NeRF



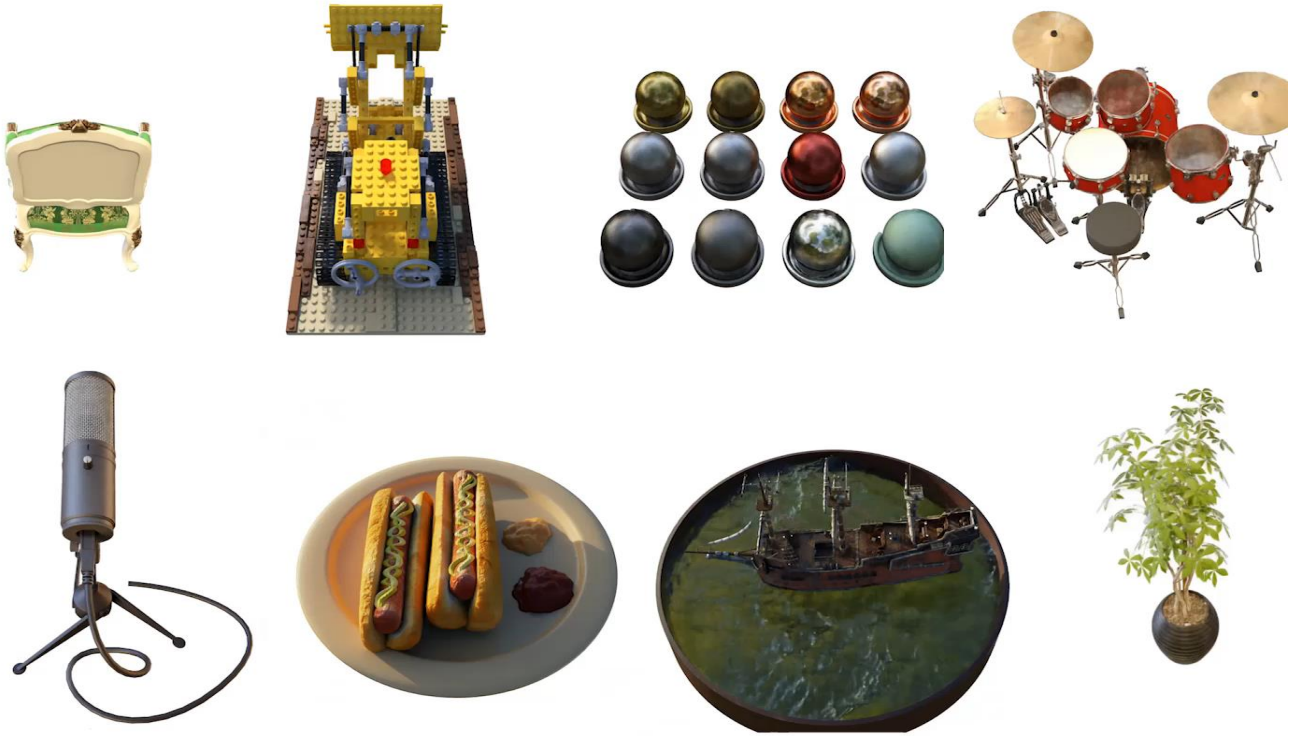
Presented By
박수혁

NeRF?

- **Novel View Synthesis**

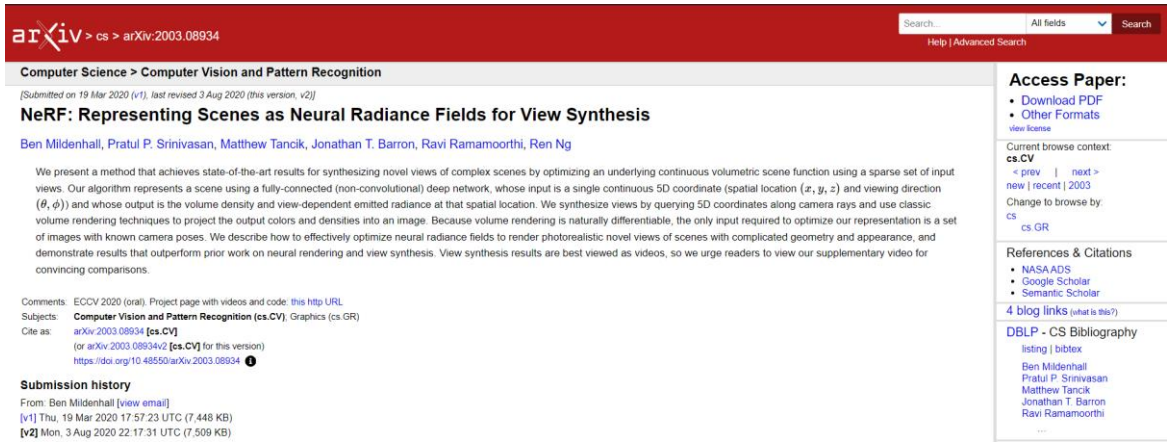
  - 특정 몇 개의 view에서 촬영한 이미지들로부터 새로운 시점(novel view)에서 바라본 scene의 이미지를 합성해내는 task

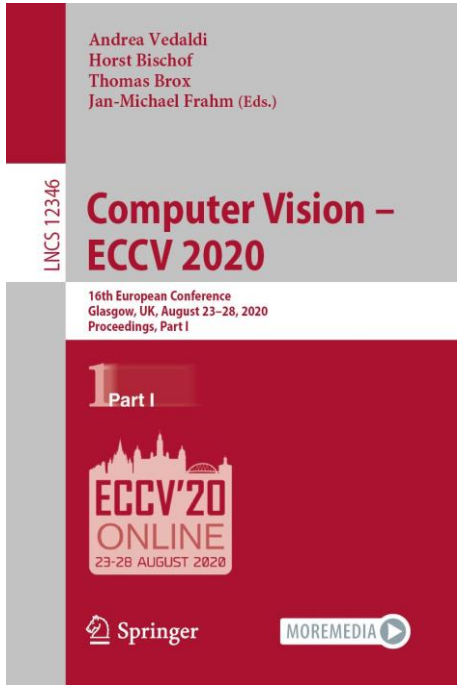
  - 대표적인 Method : NeRF, 3D Gaussian Splatting



Review Paper

- **Ben Mildenhall, et al. “NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis.”**
  - NeRF를 최초로 소개한 논문
  - ECCV 2020 Oral - Best Paper Honorable Mention
  - Application : 3D Reconstruction, Text to 3D, Pose Estimation 등





Motivation

- **Related Works**
  - ➢ Neural 3D Shape Representation
    - MLP를 이용한 3D shape의 implicit한 표현
    - 한계점 : low geometric complexity를 가진 simple한 shape에만 잘 적용.
  - ➢ View Synthesis / Image-Based Rendering
    - voxel grid 기반의 volumetric 표현 / mesh 기반 표현 (discrete representation)
    - 한계점 : 3D 공간을 잘 sampling 해야만 high-resolution 이미지 렌더링 가능하고 sample들에 대한 storage cost가 많이 듦.

- **Solution : Neural Radiance Fields for View Synthesis**

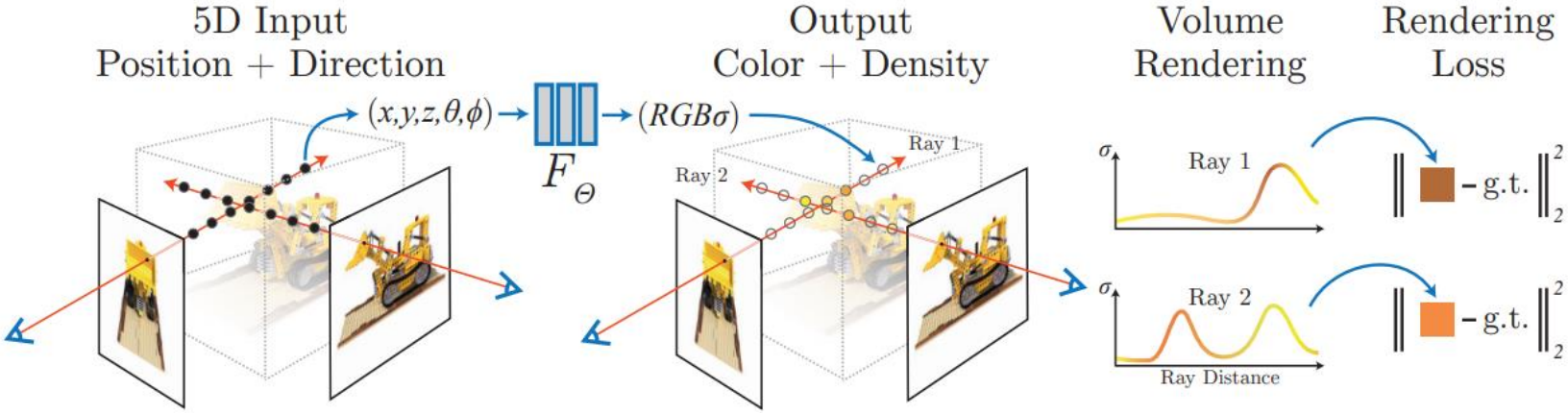
  - *Abstract: "We present a method that achieves state-of-the-art results for synthesizing novel views of complex scenes by optimizing an underlying continuous volumetric scene function using a sparse set of input views."*

Motivation

- **Related Works (Problems)**
  - 1. MLP 모델을 사용하고 있으나 complex한 object에 대해서는 적용하기 쉽지 않다.
  - 2. Volumetric technique이 view synthesis를 위한 괜찮은 기법이긴 하지만, 고해상도 이미지 렌더링을 위해서는 fine sampling이 필요하고 sample에 대한 storage cost가 많이 듦.
- **Design Idea**
  - ➢ MLP를 사용하되, complex한 구조의 scene에 대해서도 렌더링 가능하게 하자.
    - Positional Encoding
  - ➢ Classical volume rendering technique을 사용하되, sampling을 효과적으로 해보자.
    - Hierarchical Volume Sampling
  - ➢ 3D object 자체를 렌더링하는 것이 아닌, object를 바라본 모습(image)을 예측할 수 있는 모델을 만들어보자.
    - View-dependent emitted radiance($\mathbf{c}_i$)

Method

- **Overall Pipeline**



1. Camera에서 scene을 향해 view direction(θ, ϕ) 방향으로 ray를 쏜다. 즉 , camera로 찍은 2D image의 각 pixel 마다 하나의 ray를 쏘는 것. (각 ray는 하나의 pixel에 대응됨) 이때 ray위에 3D points(voxels)들을 sampling한다.

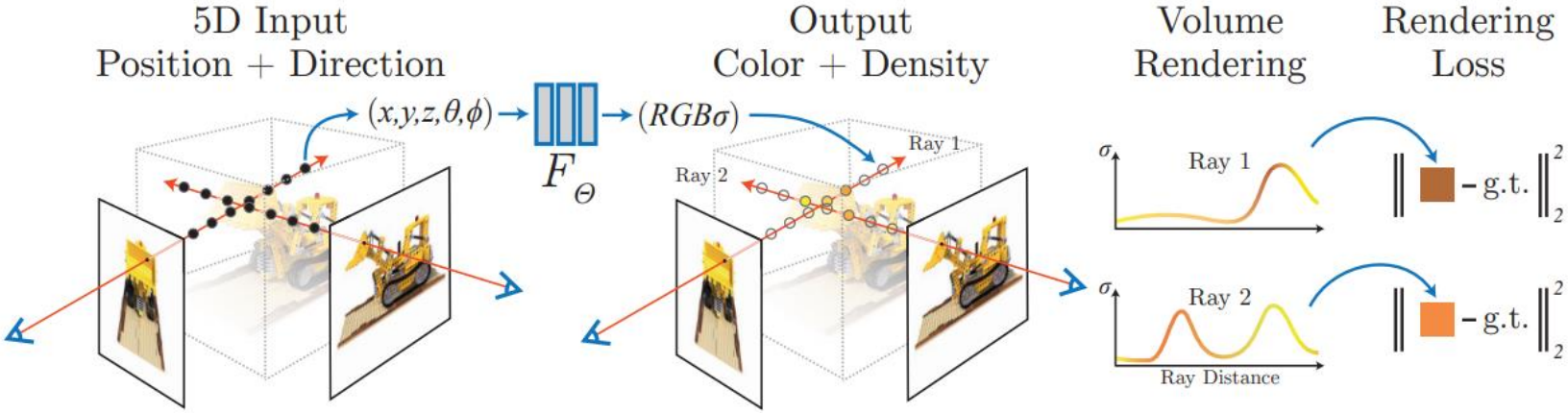
2. MLP의 input으로 각 3D point의 위치 $\mathbf{x} = (x, y, z)$와 viewing direction(θ, ϕ)이 들어가고 output으로 emitted radiance $\mathbf{c} = (r, g, b)$ volume density σ가 출력된다.

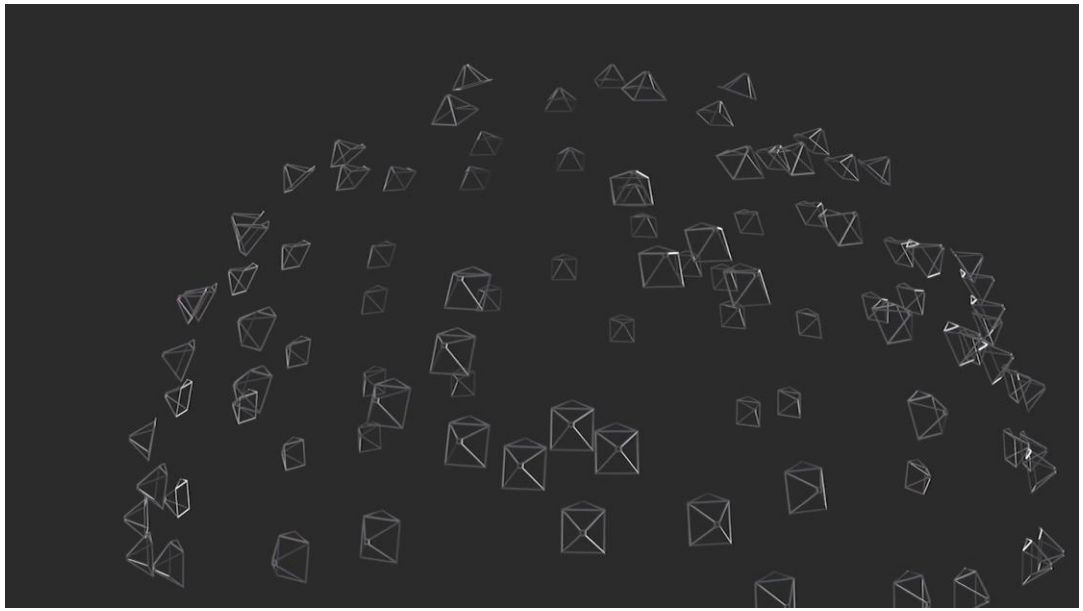
3. Weighted sum을 이용하여 2D pixel의 color를 계산한다. (volume rendering)

4. Loss를 계산하여 MLP network를 optimize한다.

Method

- **Overall Pipeline**



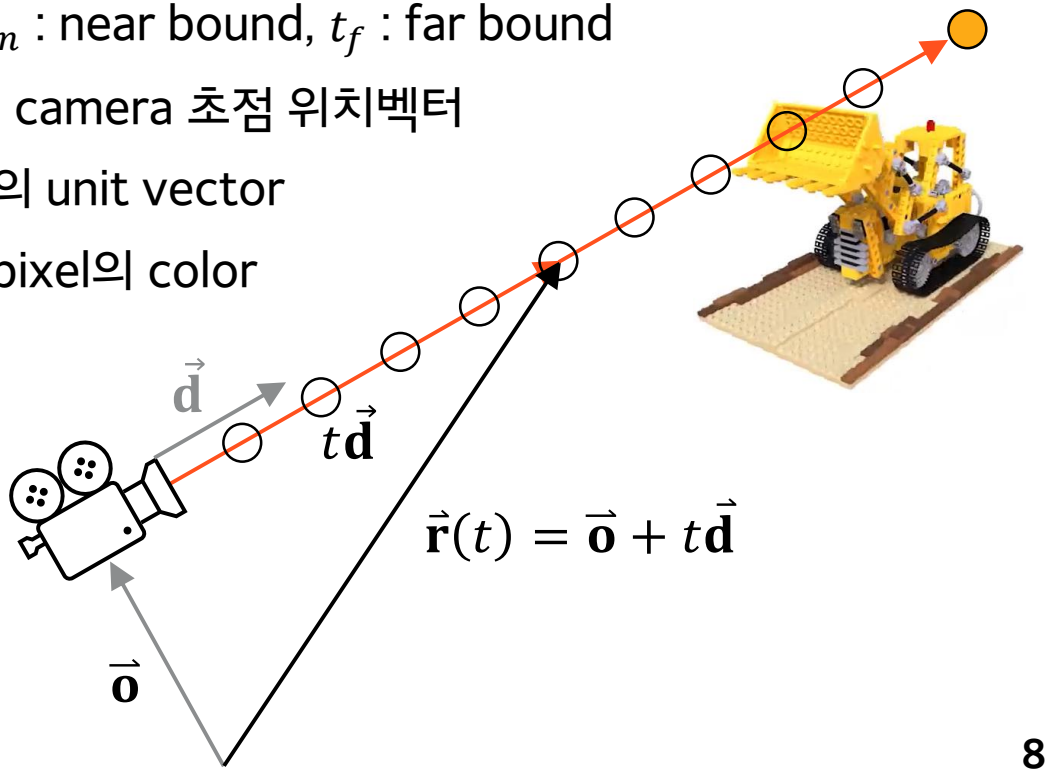


Method

- **Notation**
  - $\mathbf{x} = (x, y, z)$: 3D location of the sample point
  - (θ, ϕ) : Viewing direction → 직교좌표계 상의 방향벡터 $\mathbf{d}$로 표현
  - $\mathbf{c} = (r, g, b)$: emitted radiance (=RGB color)
  - σ : volume density
  - $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d} \quad (t_n < t < t_f)$ t_n : near bound, t_f : far bound

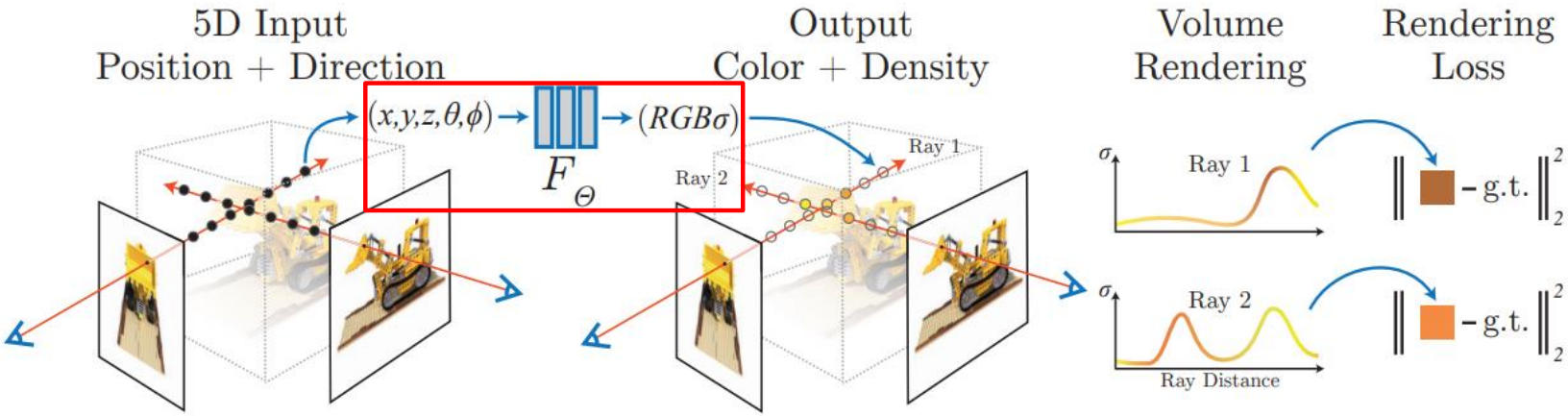
    $\mathbf{r}$: camera ray의 위치벡터 $\mathbf{o}$: camera 초점 위치벡터

    $\mathbf{d}$: viewing direction 방향으로의 unit vector
  - $C(\mathbf{r})$: camera ray $\mathbf{r}$에 대응하는 pixel의 color



Method

- **Overall Pipeline**



1. Camera에서 scene을 향해 view direction(θ, ϕ) 방향으로 ray를 쏜다. 즉 , camera로 찍은 2D image의 각 pixel 마다 하나의 ray를 쏘는 것. (각 ray는 하나의 pixel에 대응됨) 이때 ray위에 3D points(voxels)들을 sampling한다.

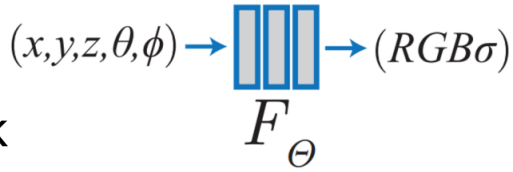
2. MLP의 input으로 각 3D point의 위치 $\mathbf{x} = (x, y, z)$와 viewing direction(θ, ϕ)이 들어가고 output으로 emitted radiance $\mathbf{c} = (r, g, b)$ volume density σ가 출력된다.

3. Weighted sum을 이용하여 2D pixel의 color를 계산한다. (volume rendering)

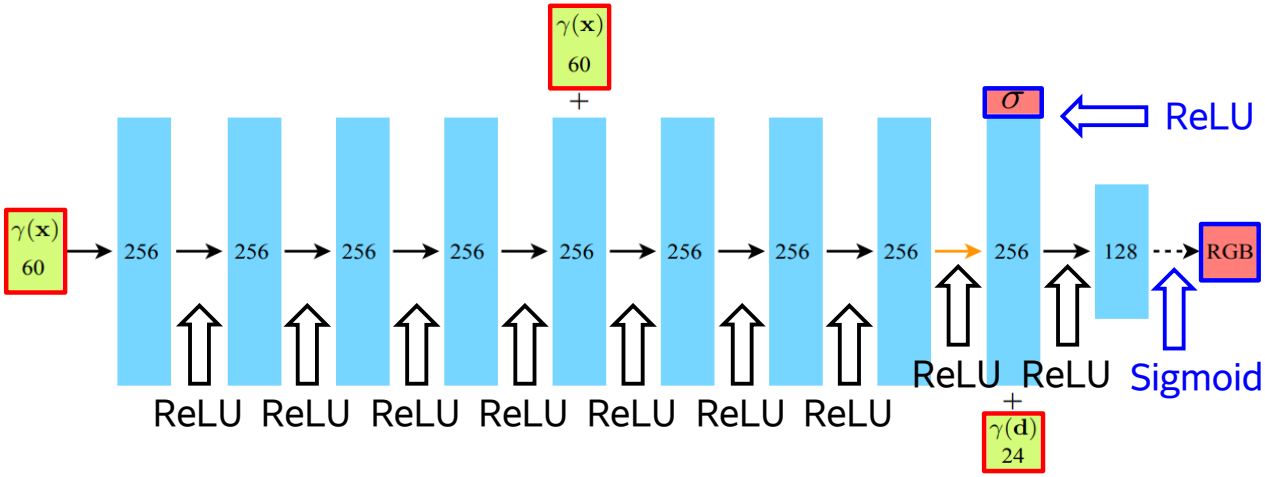
4. Loss를 계산하여 MLP network를 optimize한다.

Method

- **Network Architecture**



  - MLP(Multi-Layer Perceptron) = Fully-Connected Network

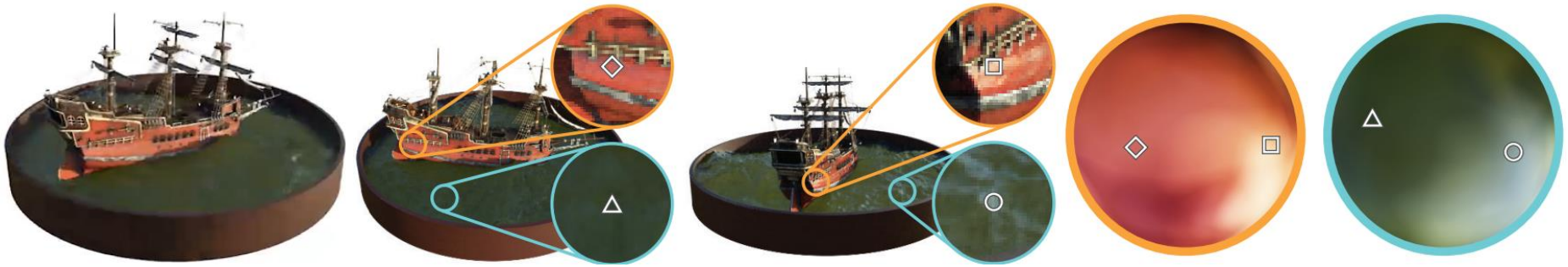


- $F_\Theta : (\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{c}, \sigma)$
- 학습 및 렌더링에 “Coarse Network”와 “Fine Network” 2가지 Network를 사용
- Activation Function : ReLU, Sigmoid
- Input : $\gamma(\mathbf{x})$, $\gamma(\mathbf{d})$ Output : σ, $\mathbf{c} = (r, g, b)$
- $\sigma \rightarrow$ function of $\mathbf{x}$, $\mathbf{c} = (r, g, b) \rightarrow$ function of $\mathbf{x}$ and $\mathbf{d} \rightarrow$ *view-dependent!*
- 학습시간 : NVIDIA V100 GPU로 1-2일 소요

Method

- **View-Dependent Emitted Radiance**
  - $\sigma \rightarrow$ function of $\mathbf{x}$, $\mathbf{c} = (r, g, b) \rightarrow$ function of $\mathbf{x}$ and $\mathbf{d} \rightarrow$ *view-dependent!*

  - 같은 3D 상의 지점을 다른 view direction에서 바라볼 때 radiance가 달라지면서 다른 RGB값을 갖게 함.



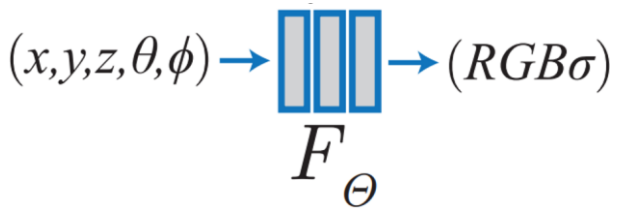
View 1 View 2 Radiance Distributions



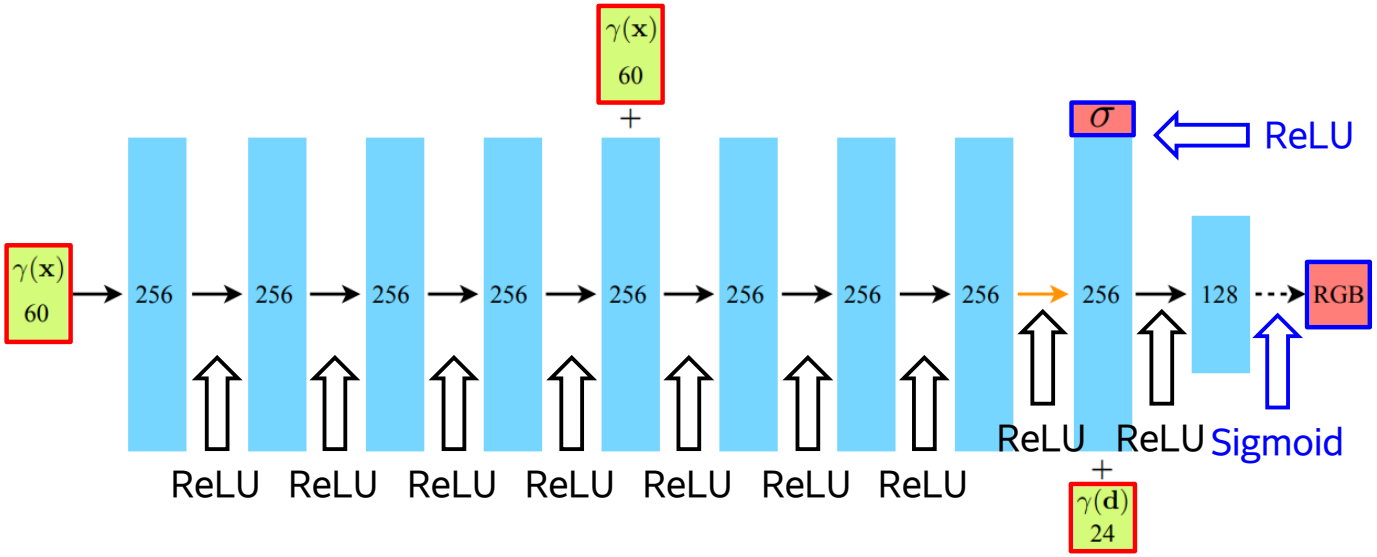


Method

- **Network Architecture**
  - MLP(Multi-Layer Perceptron)



$$F_\Theta : (\mathbf{x}, \mathbf{d}) \rightarrow (\mathbf{c}, \sigma)$$



Input : $\gamma(\mathbf{x})$, $\gamma(\mathbf{d})$ → Why??

Method

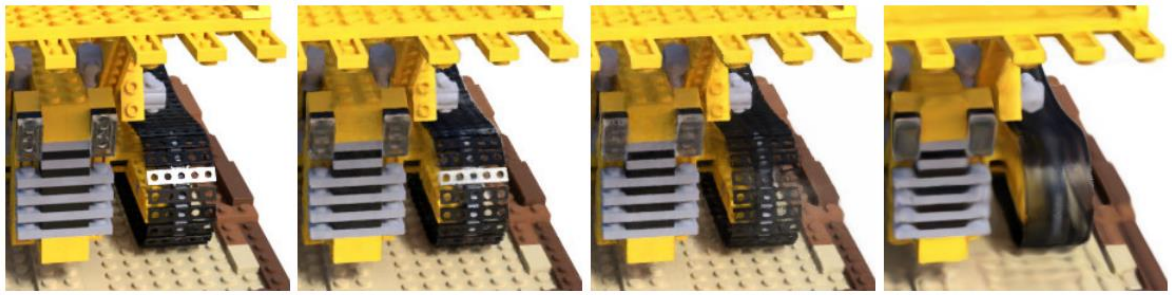
- **Related Works (Problems)**
  - 1. MLP 모델을 사용하고 있으나 complex한 object에 대해서는 적용하기 쉽지 않다.
  - 2. Volumetric technique이 view synthesis를 위한 괜찮은 기법이긴 하지만, 고해상도 이미지 렌더링을 위해서는 fine sampling이 필요하고 sample에 대한 storage cost가 많이 듦.
- **Design Idea**
  - ➢ MLP를 사용하되, complex한 구조의 scene에 대해서도 렌더링 가능하게 하자.
    - Positional Encoding
  - ➢ Classical volume rendering technique을 사용하되, sampling을 효과적으로 해보자.
    - Hierarchical Volume Sampling
  - ➢ 3D object 자체를 렌더링하는 것이 아닌, object를 바라본 모습(image)을 예측할 수 있는 모델을 만들어보자.
    - View-dependent emitted radiance($\mathbf{c}_i$)

Method

- **Positional Encoding**
  - $\mathbf{x}, \mathbf{d}$를 바로 MLP에 넣고 학습시킬 경우 color와 geometry에 high-frequency variation이 있는 scene에 대한 표현이 어색해짐
  - MLP에 넣기 전에 입력 dimension을 high dimension으로 높여주는 encoding 작업이 필요

$$F_\Theta = F'_\Theta \circ \gamma$$

$$\gamma : \mathbb{R} \rightarrow \mathbb{R}^{2L}$$



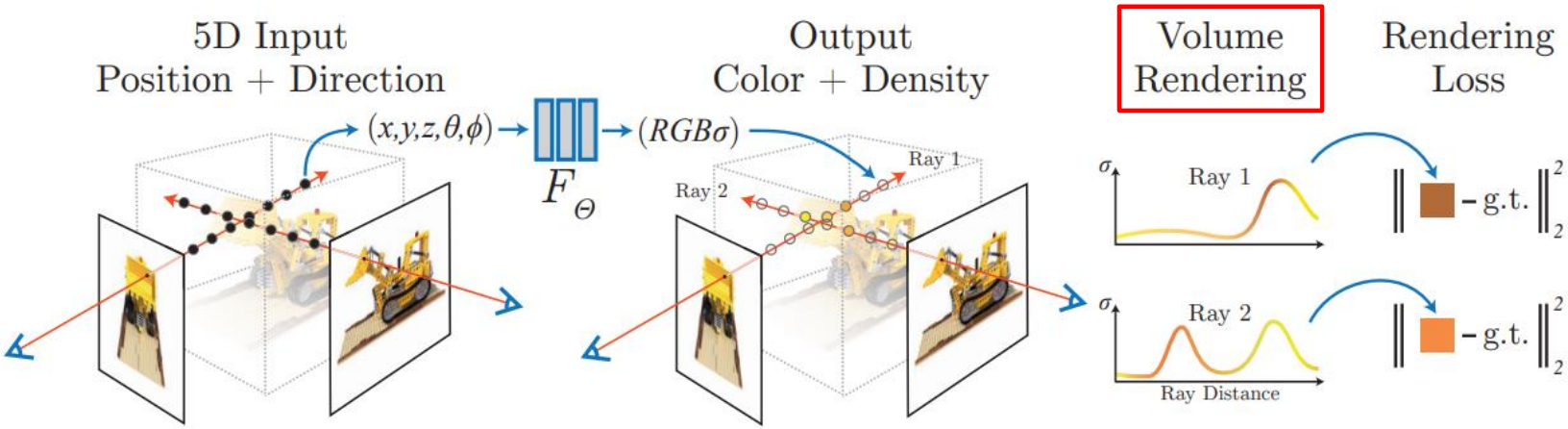
Ground Truth | Complete Model | No View Dependence | No Positional Encoding

$$\gamma(p) = \left(\sin\left(2^0\pi p\right), \cos\left(2^0\pi p\right), \cdots, \sin\left(2^{L-1}\pi p\right), \cos\left(2^{L-1}\pi p\right)\right)$$

- $\mathbf{x}$와 $\mathbf{d}$ 각각의 3개의 좌표에 대해 $\gamma(\cdot)$로 encoding하여 차원 증가 및 [-1,1] 범위로 제한
- $L = 10$ for $\gamma(\mathbf{x}) \rightarrow 60$ dimensions
- $L = 4$ for $\gamma(\mathbf{d}) \rightarrow 24$ dimensions

Method

- **Overall Pipeline**



1. Camera에서 scene을 향해 view direction(θ, ϕ) 방향으로 ray를 쏜다. 즉 , camera로 찍은 2D image의 각 pixel 마다 하나의 ray를 쏘는 것. (각 ray는 하나의 pixel에 대응됨) 이때 ray위에 3D points(voxels)들을 sampling한다.

2. MLP의 input으로 각 3D point의 위치 $\mathbf{x} = (x, y, z)$와 viewing direction(θ, ϕ)이 들어가고 output으로 emitted radiance $\mathbf{c} = (r, g, b)$ volume density σ가 출력된다.

3. Weighted sum을 이용하여 2D pixel의 color를 계산한다. (volume rendering)

4. Loss를 계산하여 MLP network를 optimize한다.

Method

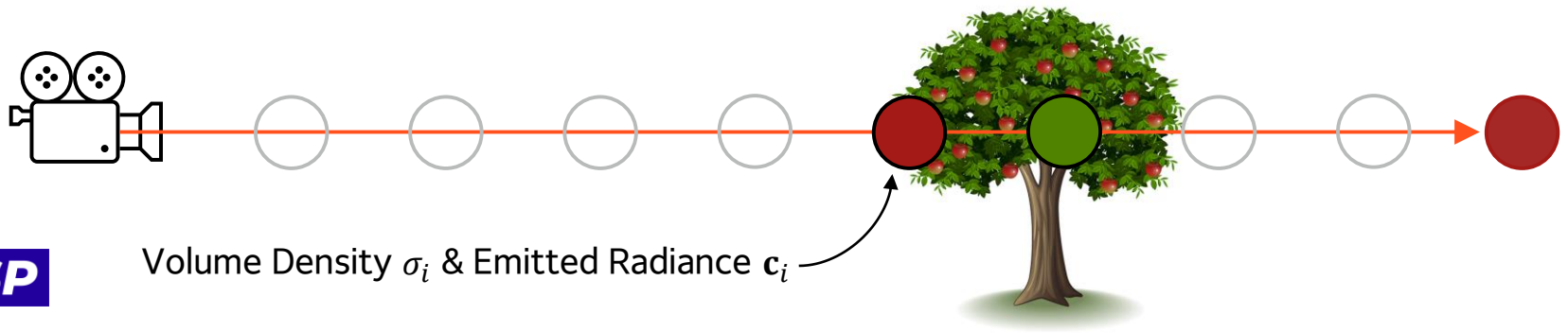
- **Volume Rendering**
  - Key Formula

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t)\sigma\big(\mathbf{r}(t)\big)\mathbf{c}(\mathbf{r}(t), \mathbf{d})dt \quad \text{where } T(t) = \exp\left(-\int_{t_n}^{t} \sigma\big(\mathbf{r}(s)\big)ds\right)$$

Discretization

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i \big(1 - \exp(-\sigma_i \delta_i)\big)\mathbf{c}_i \quad \text{where } T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right), \quad \delta_i = t_{i+1} - t_i$$

→ Meaning? : Pixel Color = Weighted sum of radiance $\mathbf{c}_i$



Method

- **Volume Rendering**
  - Weighted Sum?

Samples	선형대수학	확률통계	머신러닝	인공지능개론
Weight(w_i)	3학점	2학점	2학점	3학점
Radiance($\mathbf{c}_i$)	A0	B+	C+	A+

- 평점 : 0.3 × 4.0 + 0.2 × 3.5 + 0.2 × 2.5 + 0.3 × 4.5 = 3.75

Samples	선형대수학	확률통계	머신러닝	인공지능개론
Weight(w_i)	2학점	3학점	4학점	1학점
Radiance($\mathbf{c}_i$)	A0	B+	C+	A+

- 평점 : 0.2 × 4.0 + 0.3 × 3.5 + 0.4 × 2.5 + 0.1 × 4.5 = 3.3

Method

- **Volume Rendering**
  - Weighted sum of radiance $\mathbf{c}_i$

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i \left(1 - \exp(-\sigma_i \delta_i)\right) \mathbf{c}_i \quad \text{where } T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right),\ \delta_i = t_{i+1} - t_i$$

$$= \sum_{i=1}^{N} w_i \mathbf{c}_i \quad \text{where } w_i = T_i \left(1 - \exp(-\sigma_i \delta_i)\right)$$

- Transmittance (T_i) : 투과도, ray가 부딪히지 않고 i번째 sample까지 투과할 확률.
  → 1부터 $(i-1)$ 까지 축적된 sample들의 밀도의 합
- Opacity ($\alpha_i = 1 - \exp(-\sigma_i \delta_i)$) : 불투명도, volume density σ의 discrete한 표현.
- Emitted Radiance ($\mathbf{c}_i$) : i번째 sample의 color.

→ **Pixel Color = Σ {투과도 × 밀도(불투명도) × color}**

Method

- **Volume Rendering**
  - Weighted sum of radiance $\mathbf{c}_i$

- Volume Rendering
  - Weighted sum of radiance $\mathbf{c}_i$





$, \delta_i = t_{i+1} - t_i$

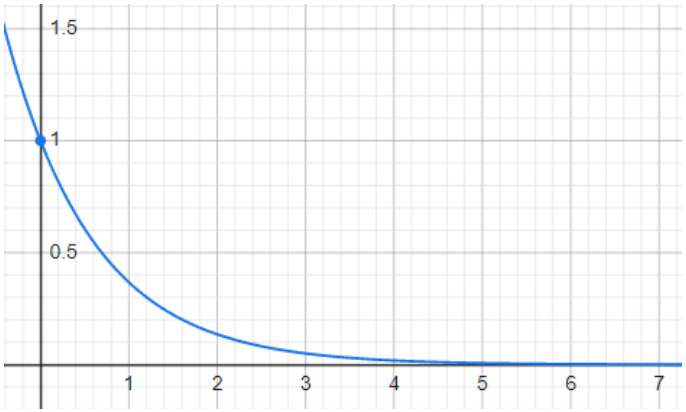
- Transmittance (T_i)



$$0 < T_i < 1$$

$$\uparrow \Rightarrow T_i \downarrow$$

- Transmittance (T_i)



$$0 < T_i < 1$$

$$\sum_{j=1}^{i-1} \sigma_j \delta_j \uparrow \Rightarrow T_i \downarrow$$

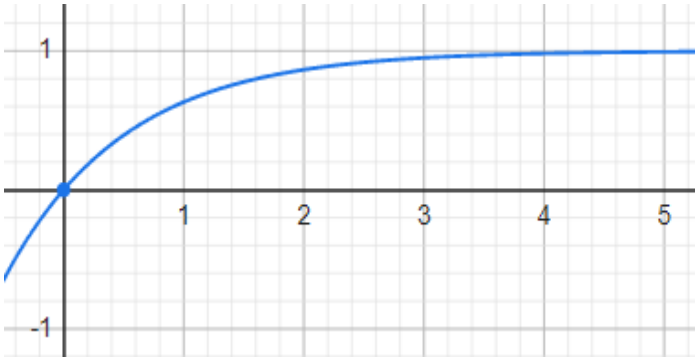
Method

- **Volume Rendering**
  - Weighted sum of radiance $\mathbf{c}_i$

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i \boxed{\left(1 - \exp(-\sigma_i \delta_i)\right)} \mathbf{c}_i \quad \text{where } T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right), \delta_i = t_{i+1} - t_i$$

$$= \sum_{i=1}^{N} w_i \mathbf{c}_i \quad \text{where } w_i = T_i \boxed{\left(1 - \exp(-\sigma_i \delta_i)\right)}$$

- Opacity ($\alpha_i = 1 - \exp(-\sigma_i \delta_i)$)



$$0 < \alpha_i < 1$$

$$\sigma_i \uparrow \Rightarrow \alpha_i \uparrow$$

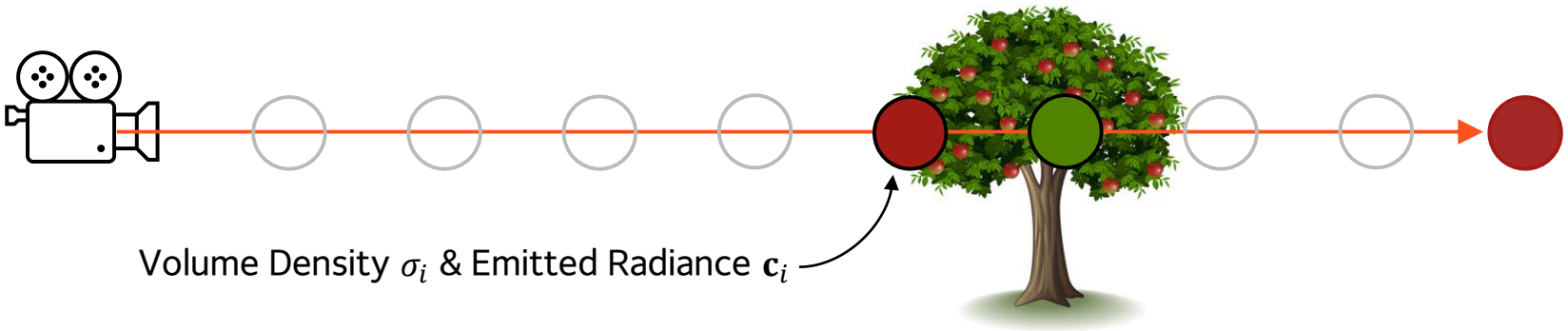
Method

- **Volume Rendering**
  - Weighted sum of radiance $\mathbf{c}_i$

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^{N} T_i \big(1 - \exp(-\sigma_i \delta_i)\big)\mathbf{c}_i \quad \text{where } T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right),\ \delta_i = t_{i+1} - t_i$$

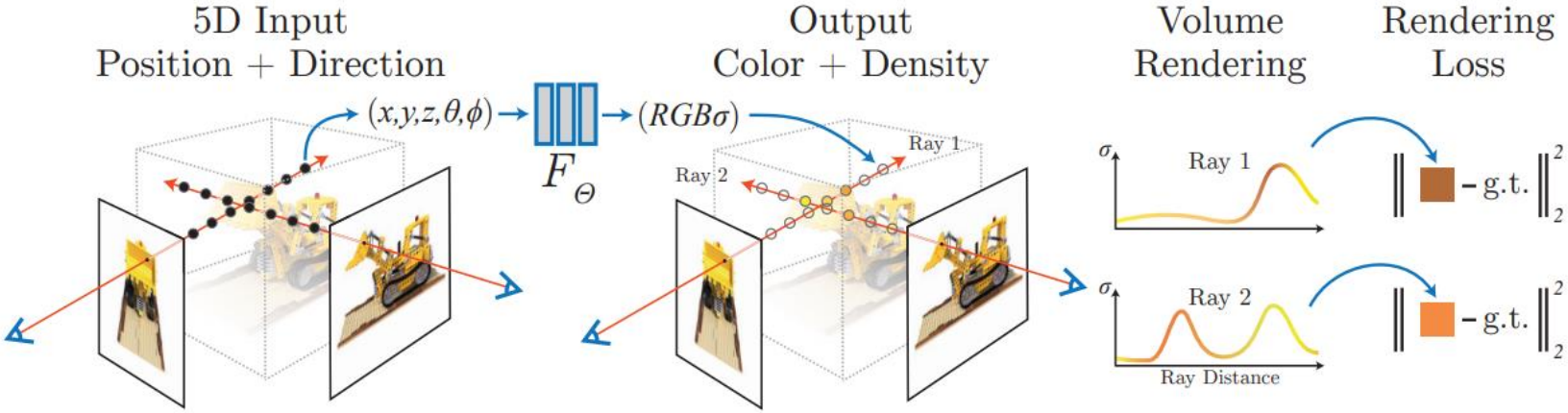
$$= \sum_{i=1}^{N} w_i \mathbf{c}_i \quad \text{where } w_i = T_i \big(1 - \exp(-\sigma_i \delta_i)\big)$$

- sample 자체의 volume density 뿐만 아니라 카메라에 가까운 sample들에 의해서 가려지는 효과(transmittance, T_i)까지 고려하여 weight를 줌으로써 pixel color를 합리적으로 계산



Method

- **Overall Pipeline**



1. Camera에서 scene을 향해 view direction(θ, ϕ) 방향으로 ray를 쏜다. 즉 , camera로 찍은 2D image의 각 pixel 마다 하나의 ray를 쏘는 것. (각 ray는 하나의 pixel에 대응됨)

이때 ray위에 3D points(voxels)들을 sampling한다. → How??

2. MLP의 input으로 각 3D point의 위치 $\mathbf{x} = (x, y, z)$와 viewing direction(θ, ϕ)이 들어가고 output으로 emitted radiance $\mathbf{c} = (r, g, b)$ volume density σ가 출력된다.

3. Weighted sum을 이용하여 2D pixel의 color를 계산한다. (volume rendering)

4. Loss를 계산하여 MLP network를 optimize한다.

Method

- **Related Works (Problems)**
  - 1. MLP 모델을 사용하고 있으나 complex한 object에 대해서는 적용하기 쉽지 않다.
  - 2. Volumetric technique이 view synthesis를 위한 괜찮은 기법이긴 하지만,

    고해상도 이미지 렌더링을 위해서는 fine sampling이 필요하고 sample에 대한 storage cost가 많이 듦.

- **Design Idea**
  - ➢ MLP를 사용하되, complex한 구조의 scene에 대해서도 렌더링 가능하게 하자.

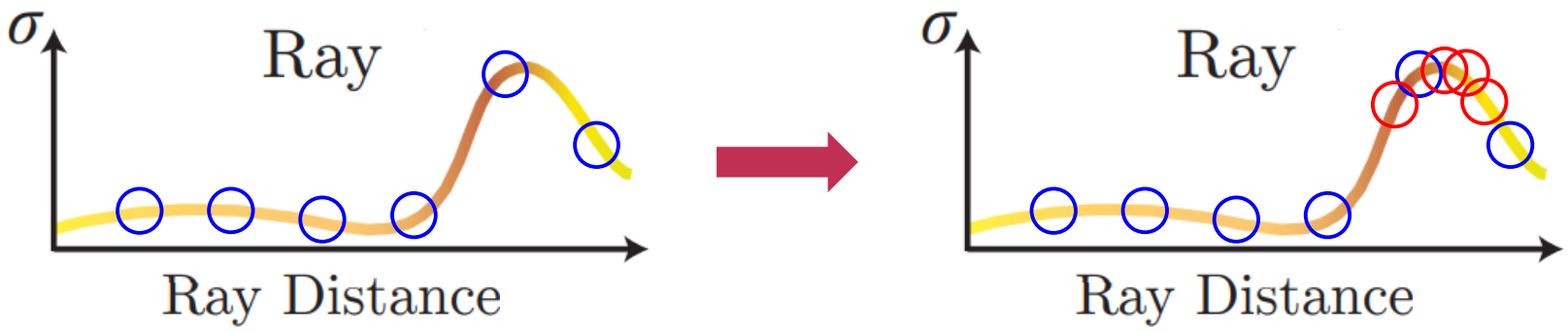
    - Positional Encoding
  - ➢ Classical volume rendering technique을 사용하되, sampling을 효과적으로 해보자.

    - Hierarchical Volume Sampling
  - ➢ 3D object 자체를 렌더링하는 것이 아닌, object를 바라본 모습(image)을 예측할 수 있는 모델을 만들어보자.

    - View-dependent emitted radiance($\mathbf{c}_i$)

Method

- **Hierarchical Volume Sampling**
  - ray위의 3D point(voxel)을 2번에 걸쳐 효율적으로 sampling 하는 것
  - 1차 Sampling : Stratified Sampling → “Coarse” Network
  - 2차 Sampling : Inverse Transform Sampling → “Fine” Network
  - 1차 Sampling 때는 주어진 ray 구간에 대해서 uniform한 확률로 random draw를 하고, 2차 Sampling 때 visible content가 있을 것으로 예상되는 region에 집중적으로 sample들을 뽑는 방식

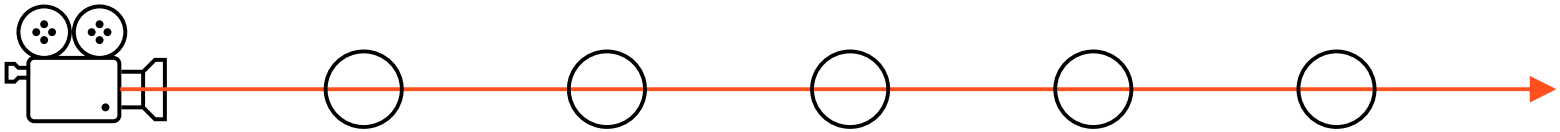


<1차 Sampling>

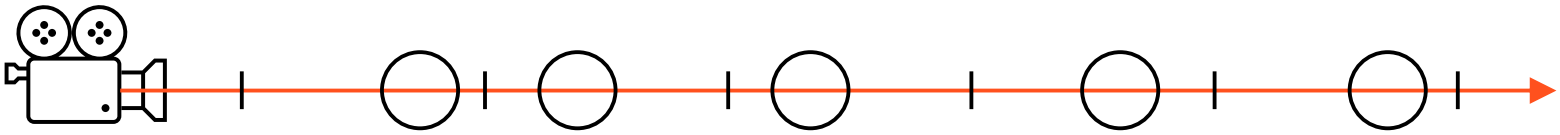
<2차 Sampling>

Method

- **Hierarchical Volume Sampling**
  - Stratified Sampling (1차 Sampling)



Deterministic Sampling



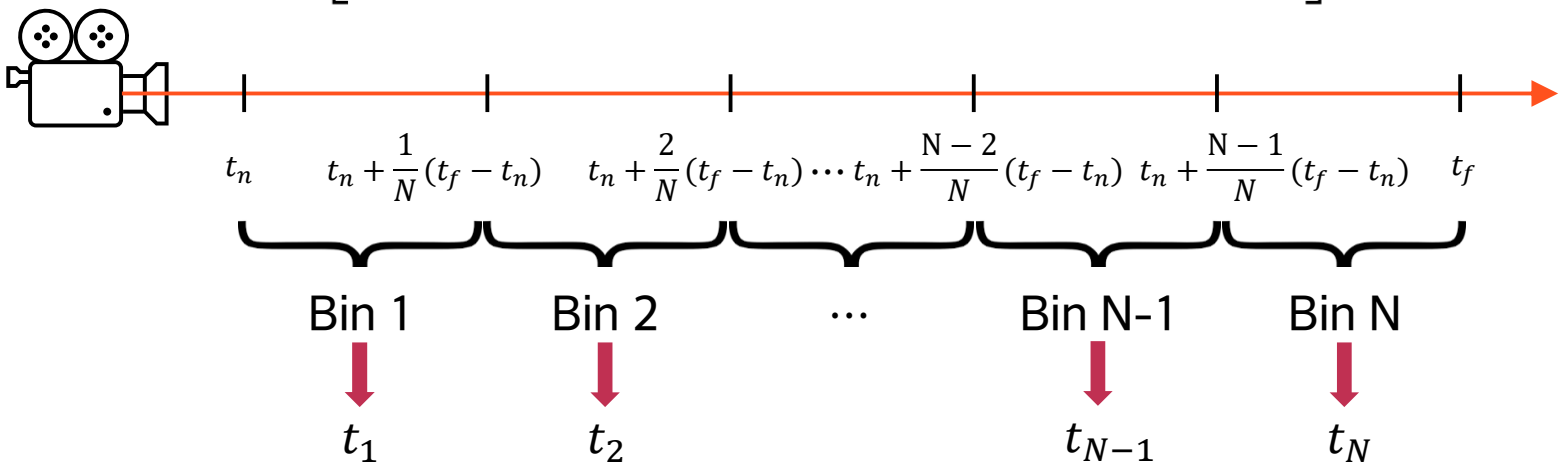
Stratified Sampling

$$t_i \sim \mathcal{U}\left[t_n + \frac{i-1}{N}(t_f - t_n),\ t_n + \frac{i}{N}(t_f - t_n)\right]$$

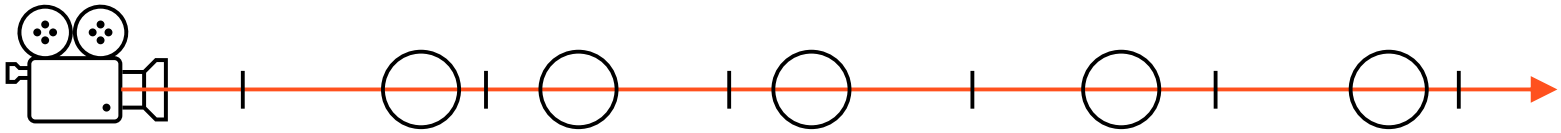
Method

- **Hierarchical Volume Sampling**
  - Stratified Sampling (1차 Sampling)

$$t_i \sim \mathcal{U}\left[t_n + \frac{i-1}{N}(t_f - t_n),\ t_n + \frac{i}{N}(t_f - t_n)\right]$$



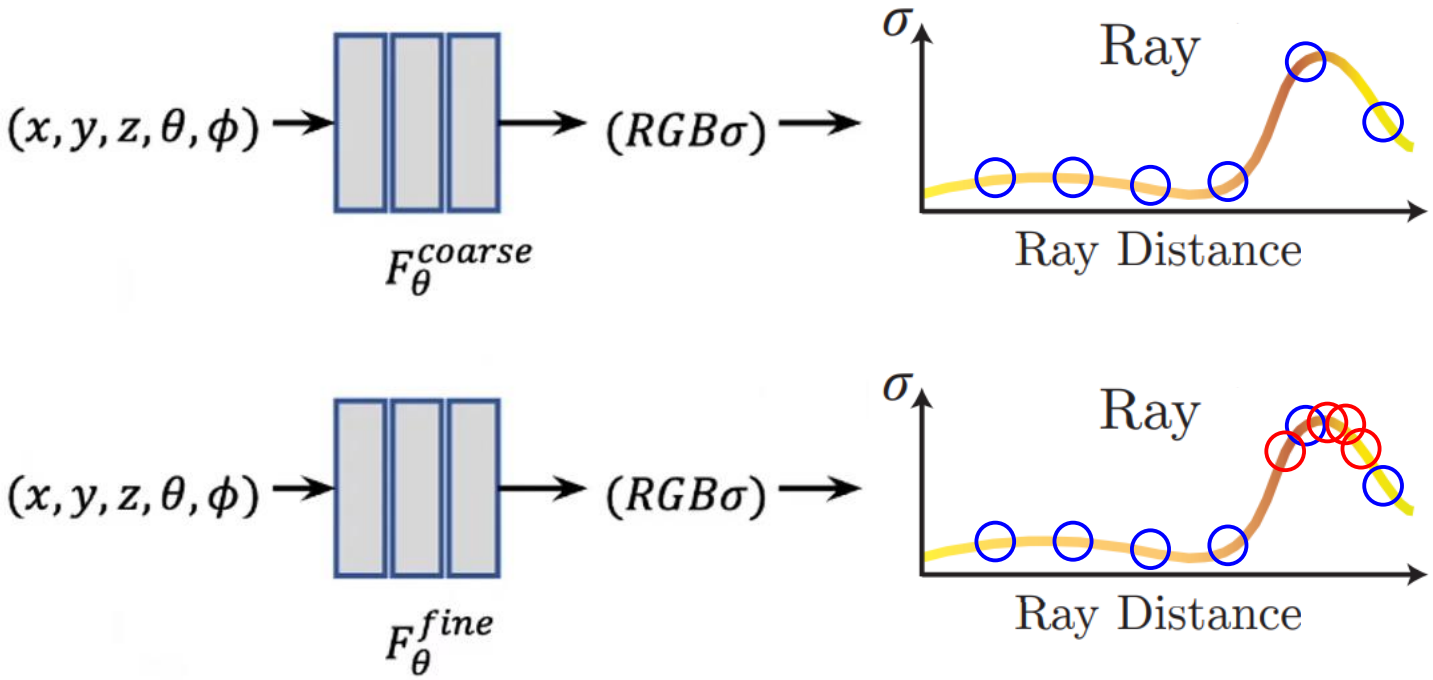
N개 Sample 추출



N_c Samples → Coarse Network

Method

- **Hierarchical Volume Sampling**
  - Fine Sampling (2차 Sampling)

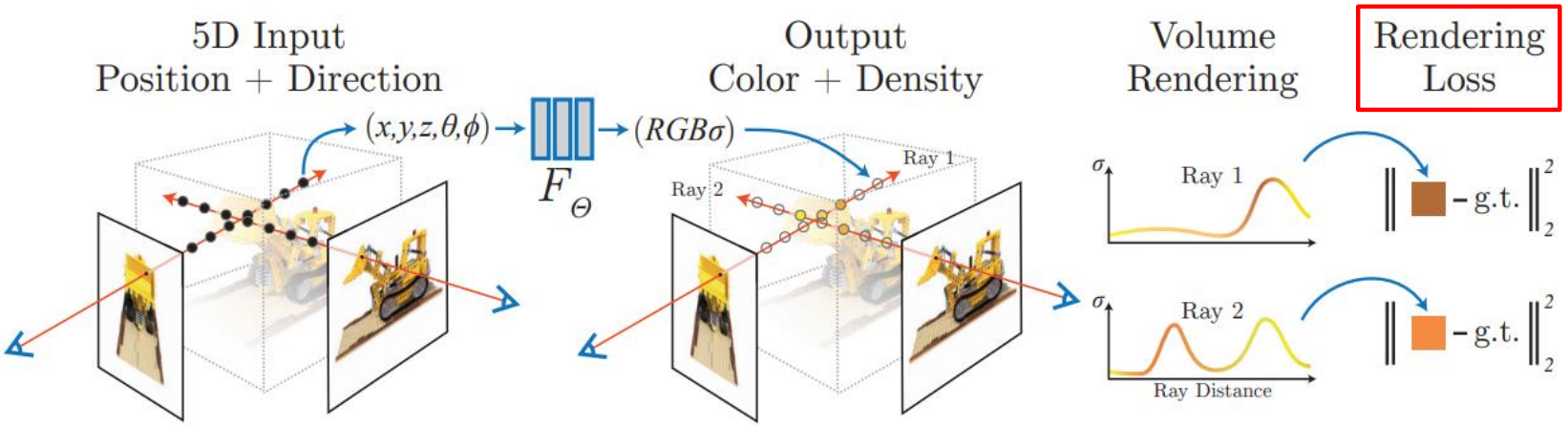


- w_i의 확률분포(PDF)로부터 w_i가 큰 구간에 대해 집중적으로 N_f개의 sample들을 추출
- $(N_f + N_f)$ Samples → Fine Network

$$N_c = 64\,, N_f = 128$$

Method

- **Overall Pipeline**



1. Camera에서 scene을 향해 view direction(θ, ϕ) 방향으로 ray를 쏜다. 즉 , camera로 찍은 2D image의 각 pixel 마다 하나의 ray를 쏘는 것. (각 ray는 하나의 pixel에 대응됨) 이때 ray위에 3D points(voxels)들을 sampling한다.

2. MLP의 input으로 각 3D point의 위치 $\mathbf{x} = (x, y, z)$와 viewing direction(θ, ϕ)이 들어가고 output으로 emitted radiance $\mathbf{c} = (r, g, b)$ volume density σ가 출력된다.

3. Weighted sum을 이용하여 2D pixel의 color를 계산한다. (volume rendering)

4. Loss를 계산하여 MLP network를 optimize한다.

Method

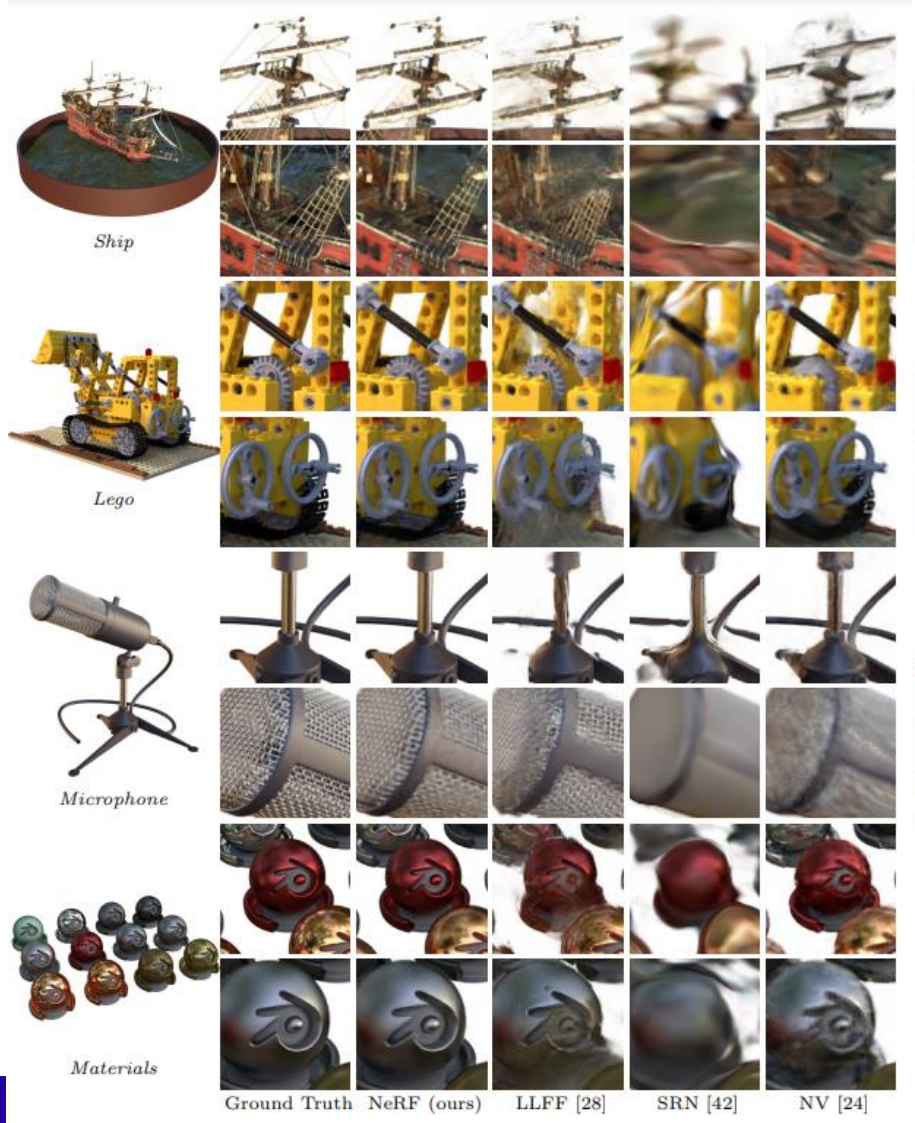
- **Loss for Training MLP**

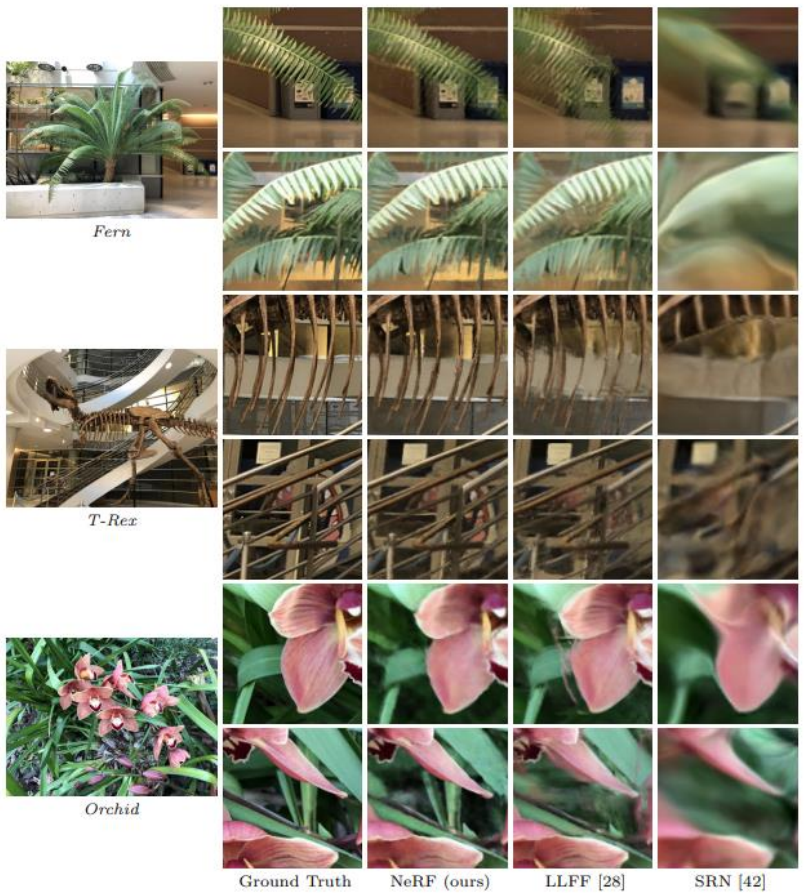
$$\mathcal{L} = \sum_{\mathbf{r}\in\mathcal{R}} \left[\left\| \hat{C}_c(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 + \left\| \hat{C}_f(\mathbf{r}) - C(\mathbf{r}) \right\|_2^2 \right]$$

- $\mathcal{R}$: 각 batch 내 ray들의 집합
- $C(\mathbf{r})$: Ground Truth
- $\hat{C}_c(\mathbf{r})$: Coarse Network를 통해 예측된 ray $\mathbf{r}$에 대응하는 pixel의 RGB color
- $\hat{C}_f(\mathbf{r})$: Fine Network를 통해 예측된 ray $\mathbf{r}$에 대응하는 pixel의 RGB color
- Loss $= \sum$ ($\hat{C}_c(\mathbf{r})$의 L2 loss $+$ $\hat{C}_f(\mathbf{r})$의 L2 loss)
- Coarse Network의 loss까지 최종 Loss에 포함시킴으로써 잘 우수한 sample들이 Fine Network에 입력되도록 설계

Results

- Comparison with Other Models

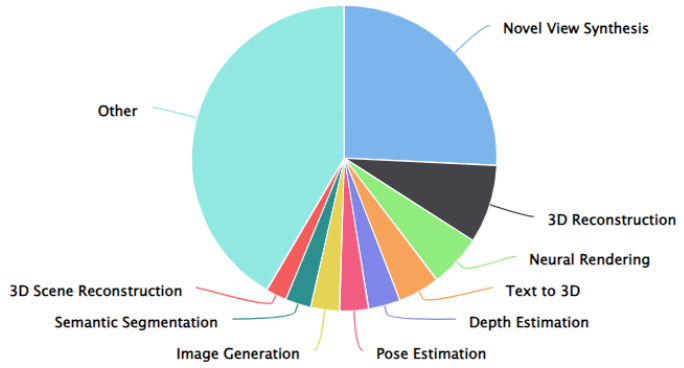




PSNR↑, SSIM↑, LPIPS↓

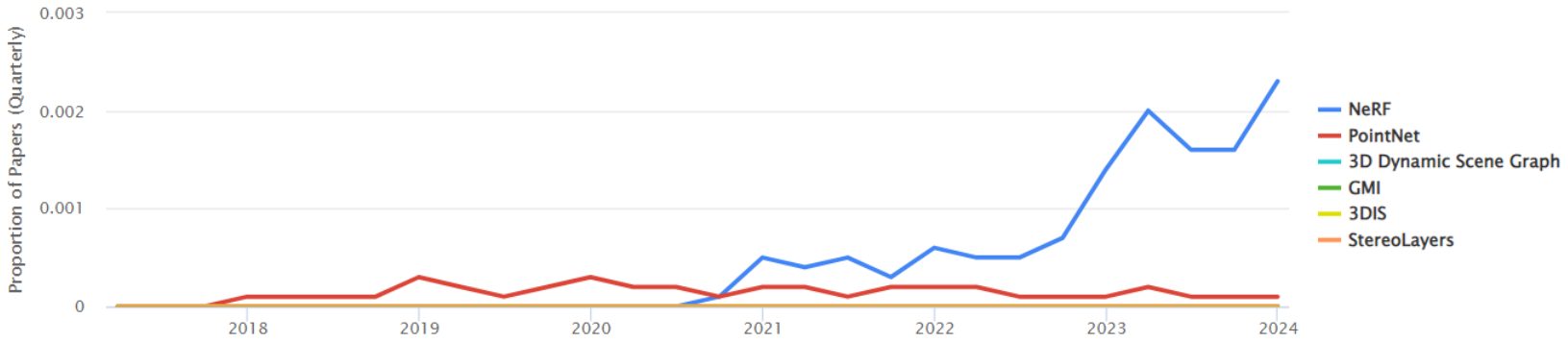
Application & Follow-up Study

Tasks



Task	Papers	Share
Novel View Synthesis	152	25.76%
3D Reconstruction	49	8.31%
Neural Rendering	33	5.59%
Text to 3D	26	4.41%
Depth Estimation	20	3.39%
Pose Estimation	18	3.05%
Image Generation	18	3.05%
Semantic Segmentation	16	2.71%
3D Scene Reconstruction	13	2.20%

Usage Over Time



3D Reconstruction

