</u>	0.2059
Ours	28.7641	0.9845	<u>0.2036</u>

Computing normal vectors in NeX

- Norm-NeX compute the ground truth normal first and encourage model to predict the g.t. normal
- We try 3 approaches to compute normal vectors which is regarded as g.t. normal in NeX

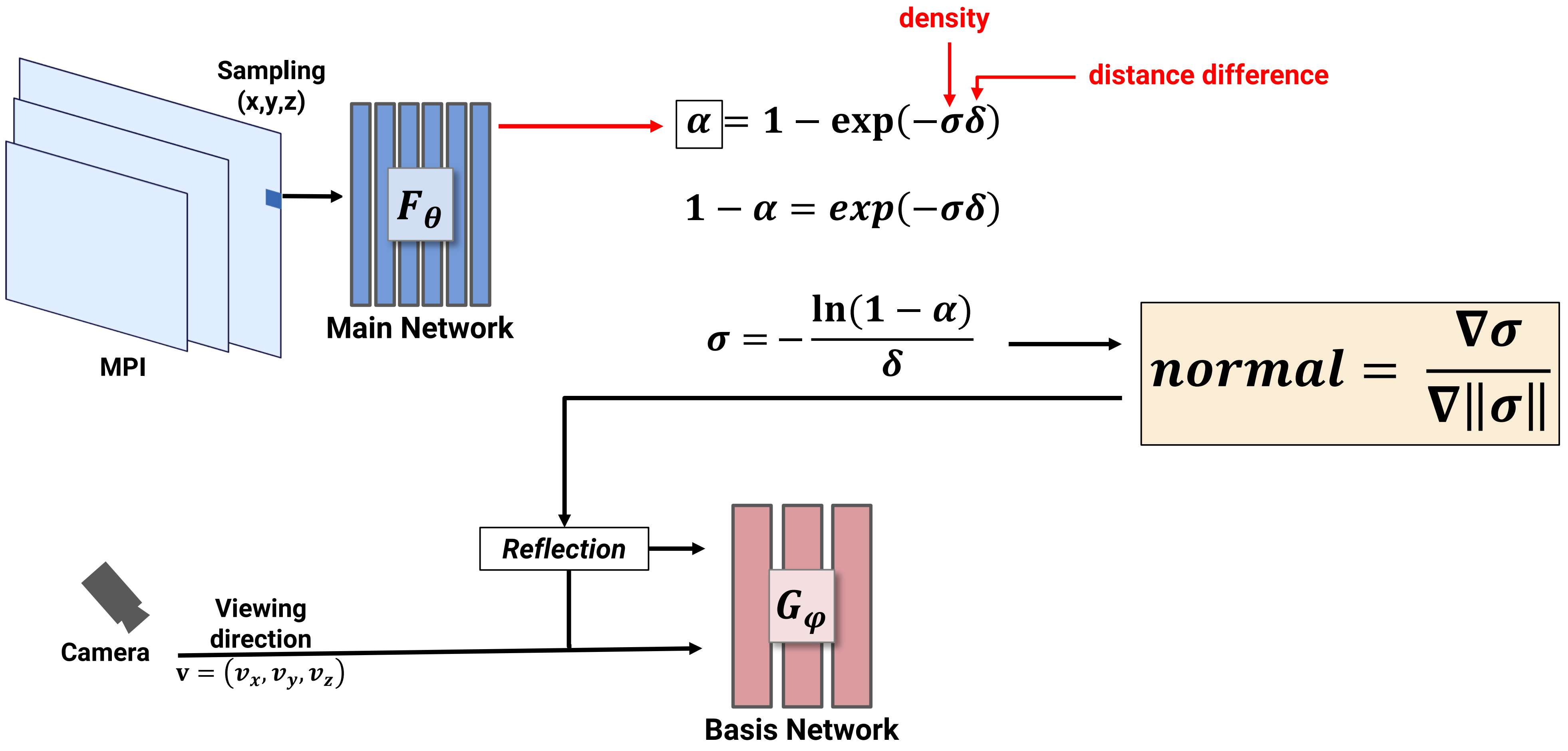
Approaches

1. Compute the density difference across the 3D space (Naïve approach)
2. Compute the normal vectors using the loss gradient relation
3. Compute the normal vectors on the rendered 2D images



Normal - Density difference (Naïve Approach)

- Compute the density using the alpha values which is the result outputs of the Norm-NeX
- Use the difference of density as a normal vectors



Sampling (x,y,z)
MPI
F_θ
Main Network
density
distance difference
$\boxed{\alpha} = 1 - \exp(-\sigma\delta)$
$1 - \alpha = exp(-\sigma\delta)$
$\sigma = -\frac{\ln(1-\alpha)}{\delta}$
$normal = \frac{\nabla\sigma}{\nabla\|\sigma\|}$
Reflection
Camera
Viewing direction
$\mathrm{v} = (v_x, v_y, v_z)$
G_φ
Basis Network

Normal – Gradient relation from Loss

- Compute the normal vectors using the gradient relation w.r.t. the total loss
- Density gradient can be decomposed of the loss gradient components
- x,y,z gradient over density gradient w.r.t. the reconstruction loss would equal with the density gradient w.r.t. x,y,z that means the normal vectors

$$\nabla\sigma = \left(\frac{d\sigma}{dx}, \frac{d\sigma}{dy}, \frac{d\sigma}{dz}\right)$$

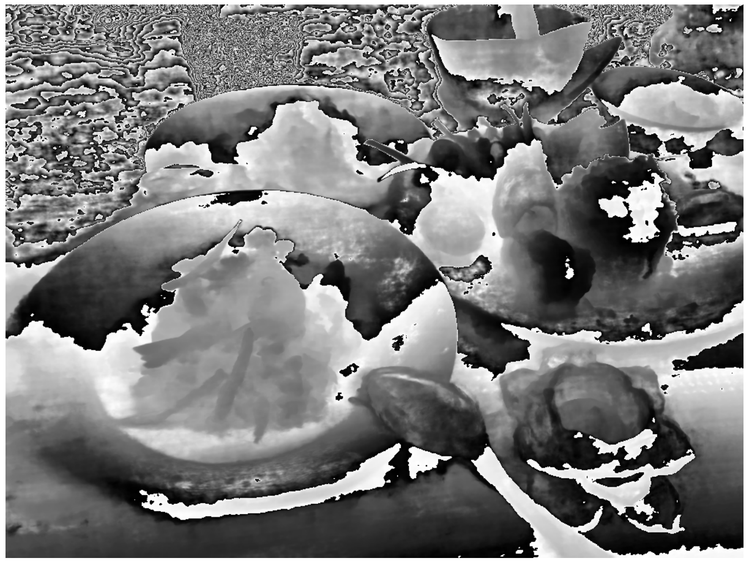
$$= \left(\frac{d\sigma}{\boldsymbol{dL}} \times \frac{\boldsymbol{dL}}{dx}, \frac{d\sigma}{\boldsymbol{dL}} \times \frac{\boldsymbol{dL}}{dy}, \frac{d\sigma}{\boldsymbol{dL}} \times \frac{\boldsymbol{dL}}{dz}\right) \longleftarrow$$

Decompose w.r.t. loss

$$= \frac{d\sigma}{\boldsymbol{dL}}\left(\frac{\boldsymbol{dL}}{dx}, \frac{\boldsymbol{dL}}{dy}, \frac{\boldsymbol{dL}}{dz}\right)$$

$$= \frac{1}{\sigma.grad}(x.grad, y.grad, z.grad) \longrightarrow$$

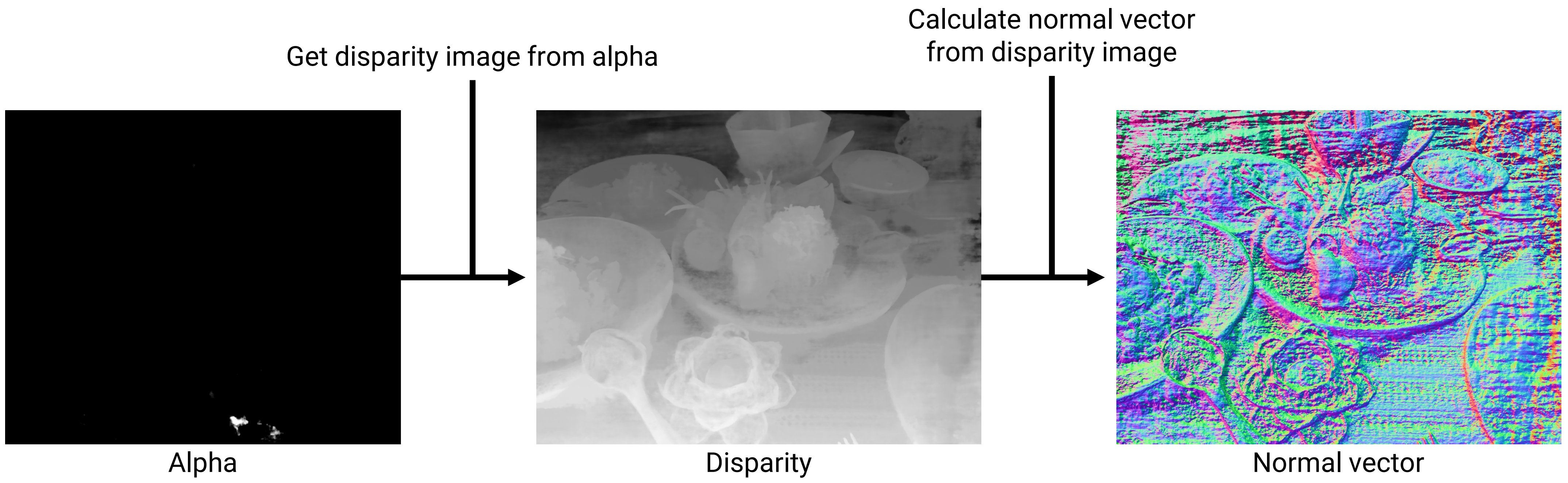
Can be calculated by pytorch autograd method


Computed normal by autograd

Normal – On the rendered 2D images

- Instead of calculating normal $\mathrm{Loss}(\hat{n}, \hat{n}')$ in the 3D space(MPI), we made the normal vector from the 2D images and calculated loss
  - $Alpha\ map \rightarrow G.T.Image_{disparity} \rightarrow G.T.Image_{normal\ vec}\ (I_{\hat{n}})$
  - Consider the normal vector(n_x, n_y, n_z) as RGB, then render image $\rightarrow Predicted\ Image_{normal\ vec}(I_{\hat{n}'})$
  - $Loss = MSE\ of\ (I_{\hat{n}}, I_{\hat{n}'})$

Alpha map made by original NeX for explanation


Get disparity image from alpha
Calculate normal vector from disparity image
Alpha
Disparity
Normal vector

Training Normal

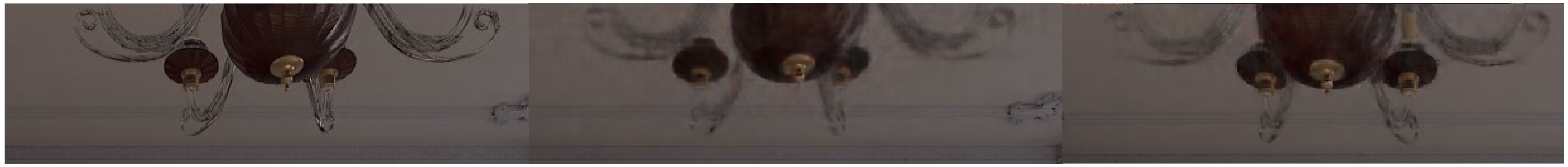
- Using pytorch autograd method (2nd) requires model's double evaluation
  - Double the training time (NeX model already spends 10 hours for 1 scene training…)

- Image based method (3rd) would not work well our model
  - Debugging is in progress for better result

- We simply choose the **first approach** to compute normal (density gradients)
  - We still test other approaches to get the better result

Normal Regularizer (Similar with refNeRF)

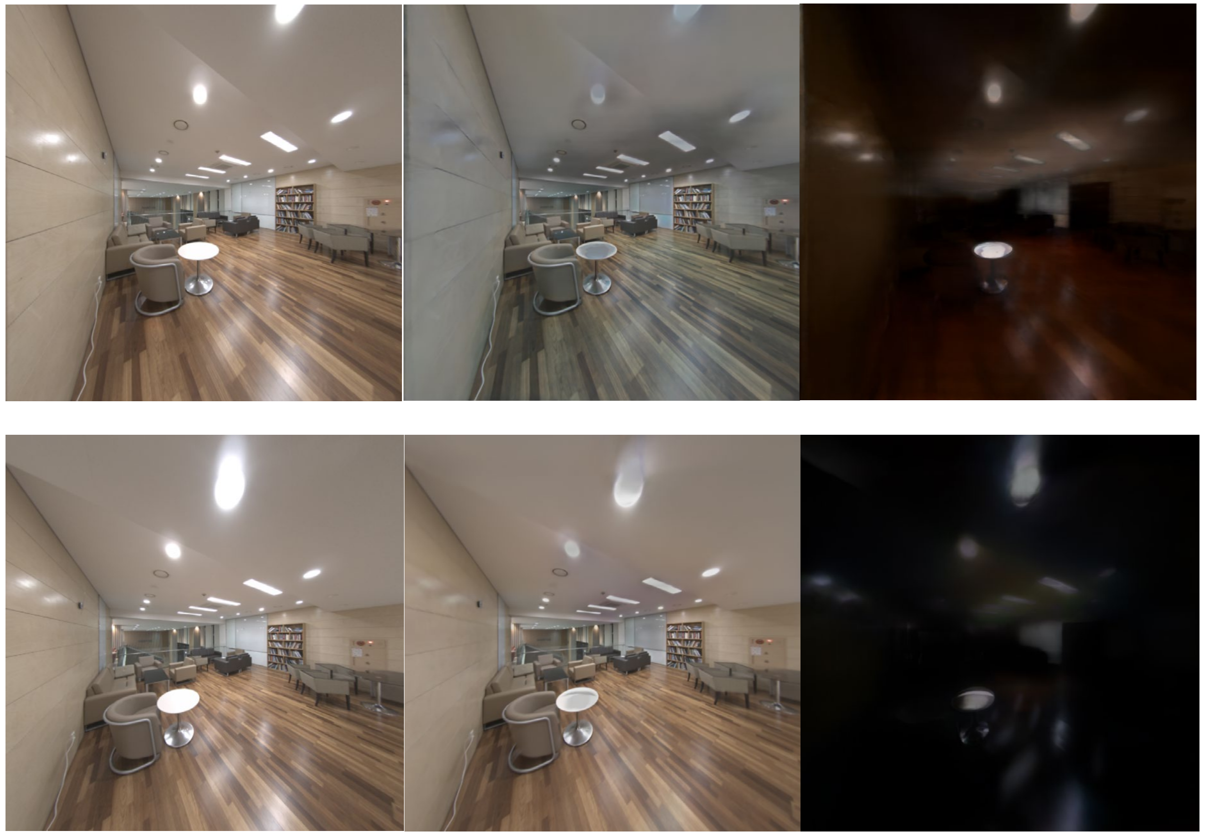
- Encourage model to reduce the back-facing normal vectors
- Make appearance more vivid



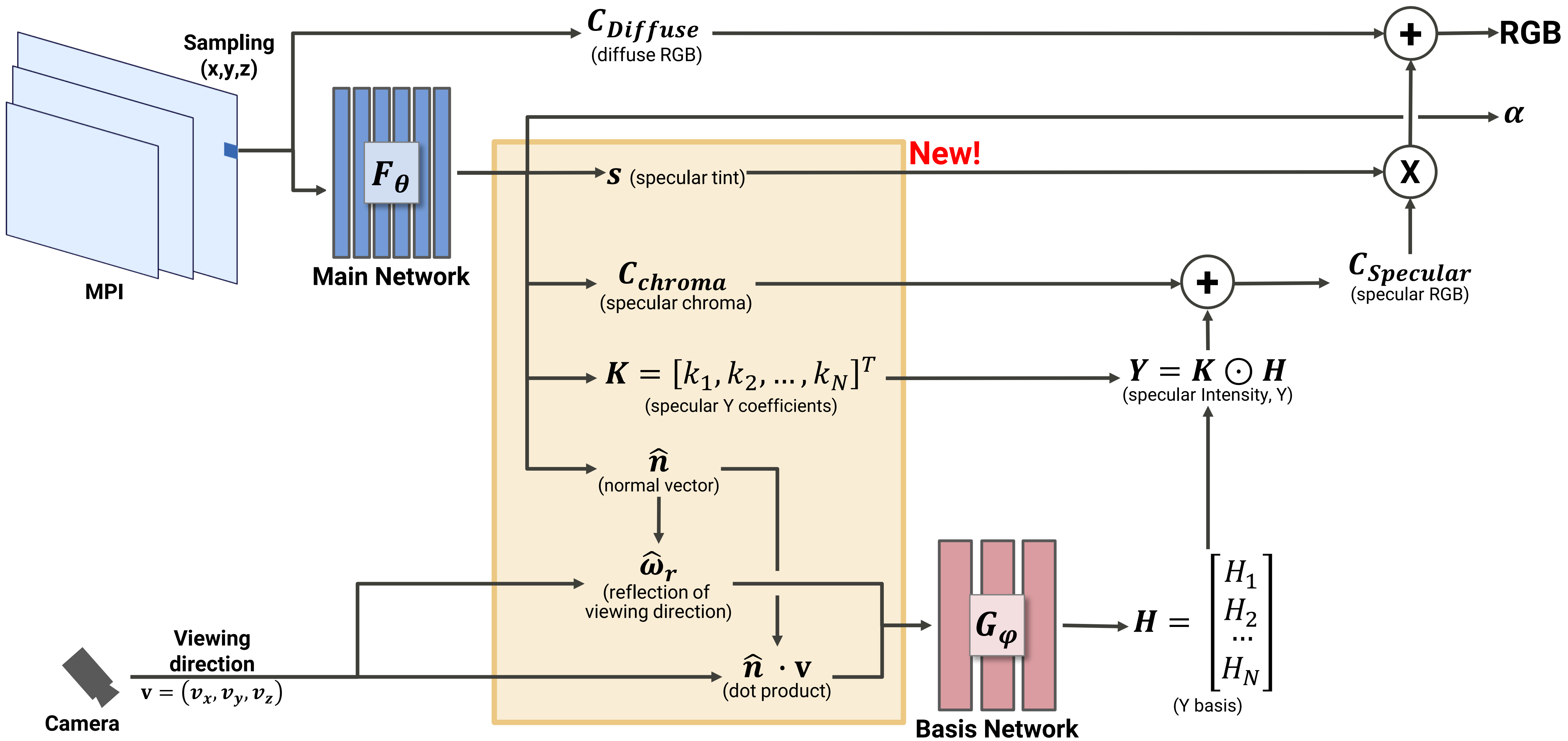
Ground truth | Without regularizer | With regularizer

Dividing specular and diffuse color

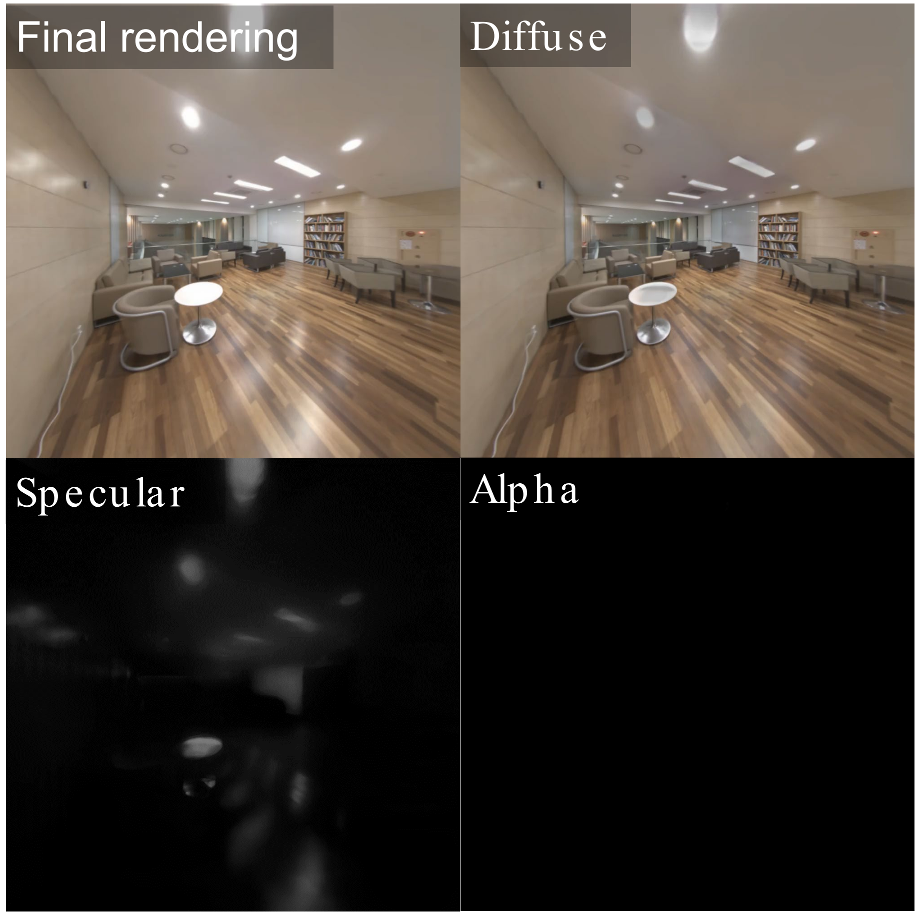
- If model trains the specular and diffuse color at the same time, model confuses with them
- Therefore, we set the threshold epoch (50 epochs, 2.5% of the entire epoch) and during that epochs we encourage model to render the images only with diffuse color


Training diffuse, specular at the same time
Training diffuse first during the 50 epochs (2.5%)
Rendered
Diffuse
Specular

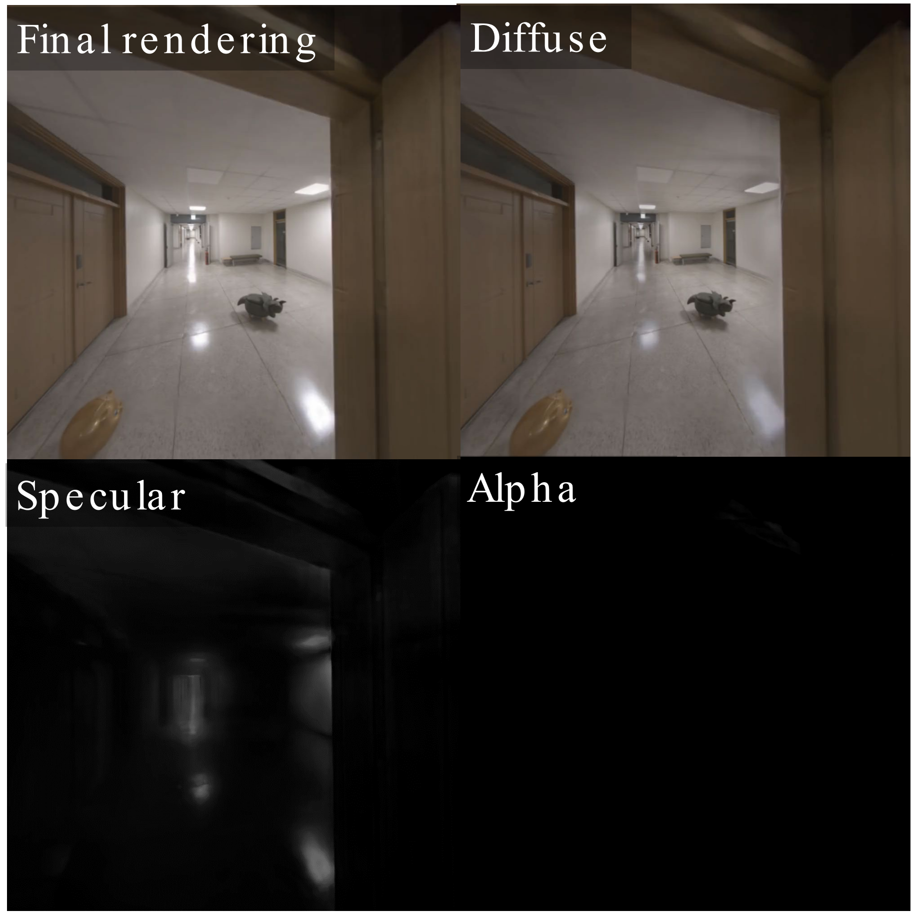
Norm-NeX pipeline



Training Results



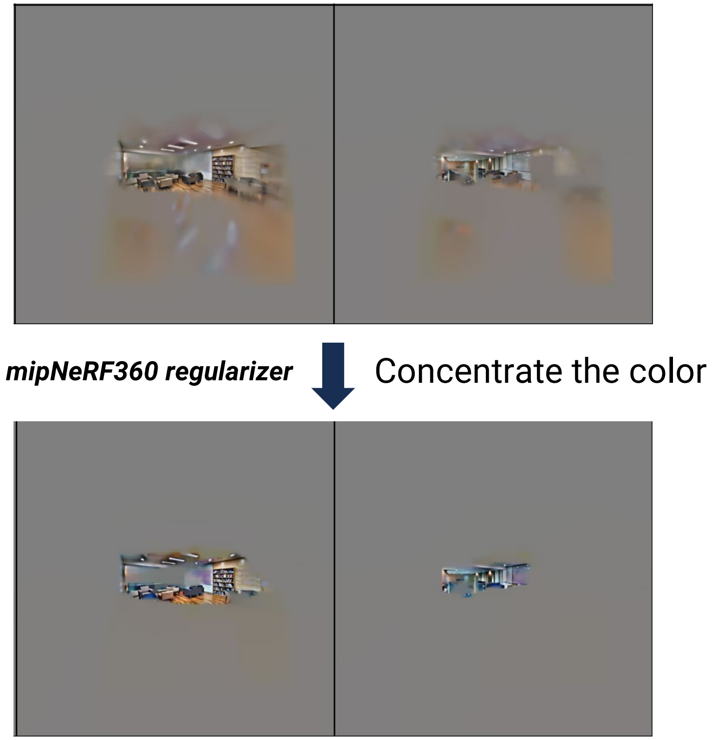
	PSNR ↑	SSIM ↑	LPIPS ↓
NeX	<u>30.7291</u>	<u>0.9855</u>	0.2059
Ours	28.76	0.9845	<u>0.2036</u>



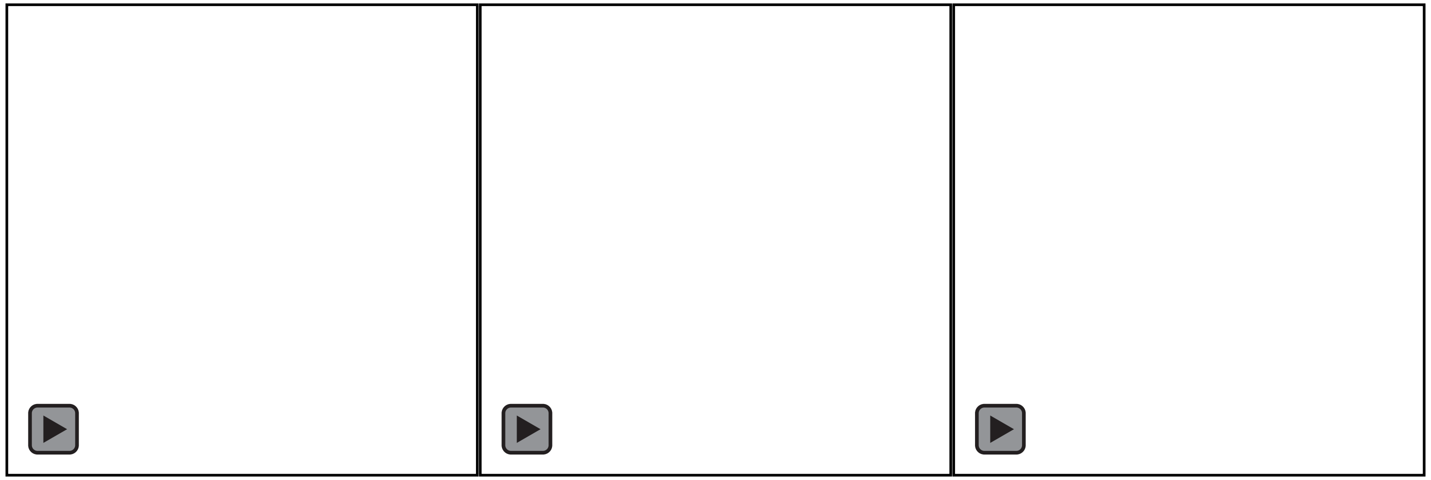
	PSNR ↑	SSIM ↑	LPIPS ↓
NeX	30.92	0.9741	<u>0.1523</u>
Ours	<u>33.64</u>	<u>0.9859</u>	0.3202

Ablation studies over mipNeRF360 Regularizer

- In order to get more accurate normal vectors, we apply mipNeRF360 regularizer which concentrates the weight on the important location
- Although it helps to concentrate the components to the important place, since it occurs floating artifacts we remove that regularizer from our model



Last two diffuse color layers



With mipNeRF360 regularizer

Conclusion

- We switched the color space from RGB to YCbCr
  - Maintain rendering quality while significantly reducing the size of the network output
  - Required memory also be reduced to store MPI
- It’s not perfect, but we got diffuse and specular components separately

- Future plan
  - Need to calculate much more accurate normal vector and alpha
  - Synthesize novel view with sharp highlights

Roles

- Jaemin Cho
  - Design Norm-NeX pipeline
  - Convert RGB to YCbCr
  - Make functions to visualize components
  - Make presentation

- Dongyoung Choi
  - Design Norm-NeX pipeline
  - Develop model with normal vector
  - Decompose diffuse and specular components
  - Make presentation

Reference

- Suttisak Wizadwongsa et al., *NeX: Real-time View Synthesis with Neural Basis Expansion*, CVPR, 2021
- Dor Verbin et al., *Ref-NeRF: Structured View-Dependent Appearance for Neural Radiance Fields*, CVPR, 2022
- Jonathan T. Barron et al., *Mip-NeRF 360: Unbounded Anti-Aliased Neural Radiance Fields*, CVPR, 2022