2023-1학기 종합설계 프로젝트

2D-to-3D 영상 변환을 위한 NeRF:Neural Radiance Fields 기술 구현 및 개선

Hong Boyoung, Bae Hyunji, Lee Juhyeon

Department of Multimedia Engineering

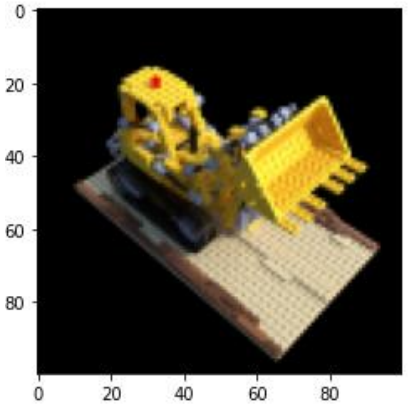
dongguk UNIVERSITY

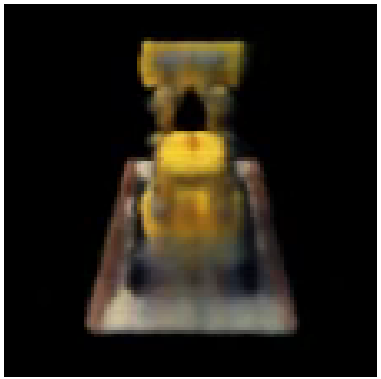
Contents

Introduction

- 단일 카메라로 촬영한 2D 영상을 이용하여 개략적인 3D 영상을 추정하는 인공지능 기술

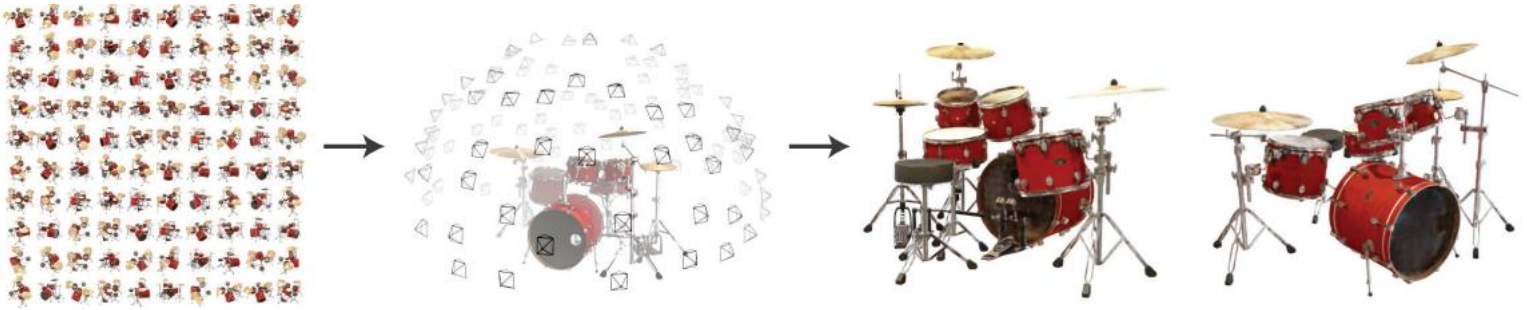
[1]





[1] https://github.com/bmild/nerf.git

[2] NeRF: Neural Radiance Fields



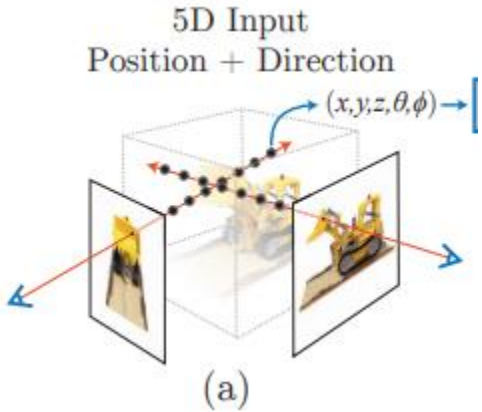
input 이미지 최적화된 NeRF 새로운 방향에서의 이미지

[2] Mildenhall, Ben, et al. "Nerf: Representing scenes as neural radiance fields for view synthesis." *Communications of the ACM* 65.1 (2021): 99-106.

[2]

NeRF: Neural Radiance Fields

- **Ray**
  물체를 찍은 방향으로 부터 물체를 향하도록 일직선으로 쏜 선

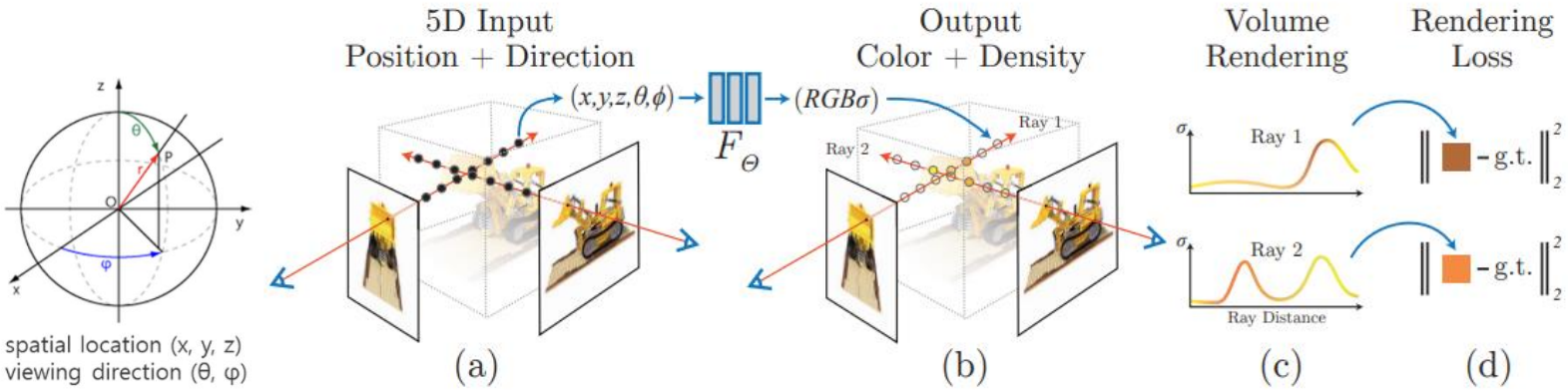


[2] Mildenhall, Ben, et al. "Nerf: Representing scenes as neural radiance fields for view synthesis." *Communications of the ACM* 65.1 (2021): 99-106.

[2]

NeRF: Neural Radiance Fields

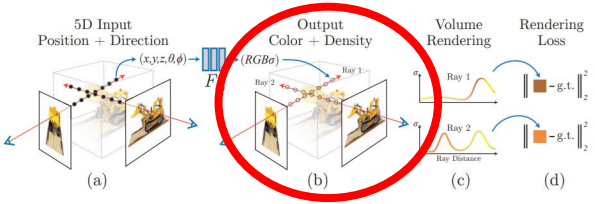
- **input (ray직선 내에 포함되는 point들의 3d좌표값(x,y,z), ray의 방향을 나타내는viewing direction(θ, φ))**
- **output (color(r,g,b), density)**



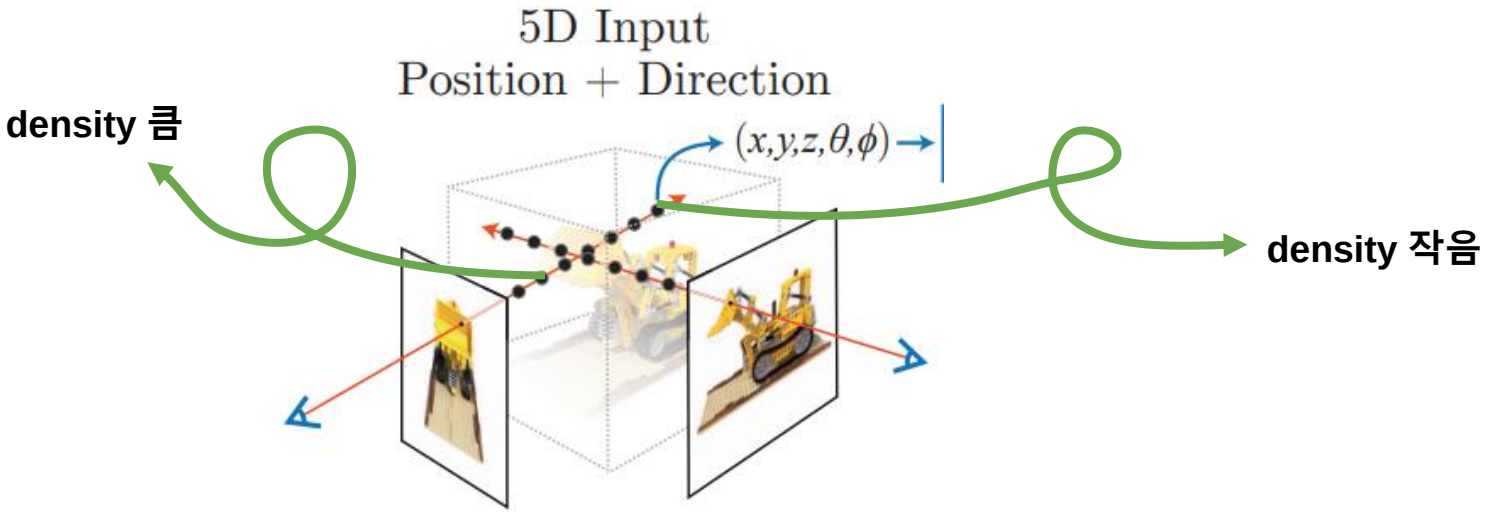
[2] Mildenhall, Ben, et al. "Nerf: Representing scenes as neural radiance fields for view synthesis." *Communications of the ACM* 65.1 (2021): 99-106.

[2]

NeRF: Neural Radiance Fields



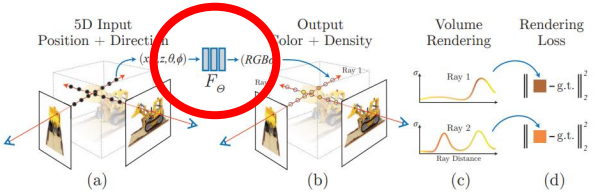
- **density 사용 이유**



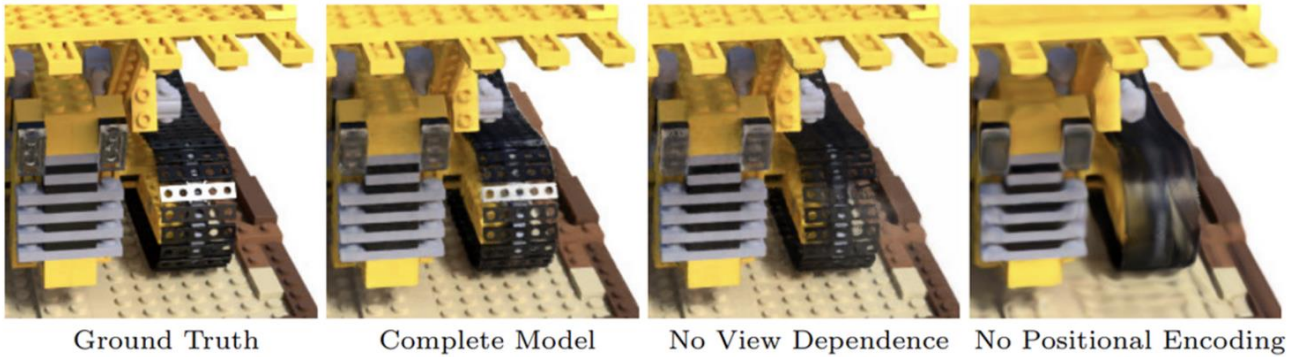
[2] Mildenhall, Ben, et al. "Nerf: Representing scenes as neural radiance fields for view synthesis." *Communications of the ACM* 65.1 (2021): 99-106.

[2]

NeRF: Neural Radiance Fields



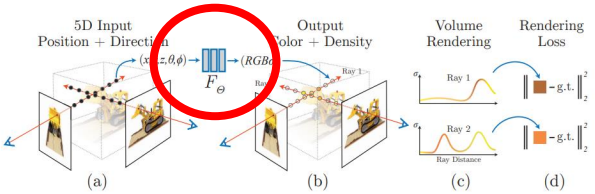
- **Positional Encoding**



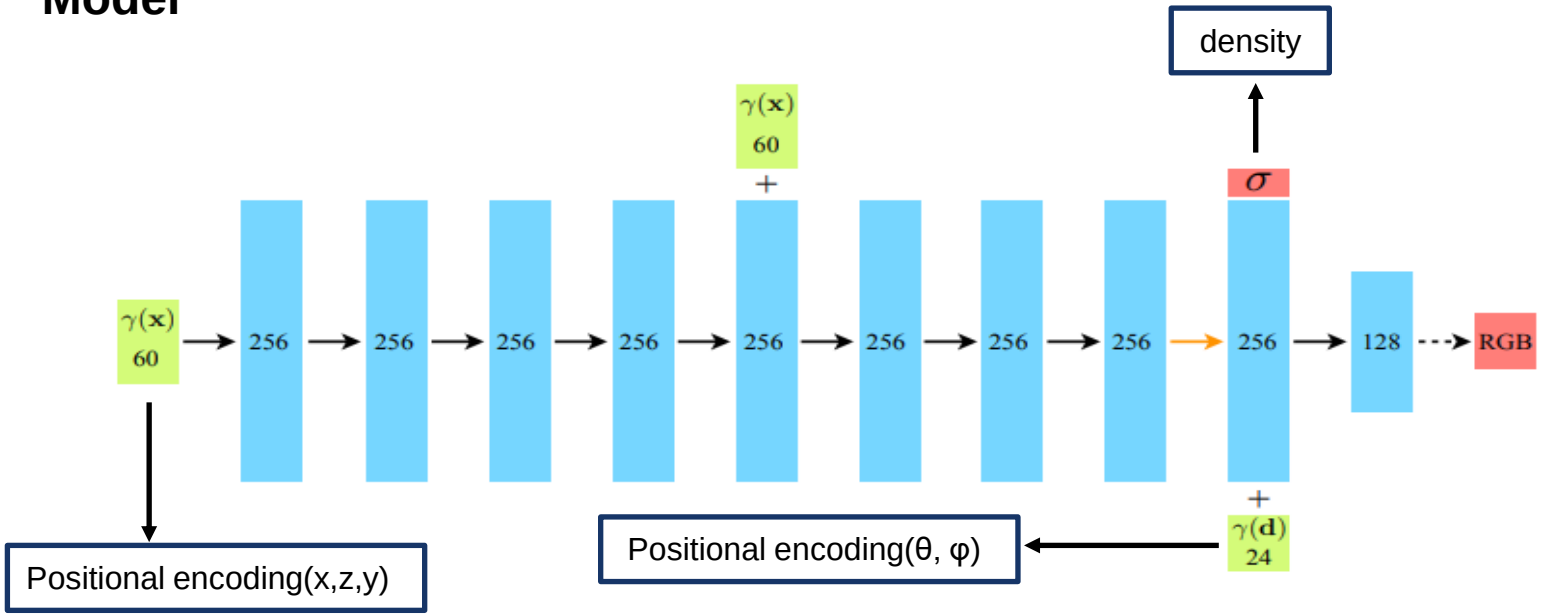
[2] Mildenhall, Ben, et al. "Nerf: Representing scenes as neural radiance fields for view synthesis." *Communications of the ACM* 65.1 (2021): 99-106.

[2]

NeRF: Neural Radiance Fields

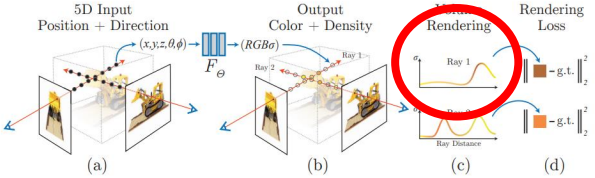


- **Model**



[2] Mildenhall, Ben, et al. "Nerf: Representing scenes as neural radiance fields for view synthesis." *Communications of the ACM* 65.1 (2021): 99-106.

NeRF: Neural Radiance Fields

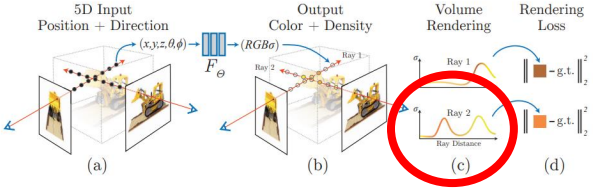


- **Hierarchical Volume Rendering(Coarse Model)**

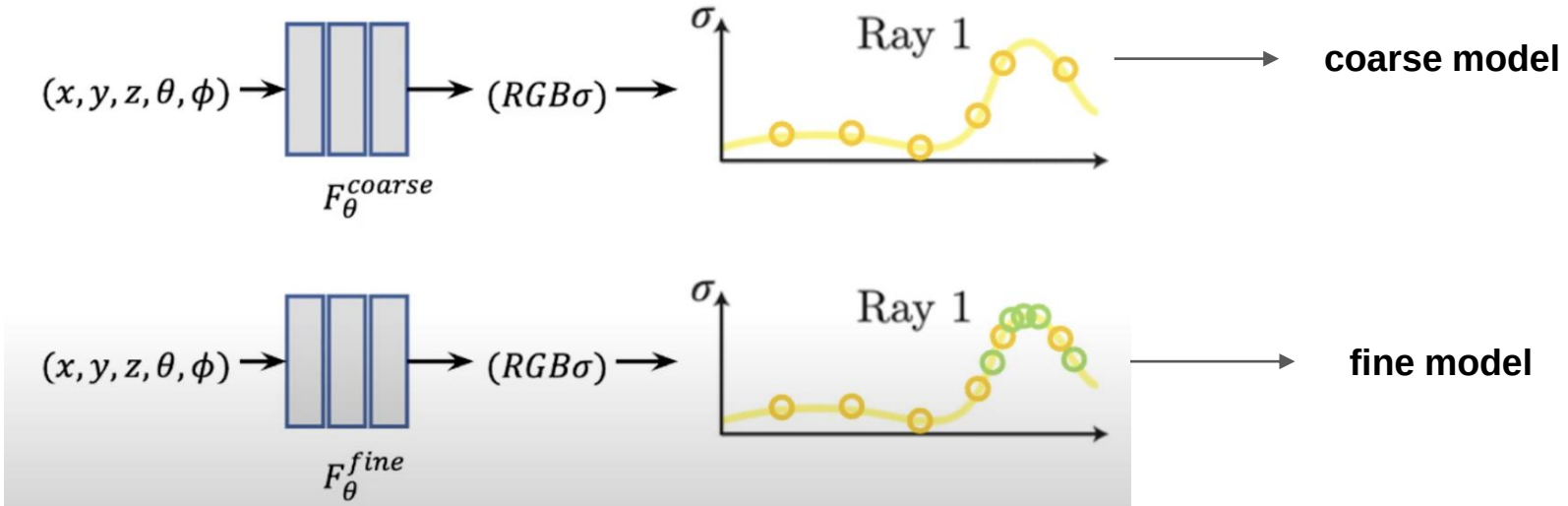
Stratified sampling

Deterministic sampling

NeRF: Neural Radiance Fields

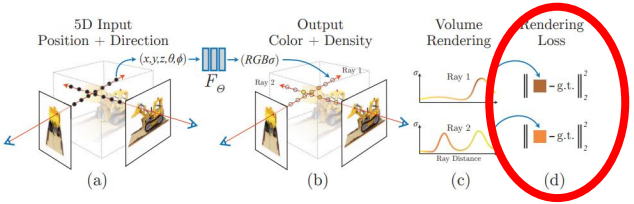


- **Hierarchical Volume Rendering(Fine Model)**



[2]

NeRF: Neural Radiance Fields



- **Loss Function**

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

[2] Mildenhall, Ben, et al. "Nerf: Representing scenes as neural radiance fields for view synthesis." *Communications of the ACM* 65.1 (2021): 99-106.

Progress 1

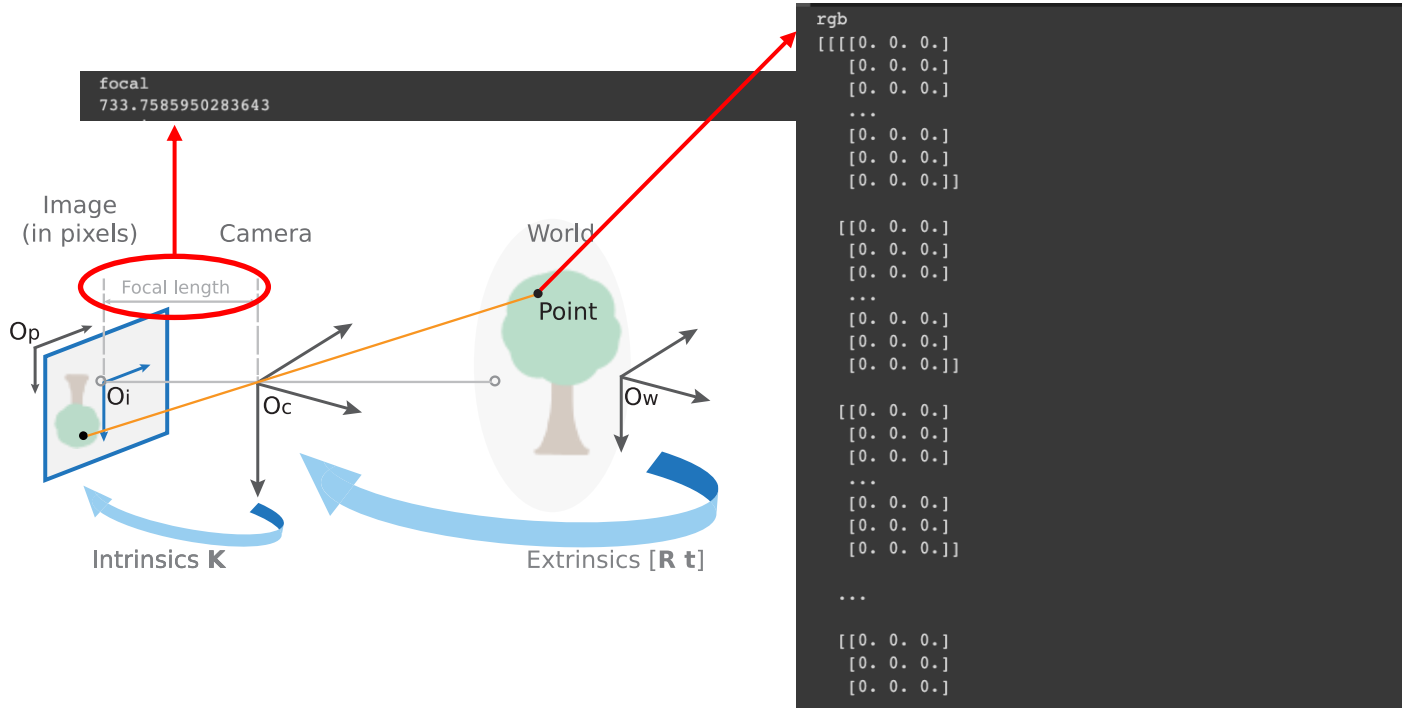
NeRF: Neural Radiance Fields

NeRF input data(.npz)

```
# Load dataset.
data_f = "/content/drive/MyDrive/ss.npz"
data = np.load(data_f)
```

NeRF: Neural Radiance Fields

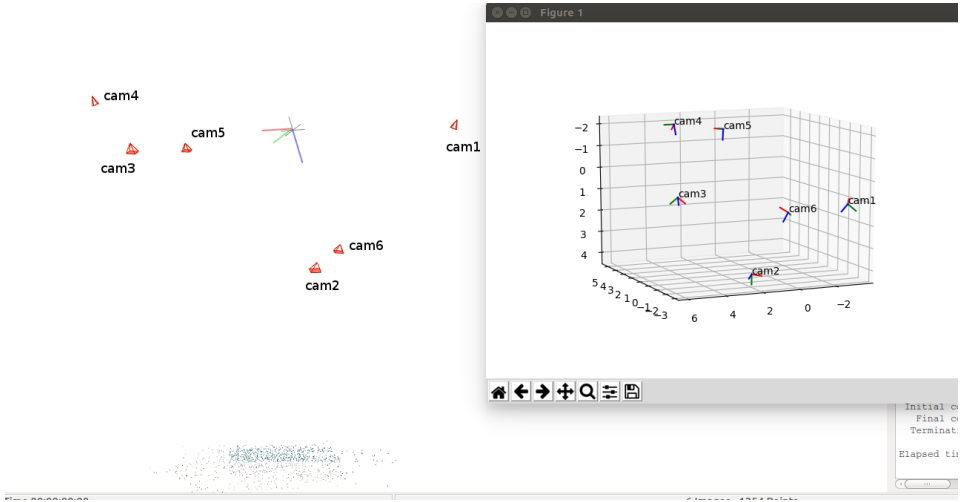
NeRF input data(.npz) 구성



NeRF: Neural Radiance Fields

NeRF input data(.npz) 구성

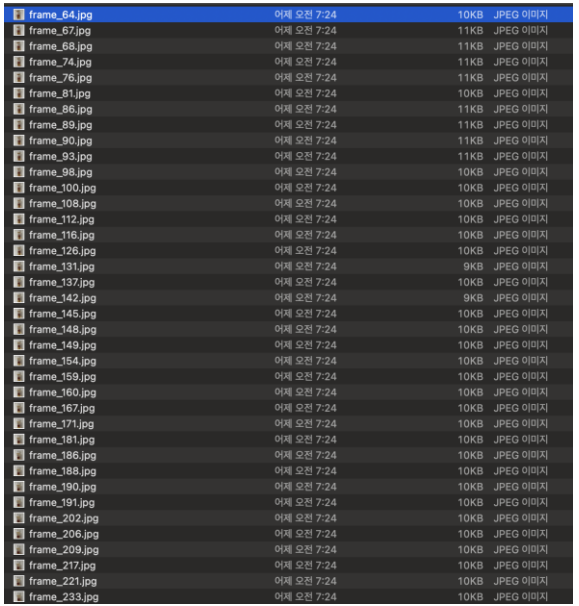
```
poses
[[[-4.35209662e-01  2.95908600e-02 -8.99842739e-01  4.55511427e+00]
  [-1.48623556e-01  9.83386636e-01  1.04220055e-01 -1.56800270e-01]
  [ 8.87977242e-01  1.79095402e-01 -4.23581451e-01  4.03380823e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00]]

 [[-4.19371516e-01  2.77612247e-02 -9.07390118e-01  4.50792456e+00]
  [-1.51415825e-01  9.83392000e-01  1.00066826e-01 -1.01143152e-01]
  [ 8.95098209e-01  1.79358393e-01 -4.08203095e-01  3.84834123e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00]]

 [[-4.08392727e-01  2.76310127e-02 -9.12388027e-01  4.46232557e+00]
  [-1.47633374e-01  9.84382451e-01  9.58932862e-02 -1.01978794e-01]
  [ 9.00788426e-01  1.73861042e-01 -3.97935390e-01  3.89384055e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00]]

 ...

 [[ 9.62665856e-01 -8.64555314e-02  2.56514996e-01 -3.35651088e+00]
  [ 8.40836316e-02  9.96253490e-01  2.02217642e-02 -4.09667283e-01]
  [-2.57302225e-01  2.10191077e-03  9.66328681e-01  3.62111068e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00]]

 [[ 9.62440193e-01 -9.01462957e-02  2.56090939e-01 -3.39982367e+00]
  [ 8.76862630e-02  9.95926082e-01  2.10326016e-02 -4.18532610e-01]
  [-2.56943643e-01  2.21303641e-03  9.66423869e-01  3.77356005e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00]]

 [[ 9.60736275e-01 -9.43725407e-02  2.60920644e-01 -3.46999192e+00]
  [ 8.95449966e-02  9.95520055e-01  3.03564407e-02 -4.38342005e-01]
  [-2.62616545e-01 -5.80039620e-03  9.64882851e-01  3.54331017e+00]
  [ 0.00000000e+00  0.00000000e+00  0.00000000e+00  1.00000000e+00]]]
```



Progress1

1. input data 만들기

누룽지 모델에 대한 영상 촬영

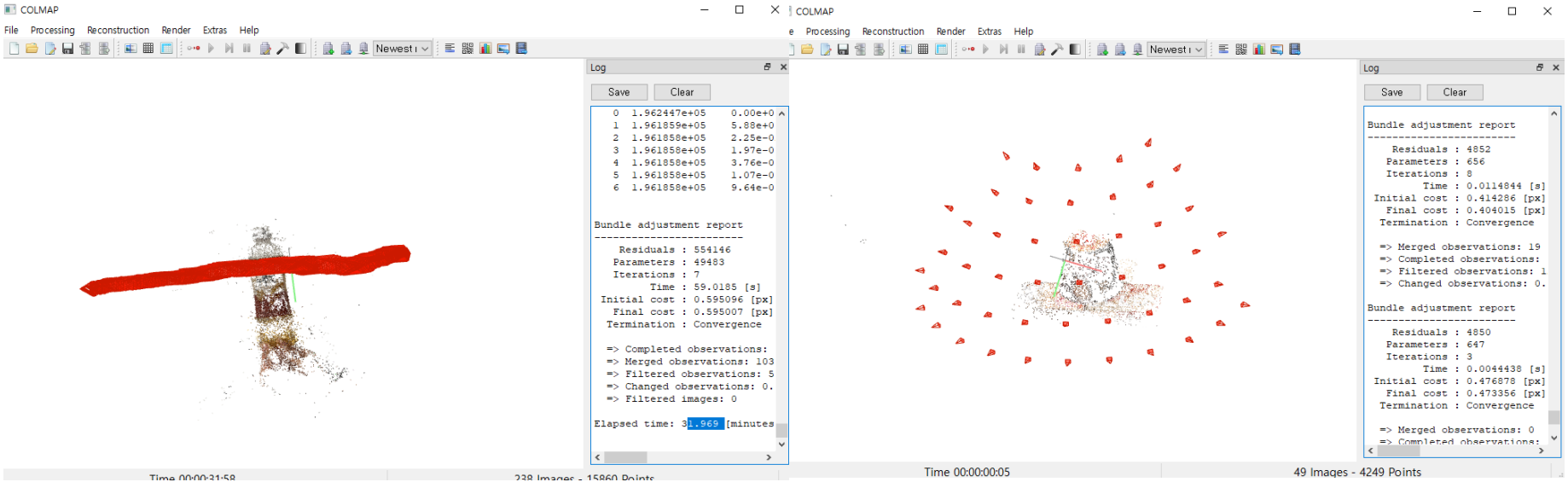


촬영한 영상에서 frame단위로
이미지 파일을 추출함(42개)

Progress1

2. 데이터 전처리

1) colmap 사용 → 결과 : QW, QX, QY, QZ, tx, ty, tz, focal length
3D Point(X, Y, Z), RGB value



Progress1

2. 데이터 전처리

2) python이용하여 필요한 데이터 추출 - normalized_array(RGB) – colmap의 RGB값 이용

```
[ ]  import math

     images = sorted(glob.glob('/content/nooo/*.jpg'))
     chunk_size = 50
     num_chunks = int(math.ceil(len(images) / float(chunk_size)

     for i in range(num_chunks):
         start_idx = i * chunk_size
         end_idx = min((i + 1) * chunk_size, len(images))
         chunk = images[start_idx:end_idx]
         rgb_array = []
         for fname in chunk:
             img = cv2.imread(fname)
             rgb = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
             rgb_array.append(rgb)
         normalized_array = np.array(rgb_array)/255.0
         # process normalized_array here
```

```
[ ]  normalized_array.shape

     (42, 200, 200, 3)
```

Progress1

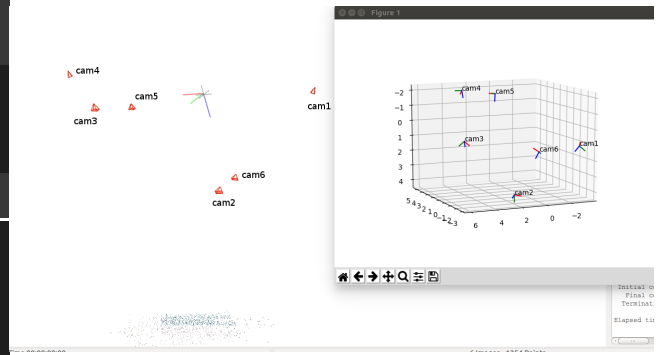
2. 데이터 전처리

2) python이용하여 필요한 데이터 추출 - poses – colmap의 QW, QX, QY, QZ, tx, ty, tz값 이용

->위의 정제된 데이터에서 qw,qx,qy,qz,tx,ty,tx 만 따로 빼오기

```
[ ]  final_images_data = images_df.loc[:,['QW','QX','QY','QZ','TX','TY','TZ']]
     numpy_images_data = images_df[['QW','QX','QY','QZ','TX','TY','TZ']].values.tolist()
     print(np.array(numpy_images_data).shape)

     (42, 7)
```

```
# 회전 쿼터니언으로부터 회전 행렬을 구함
R = []
for i in range(num):
  R.append(np.array([[1-2*qy[i]**2-2*qz[i]**2, 2*qx[i]*qy[i]-2*qz[i]*qw[i], 2*qx[i]*qz[i]+2*qy[i]*qw[i]],
                [2*qx[i]*qy[i]+2*qz[i]*qw[i], 1-2*qx[i]**2-2*qz[i]**2, 2*qy[i]*qz[i]-2*qx[i]*qw[i]],
                [2*qx[i]*qz[i]-2*qy[i]*qw[i], 2*qy[i]*qz[i]+2*qx[i]*qw[i], 1-2*qx[i]**2-2*qy[i]**2]]))

# 변환 행렬을 구함
T = []
P = []
for i in range(num):
  T.append(np.array([tx[i], ty[i], tz[i]]).reshape(-1, 1))
for i in range(num):
  P.append(np.hstack([R[i], T[i]]))
for i in range(num):
  P[i] = np.vstack([P[i], [0.0, 0.0, 0.0, 1.0]])
```



$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \qquad \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{11} & r_{12} & r_{13} & t_x \\ r_{11} & r_{12} & r_{13} & t_x \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = [R|t] \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Progress1

2. 데이터 전처리

2) python이용하여 필요한 데이터 추출 - focal length – colmap의 focal length값 이용

```
camera_data_split = [x.strip().split() for x in camera_data[3:]]
refine_camera_data = []
for i in range(len(camera_data_split)):
  refine_camera_data.append(camera_data_split[i][:5])
print(refine_camera_data)
total = 0
for row in refine_camera_data:
    total += float(row[4])

# 평균을 계산합니다.
focal_length = total / len(refine_camera_data)
print(focal_length )
```

```
[['2', 'SIMPLE_RADIAL', '200', '200', '365.44184091808307'], ['3',
411.49892339772384
```

3) 추출한 모든 이미지에 대한 RGB값과, pose 행렬, focal length를 하나의 npz 파일로 저장

```
np.savez('nooo.npz',normalized_array=normalized_array, poses=poses, focal=focal_length)
```

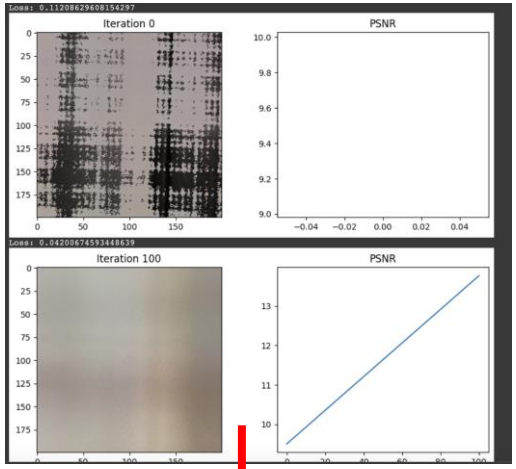
Progress1

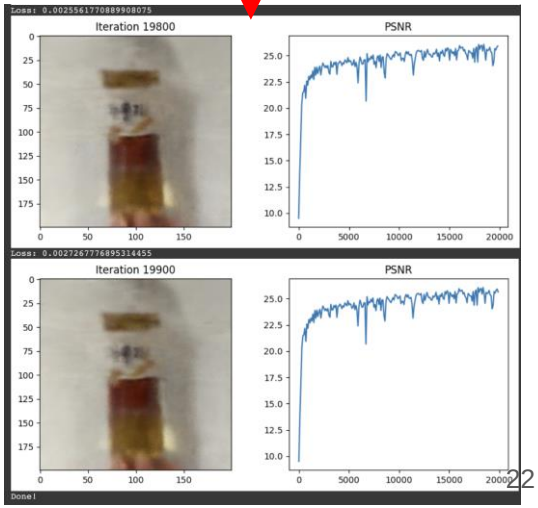
3. training

```
seed = 9458
torch.manual_seed(seed)
np.random.seed(seed)

device = "cuda:0"
F_c = VeryTinyNeRFMLP().to(device)
chunk_size = 16384

lr = 5e-3
optimizer = optim.Adam(F_c.parameters(), lr=lr)
criterion = nn.MSELoss()

data_f = "/content/nooo.npz"
data = np.load(data_f)

images = data["normalized_array"] #/ 255
img_size = images.shape[1]
xs = torch.arange(img_size) - (img_size / 2 - 0.5)
ys = torch.arange(img_size) - (img_size / 2 - 0.5)
(xs, ys) = torch.meshgrid(xs, -ys, indexing="xy")
focal = float(data["focal"])
pixel_coords = torch.stack([xs, ys, torch.full_like(xs, -focal)], dim=-1)
camera_coords = pixel_coords / focal
init_ds = camera_coords.to(device)
init_o = torch.Tensor(np.array([0, 0, 2.25])).to(device)
print(images.shape)
test_idx = 0
plt.imshow(images[test_idx])
plt.show()
test_img = torch.Tensor(images[test_idx]).to(device)
poses = data["poses"]
test_R = torch.Tensor(poses[test_idx, :3, :3]).to(device)
test_ds = torch.einsum("ij,hwj->hwi", test_R, init_ds)
test_os = (test_R @ init_o).expand(test_ds.shape)


t_n = 1.0
t_f = 4.0
N_c = 32
t_i_c_gap = (t_f - t_n) / N_c
t_i_c_bin_edges = (t_n + torch.arange(N_c) * t_i_c_gap).to(device)

train_idxs = np.arange(len(images)) != test_idx
images = torch.Tensor(images[train_idxs])
poses = torch.Tensor(poses[train_idxs])
psnrs = []
iternums = []
num_iters = 20000
display_every = 100
F_c.train()
```

```
[3] F_c.train()

(42, 200, 200, 3)
```


```
VeryTinyNeRFMLP(
  (early_mlp): Sequential(
    (0): Linear(in_features=39, out_features=256, bias=True)
    (1): ReLU()
```





Progress1

4. 3D 동영상 렌더링

▲ Theta 방향 회전

▲ Phi방향 회전

Progress1

5. Limitation

1) 객체로 투명성이 있는 페트병을 선택하였기 때문에 density 값이 작음
2) 그림자가 있는 배경도 객체로 인식을 하기 때문에 객체만 추출하는 깔끔한 데이터 전처리가 필요함
3) 컴퓨터 성능으로 인하여 많은 iteration을 돌릴 수 없음 (NeRF논문 30 - 40만번)
4) 동영상 생성 시 회전 축 재설정 필요

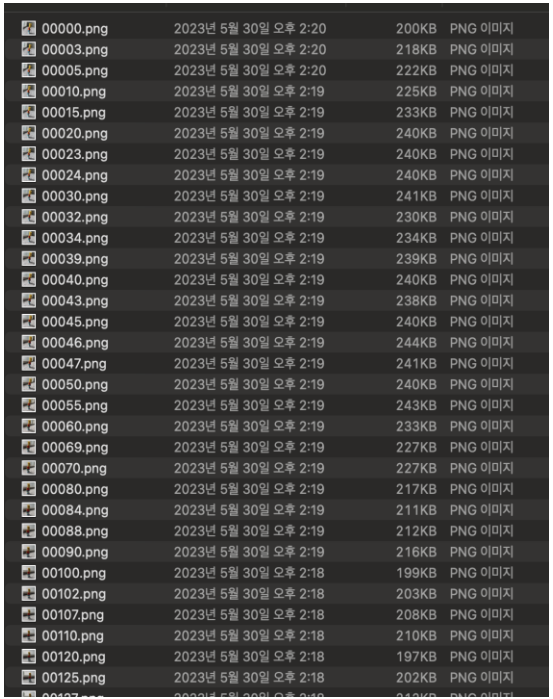
Progress 2

Progress2

1. Input data image 만들기

   -> density 값 높은 물체를 객체로 사용

심슨 모델에 대한 영상 촬영



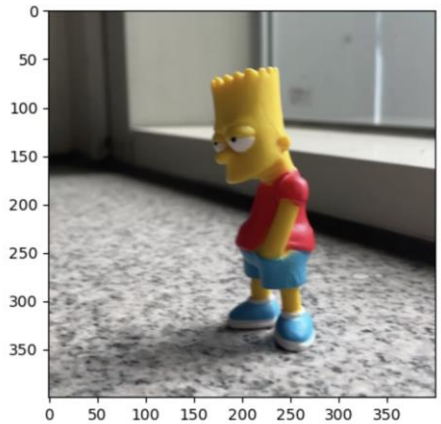
00000.png	2023년 5월 30일 오후 2:20	200KB	PNG 이미지
00003.png	2023년 5월 30일 오후 2:20	218KB	PNG 이미지
00005.png	2023년 5월 30일 오후 2:20	222KB	PNG 이미지
00010.png	2023년 5월 30일 오후 2:19	225KB	PNG 이미지
00015.png	2023년 5월 30일 오후 2:19	233KB	PNG 이미지
00020.png	2023년 5월 30일 오후 2:19	240KB	PNG 이미지
00023.png	2023년 5월 30일 오후 2:19	240KB	PNG 이미지
00024.png	2023년 5월 30일 오후 2:19	240KB	PNG 이미지
00030.png	2023년 5월 30일 오후 2:19	241KB	PNG 이미지
00032.png	2023년 5월 30일 오후 2:19	230KB	PNG 이미지
00034.png	2023년 5월 30일 오후 2:19	234KB	PNG 이미지
00039.png	2023년 5월 30일 오후 2:19	239KB	PNG 이미지
00040.png	2023년 5월 30일 오후 2:19	240KB	PNG 이미지
00043.png	2023년 5월 30일 오후 2:19	238KB	PNG 이미지
00045.png	2023년 5월 30일 오후 2:19	240KB	PNG 이미지
00046.png	2023년 5월 30일 오후 2:19	244KB	PNG 이미지
00047.png	2023년 5월 30일 오후 2:19	241KB	PNG 이미지
00050.png	2023년 5월 30일 오후 2:19	240KB	PNG 이미지
00055.png	2023년 5월 30일 오후 2:19	243KB	PNG 이미지
00060.png	2023년 5월 30일 오후 2:19	233KB	PNG 이미지
00069.png	2023년 5월 30일 오후 2:19	227KB	PNG 이미지
00070.png	2023년 5월 30일 오후 2:19	227KB	PNG 이미지
00080.png	2023년 5월 30일 오후 2:19	217KB	PNG 이미지
00084.png	2023년 5월 30일 오후 2:19	211KB	PNG 이미지
00088.png	2023년 5월 30일 오후 2:19	212KB	PNG 이미지
00090.png	2023년 5월 30일 오후 2:19	216KB	PNG 이미지
00100.png	2023년 5월 30일 오후 2:18	199KB	PNG 이미지
00102.png	2023년 5월 30일 오후 2:18	203KB	PNG 이미지
00107.png	2023년 5월 30일 오후 2:18	208KB	PNG 이미지
00110.png	2023년 5월 30일 오후 2:18	210KB	PNG 이미지
00120.png	2023년 5월 30일 오후 2:18	197KB	PNG 이미지
00125.png	2023년 5월 30일 오후 2:18	202KB	PNG 이미지

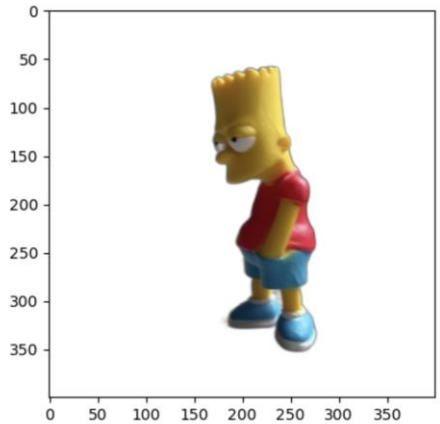
촬영한 영상에서 frame단위로
이미지 파일을 추출함(랜덤 50개)

Progress2

1. Input data image 만들기
   배경 제거 ‘rembg’ 패키지 설치
   -> 객체만 추출하는 깔끔한 데이터 전처리

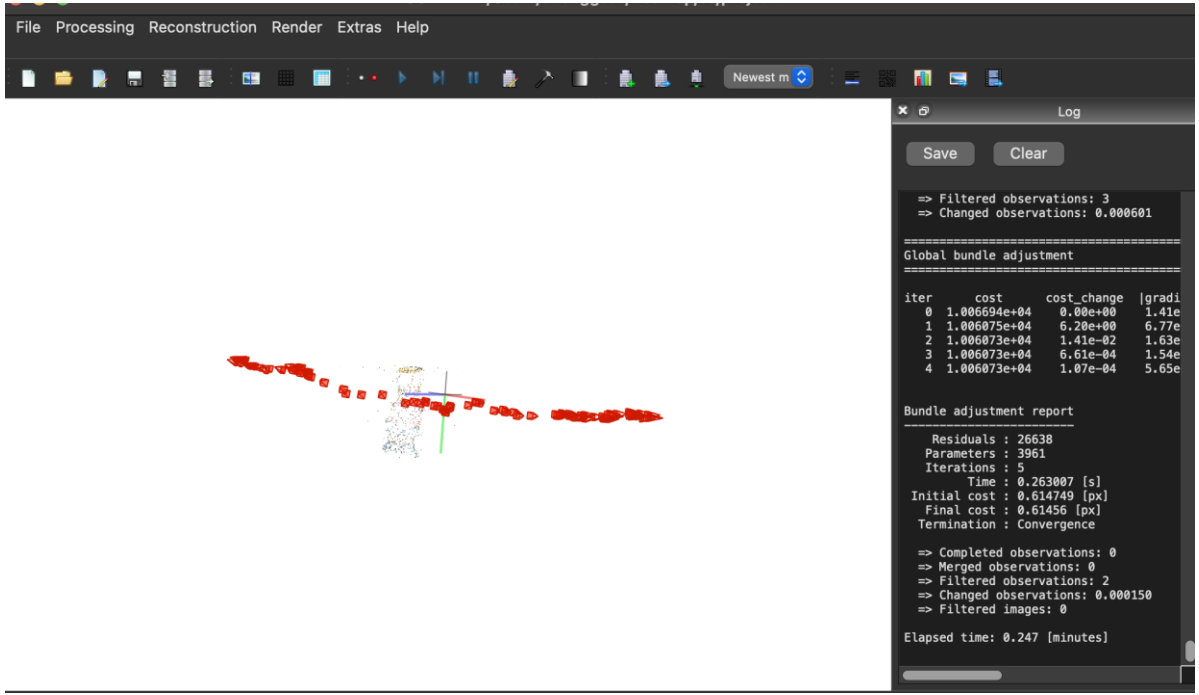
```
input_folder = glob.glob('/Users/ddonggssa/Desktop/simsim/*.png')
output_folder = "/Users/ddonggssa/Desktop/ss/"
for fi in input_folder:
    fig = plt.figure(figsize=(10, 10))
    # show original image
    fig.add_subplot(1, 2, 1)
    orig_img = Image.open(fi)
    plt.imshow(orig_img)

    # show bg removed image
    fig.add_subplot(1, 2, 2)
    with open(fi, "rb") as f:
        img_data = f.read()

    started = time.time()
    result = remove_bg(img_data)
    elapsed = time.time() - started
    print(f'it takes {elapsed} seconds for removing bg.')

    img = Image.open(io.BytesIO(result)).convert("RGBA")
    #plt.imshow(img)

    filename = os.path.basename(fi)
    output_path = os.path.join(output_folder, filename)
    img.save(output_path)
    print(f'Result image saved at {output_path}')
```





Progress2

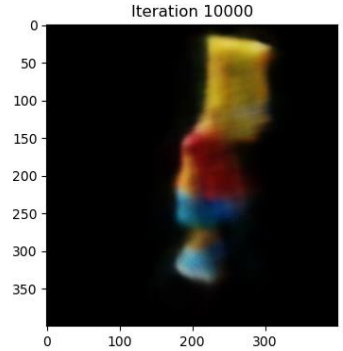
2. 데이터 전처리 및 npz파일 생성

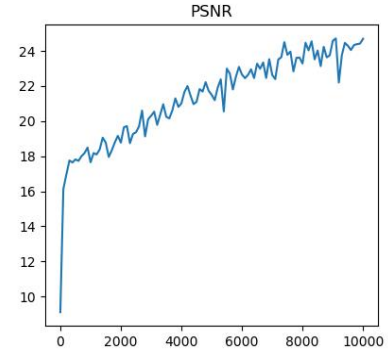


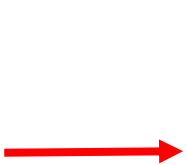
Progress2

3-1. iteration에 따른 성능 확인 -> 많은 iteration 실행 가능(15만번 1-2일 소요) & 회전축 재설정

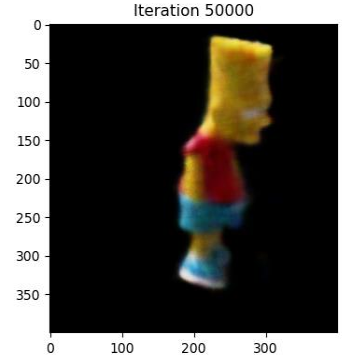
Loss: 0.0033950188662856817

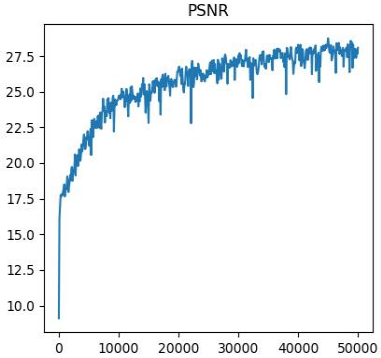


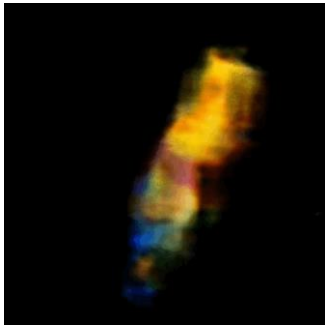




Loss: 0.0015571739058941603



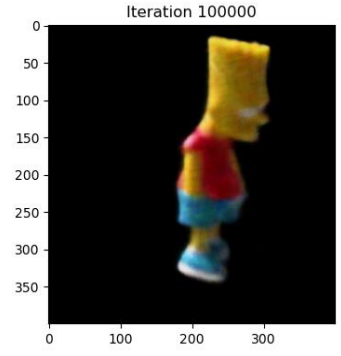


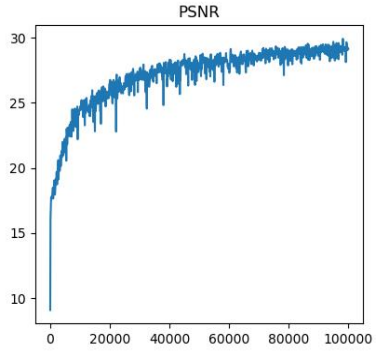


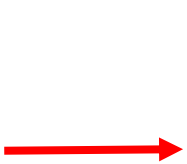
Progress2

3-1. iteration에 따른 성능 확인 -> 많은 iteration 실행 가능(15만번 1-2일 소요) & 회전축 재설정

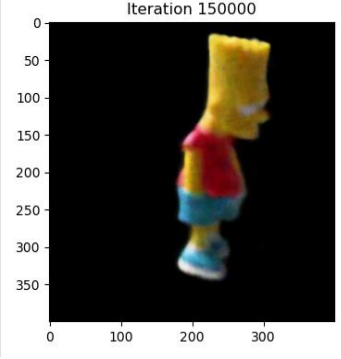
Loss: 0.0012174616567790508

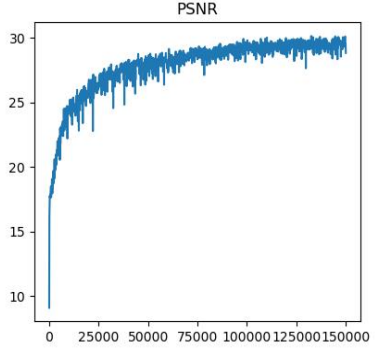






Loss: 0.0013129636645317078





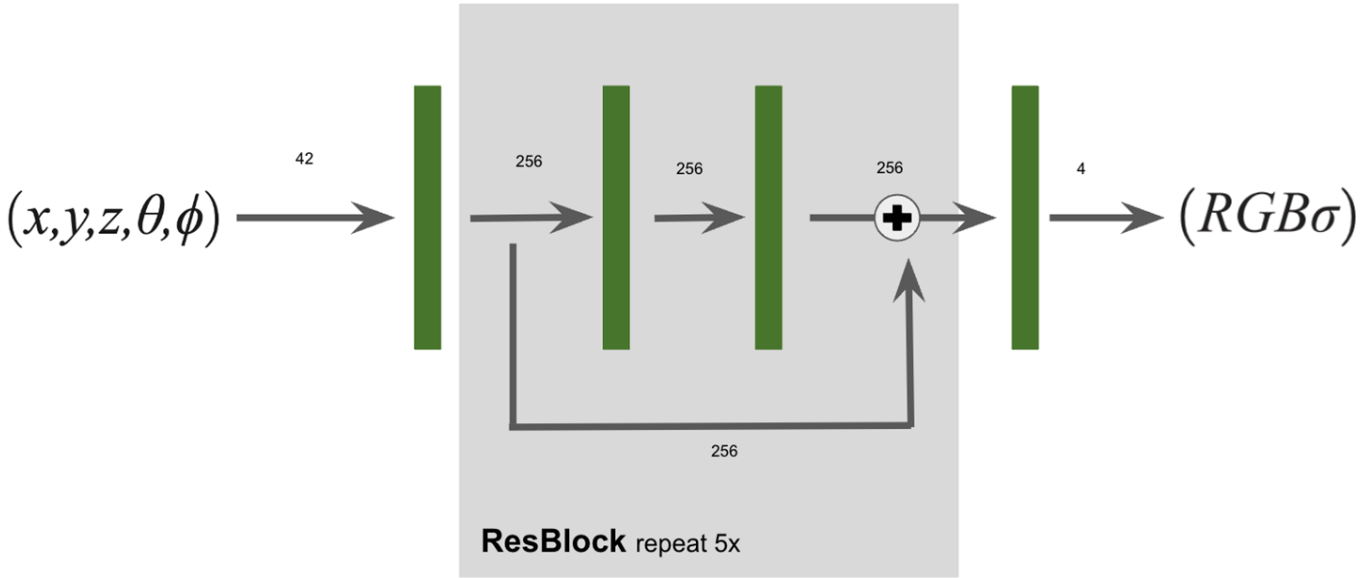
Progress2

3-1. Limitation

1) 객체의 노이즈가 심해, 이 부분을 제거해야 할 필요가 있음
2) 표면 정보가 잘 드러나지 않음

Progress2

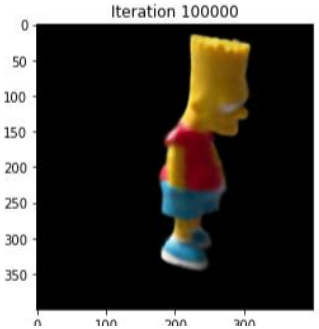
3-2. 네트워크 구조에 따른 성능 확인(MLP vs MLP + Residual block)

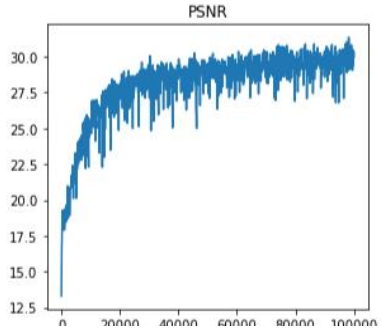


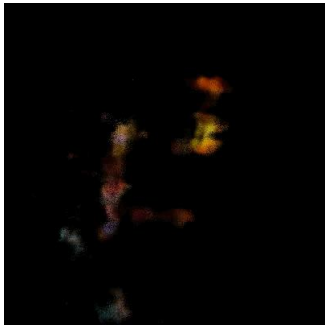
Progress2

3-2. training results

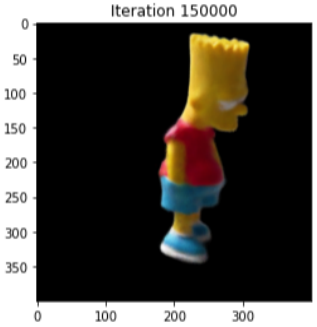
Loss: 0.0010062421206384897

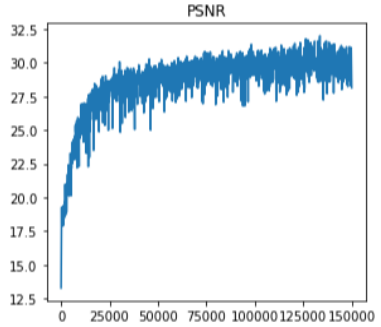


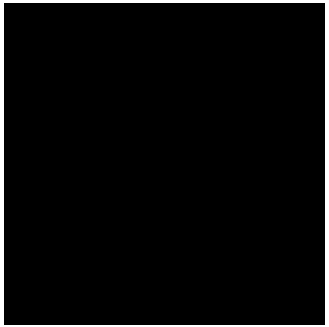




Loss: 0.0015340546378865838



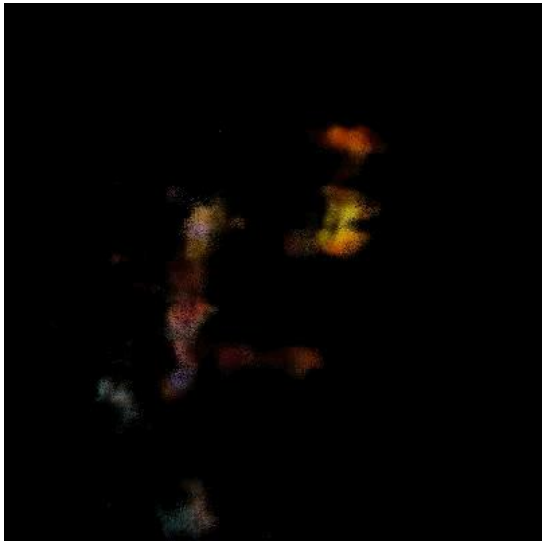




Progress2

3-2. training results(MLP vs MLP + Residual block)

▲ **MLP**



▲ **MLP + Residual block**

Thank you

dongguk UNIVERSITY