

Neural Radiance Field, Language Guidance and Robotic Manipulation

Peizhen Li

Faculty of Science and Engineering
Macquarie University

Dec 15, 2023

Outline

1 Neural Radiance Field & Feature Field Distillation

- Neural Radiance Field
- Distilled Feature Field

2 Language-Guided Robotic Manipulation

- Problem Formulation
- Represent and Infer 6-DOF Poses
- Open-Text Language-Guided Manipulation

3 Reflection & Future Work

- Reflection
- Future Work

Neural Radiance Field

NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis¹

- Input: (x, y, z, θ, ϕ)
- Output: (r, g, b, σ)
- MLP: $F_\theta : (x, y, z, \theta, \phi) \rightarrow (r, g, b, \sigma)$

What Can NeRF Representation Do?

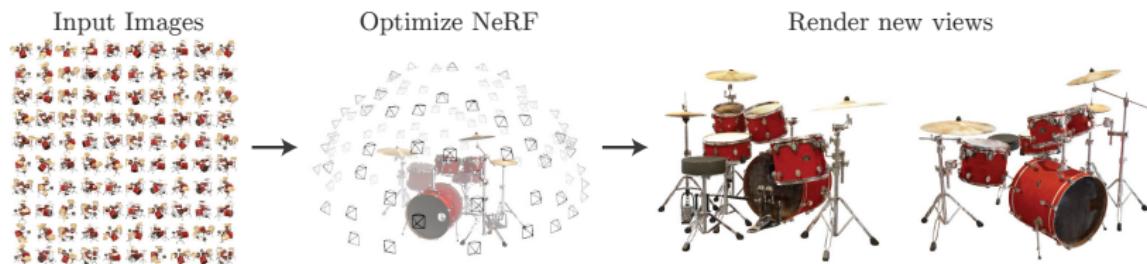
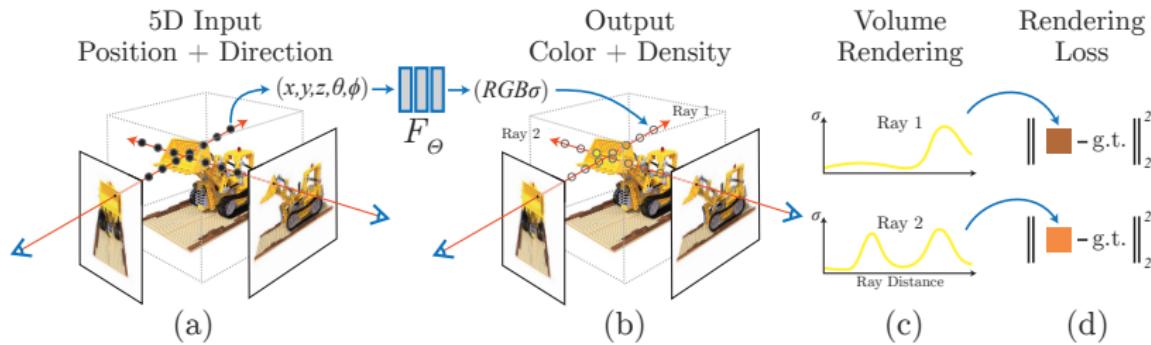


Figure: Optimize from a set of input images and render novel views

How to Optimize

- Given: a set of captured RGB images of the scene
- Volume Rendering: 5D radiance field → RGB images
- Loss: $\mathcal{L} = \sum_{\mathbf{r} \in \mathcal{R}} [\|\hat{C}_c(\mathbf{r}) - C(\mathbf{r})\|_2^2 + \|\hat{C}_f(\mathbf{r}) - C(\mathbf{r})\|_2^2]$

NeRF Overview



Volume Rendering with Radiance Fields

- NeRF output: $(\mathbf{c}(\mathbf{x}, \mathbf{d}), \sigma(\mathbf{x}))$
- Expected color:

$$C(\mathbf{r}) = \int_{t_n}^{t_f} T(t) \sigma(\mathbf{r}(t)) \mathbf{c}(\mathbf{r}(t), \mathbf{d}) dt \quad (1)$$

- Accumulated transmittance:

$$T(t) = \exp\left(- \int_{t_n}^t \sigma(\mathbf{r}(s)) ds\right) \quad (2)$$

Numerical Estimation

- Stratified sampling:

$$t_i \sim \mathcal{U}\left[t_n + \frac{i-1}{N}(t_f - t_n), \quad t_n + \frac{i}{N}(t_f - t_n)\right] \quad (3)$$

- Approximated color:

$$\hat{C}(\mathbf{r}) = \sum_{i=1}^N T_i (1 - \exp(-\sigma_i \delta_i)) \mathbf{c}_i \quad (4)$$

where

$$T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right), \quad \delta_i = t_{i+1} - t_i$$

Volume Rendering Digest

- Differential probability (reparameterization):

$$\sigma(\mathbf{x}) \rightarrow \sigma(t)$$

given $\mathbf{r} = (\mathbf{o}, \mathbf{d})$, $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$

- Transmittance:

$$T(t + dt) = T(t) \cdot (1 - dt \cdot \sigma(t)) \quad (5)$$

Transmittance

$$T(t + dt) = T(t) \cdot (1 - dt \cdot \sigma(t))$$
$$\frac{T(t + dt) - T(t)}{dt} \equiv T'(t) = -T(t)\sigma(t)$$

Solve this classical differential equation as follows:

$$T'(t) = -T(t)\sigma(t)$$
$$\int_a^b \frac{T'(t)}{T(t)} dt = - \int_a^b \sigma(t) dt$$
$$\log T(t) \Big|_a^b = - \int_a^b \sigma(t) dt \quad (6)$$
$$T(a \rightarrow b) \equiv \frac{T(b)}{T(a)} = \exp\left(- \int_a^b \sigma(t) dt\right)$$

Homogeneous Media

- constant \mathbf{c}_a, σ_a over a ray segment $[a, b]$

$$\begin{aligned}
 \mathbf{C}(a \rightarrow b) &= \int_a^b T(a \rightarrow t) \cdot \sigma(t) \cdot \mathbf{c}(t) \, dt \\
 &= \sigma_a \cdot \mathbf{c}_a \int_a^b T(a \rightarrow t) \, dt \\
 &= \sigma_a \cdot \mathbf{c}_a \int_a^b \exp\left(-\int_a^t \sigma(u) \, du\right) \, dt \\
 &= \sigma_a \cdot \mathbf{c}_a \int_a^b \exp(-\sigma_a(t-a)) \, dt \\
 &= \sigma_a \cdot \mathbf{c}_a \cdot \frac{\exp(-\sigma_a(t-a))}{-\sigma_a} \Big|_a^b \\
 &= \mathbf{c}_a \cdot (1 - \exp(-\sigma_a(b-a)))
 \end{aligned} \tag{7}$$

Transmittance Is Multiplicative

$$\begin{aligned} T(a \rightarrow c) &= \exp \left(- \int_a^c \sigma(t) dt \right) \\ &= \exp \left(- \left[\int_a^b \sigma(t) dt + \int_b^c \sigma(t) dt \right] \right) \tag{8} \\ &= \exp \left(- \int_a^b \sigma(t) dt \right) \exp \left(- \int_b^c \sigma(t) dt \right) \\ &= T(a \rightarrow b) \cdot T(b \rightarrow c) \end{aligned}$$

Transmittance for Piecewise Constant Data

- Given: $\{[t_i, t_{i+1}]\}_{i=1}^N$, $t_1 = 0$, $\delta_i = t_{i+1} - t_i$

- Constant: σ_i

$$\begin{aligned} T_i &= T(t_i) = T(0 \rightarrow t_i) = \exp \left(- \int_0^{t_i} \sigma(t) dt \right) \\ &= \exp \left(\sum_{j=1}^{i-1} -\sigma_j \delta_j \right) \end{aligned} \tag{9}$$

Volume Rendering of Piecewise Constant Data

$$\begin{aligned}\mathbf{C}(t_{N+1}) &= \sum_{i=1}^N \int_{t_i}^{t_{i+1}} T(t) \cdot \sigma_i \cdot \mathbf{c}_i \, dt \\ &= \sum_{i=1}^N \int_{t_i}^{t_{i+1}} T(0 \rightarrow t_i) \cdot T(t_i \rightarrow t) \cdot \sigma_i \cdot \mathbf{c}_i \, dt \\ &= \sum_{i=1}^N T(0 \rightarrow t_i) \int_{t_i}^{t_{i+1}} T(t_i \rightarrow t) \cdot \sigma_i \cdot \mathbf{c}_i \, dt \quad \text{constant} \\ &= \sum_{i=1}^N T(0 \rightarrow t_i) \cdot (1 - \exp(-\sigma_i(t_{i+1} - t_i))) \cdot \mathbf{c}_i \quad \text{from (7)}\end{aligned}$$

Volume Rendering from NeRF

$$\mathbf{C}(t_{N+1}) = \sum_{i=1}^N T_i \cdot (1 - \exp(-\sigma_i \delta_i)) \cdot \mathbf{c}_i, \quad (10)$$

where

$$T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right), \quad \delta_i = t_{i+1} - t_i$$

re-express:

$$\mathbf{C}(t_{N+1}) = \sum_{i=1}^N T_i \cdot \alpha_i \cdot \mathbf{c}_i \quad (11)$$

where

$$\alpha_i \equiv 1 - \exp(-\sigma_i \delta_i)$$

Volume Rendering in Practice

nerf demo

```
# Run network
pts_flat = tf.reshape(pts, [-1,3])
pts_flat = embed_fn(pts_flat)
raw = batchify(network_fn)(pts_flat)
raw = tf.reshape(raw, list(pts.shape[:-1]) + [4])

# Compute opacities and colors
sigma_a = tf.nn.relu(raw[...,:3])
rgb = tf.math.sigmoid(raw[...,:3])

# Do volume rendering
dists = tf.concat([z_vals[...,:1] - z_vals[...,:-1], tf.broadcast_to([1e10], z_vals[...,:1].shape)], -1)
alpha = 1.-tf.exp(-sigma_a * dists)
weights = alpha * tf.math.cumprod(1.-alpha + 1e-10, -1, exclusive=True)

rgb_map = tf.reduce_sum(weights[...,:None] * rgb, -2)
depth_map = tf.reduce_sum(weights * z_vals, -1)
acc_map = tf.reduce_sum(weights, -1)
```

NeRF Core Optimization Loop

```
with tf.GradientTape() as tape:  
    # Make predictions for color, disparity, accumulated opacity.  
    rgb, disp, acc, extras = render(  
        H, W, focal, chunk=args.chunk, rays=batch_rays,  
        verbose=i < 10, retraw=True, **render_kwargs_train)  
    # Compute MSE loss between predicted and true RGB.  
    img_loss = img2mse(rgb, target_s)  
    trans = extras['raw'][..., -1]  
    loss = img_loss  
    psnr = mse2psnr(img_loss) # peak signal to noise ratio  
    # Add MSE loss for coarse-grained model  
    if 'rgb0' in extras:  
        img_loss0 = img2mse(extras['rgb0'], target_s)  
        loss += img_loss0  
        psnr0 = mse2psnr(img_loss0)  
  
    gradients = tape.gradient(loss, grad_vars)  
    optimizer.apply_gradients(zip(gradients, grad_vars))
```

How Did NeRF and VLMs Get Married?

Decomposing NeRF for Editing via Feature Field Distillation²

- Editing a scene represented by a NeRF is challenging
- Semantic scene decomposition? 3D feature field
- Distill the knowledge of VLMs

Distilled Feature Field

- Maps from (\mathbf{x}, \mathbf{d}) to $(\mathbf{c}(\mathbf{x}, \mathbf{d}), \sigma(\mathbf{x}), \mathbf{f}(\mathbf{x}))$
- Volume Rendering

$$\hat{\mathbf{F}}(\mathbf{r}) = \sum_{k=1}^K \hat{T}(t_k) \alpha(\sigma_k \delta_k) \mathbf{f}(\mathbf{x}_k) \quad (12)$$

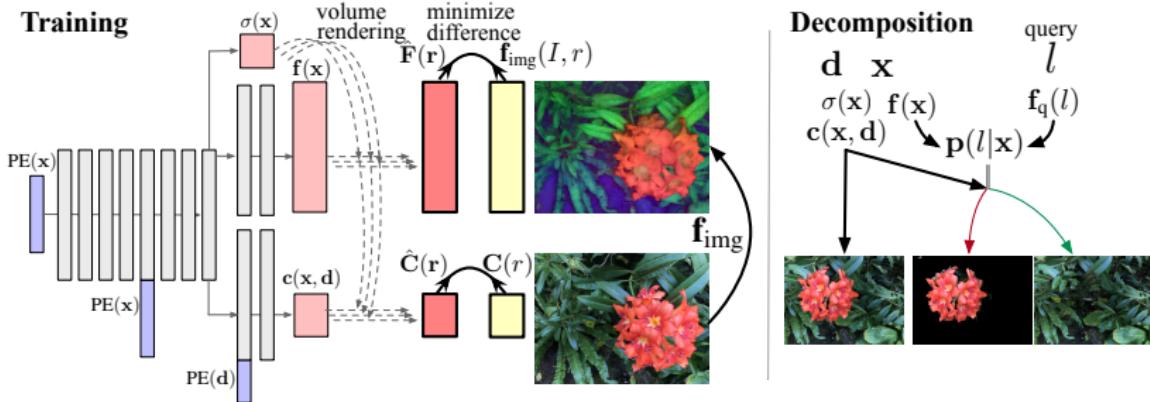
- Loss

$$L = L_p + \lambda L_f$$

$$L_p = \sum_{\mathbf{r} \in \mathcal{R}} \left\| \hat{\mathbf{C}}(\mathbf{r}) - \mathbf{C}(r) \right\|_2^2 \quad (13)$$

$$L_f = \sum_{\mathbf{r} \in \mathcal{R}} \left\| \hat{\mathbf{F}}(\mathbf{r}) - \mathbf{f}_{\text{img}}(I, r) \right\|_1$$

Feature Field Distillation



$$\mathbf{p}(\mathbf{l}|\mathbf{x}) = \frac{\exp(\mathbf{f}(\mathbf{x})\mathbf{f}_q(\mathbf{l})^T)}{\sum_{l' \in \mathcal{L}} \exp(\mathbf{f}(\mathbf{x})\mathbf{f}_q(l')^T)} \quad (14)$$

Pseudo Code for MaskCLIP

Extract Free Dense Labels from CLIP³

Algorithm 1 Image Feature (Original)

```
1 def forward(x):  
2     q, k, v = W_qkv @ self.ln_1(x)  
3     v = (q[:1] * k).softmax(dim=-1) * v  
4     x = x + W_out @ v  
5     x = x + self.mlp(self.ln_2(x))  
6     return x[:1]      # the CLS token
```

Algorithm 2 Dense Features (MaskCLIP 11)

```
1 def forward(x):  
2     v = W_v @ self.ln_1(x)  
3     z = W_out @ v  
4     return z[1:] # all but the CLS token
```

CLIP for Patch-level Feature

```
def forward_v(self, x: torch.Tensor):  
    """  
    Forward function for computing the value features for dense prediction  
    (i.e., features for every image patch).  
    """  
    v_in_proj_weight = self.attn.in_proj_weight[-self.attn.embed_dim:]  
    v_in_proj_bias = self.attn.in_proj_bias[-self.attn.embed_dim:]  
    # raw_images [3, 3, 336, 597], patch_size: 14, tokenized [3, 24, 42, embed_dim]  
    v_in = F.linear(self.ln_1(x), v_in_proj_weight, v_in_proj_bias)  
    v_out = F.linear(v_in, self.attn.out_proj.weight, self.attn.out_proj.bias)  
    return v_out # pz note: [1009, 3, 1024] , 1009 = 24 * 42 + 1
```

Pixel, Ray and Patch

- Sample pixels then generate ray⁴

```
def next_train(self, step: int) -> Tuple[RayBundle, Dict]:  
    """Returns the next batch of data from the train dataloader."""  
    self.train_count += 1  
    image_batch = next(self.iter_train_image_dataloader)  
    assert self.train_pixel_sampler is not None  
    assert isinstance(image_batch, dict)  
    batch = self.train_pixel_sampler.sample(image_batch)  
    ray_indices = batch["indices"]  
    ray_bundle = self.train_ray_generator(ray_indices)  
    return ray_bundle, batch
```

⁴nerfstudio

Pixel, Ray and Patch

■ From pixel index to patch index

```
def next_train(self, step: int) -> Tuple[RayBundle, Dict]:  
    """Nearest neighbor interpolation of features"""  
    ray_bundle, batch = super().next_train(step)  
    ray_indices = batch["indices"]  
    camera_idx = ray_indices[:, 0]  
    y_idx = (ray_indices[:, 1] * self.scale_h).long()  
    x_idx = (ray_indices[:, 2] * self.scale_w).long()  
    batch["feature"] = self.train_features[camera_idx, y_idx, x_idx]  
    # (4096, 768) === (N_rays, feature_dim)  
    return ray_bundle, batch
```

Bridge 2D-to-3D Gap for Robotic Manipulation

Distilled Feature Fields Enable Few-Shot Language-Guided Manipulation⁵

- DFFs: 3D geometry + rich semantics from 2D VLMs
- 6-DOF grasp or place pose: $\mathbf{T} = (\mathbf{R}, \mathbf{t})$
- Few-shot manipulation: $< \{\mathbf{I}\}, \mathbf{T}^* >$, $\{\mathbf{I}\}_{i=1}^N$
- Open-text language-guided manipulation: L^+ , L^-

⁵CoRL 2023 - Best Paper

Transformation Explained

- SE(3): Special Euclidean group in 3 dimensions
- Rotation matrix example

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix}$$

- Translation

$$(x, y, z) \xrightarrow{(a,b,c)} (x + a, y + b, z + c)$$

Representing 6-DOF Poses with Feature Fields

- Approximate **local** 3D feature field via query points

$$\mathcal{X} = \{\mathbf{x} \in \mathbb{R}^3\}_{N_q}$$

- α -weighted features

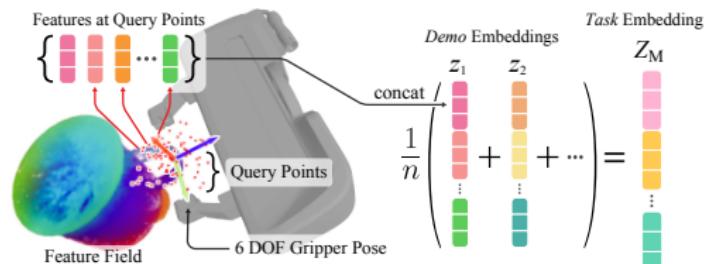
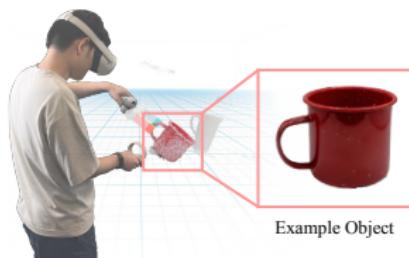
$$\mathbf{f}_\alpha(\mathbf{x}) = \alpha(\mathbf{x}) \cdot \mathbf{f}(\mathbf{x}), \quad \alpha(\mathbf{x}) = 1 - \exp(-\sigma(\mathbf{x}) \cdot \delta)$$

- Demo pose \mathbf{T} representation: sample and concatenate

$$\{\mathbf{f}_\alpha(\mathbf{x}) \mid \mathbf{x} \in \mathbf{T}\mathcal{X}\}, \quad \mathbf{z_T} \in \mathbb{R}^{N_q \cdot |\mathbf{f}|}$$

A More Concrete View

query points $\xrightarrow{\text{sampling}}$ features $\xrightarrow{\text{concat}}$ demo embed $\xrightarrow{\text{avg}}$ task embed



Inferring 6-DOF Poses

- Coarse pre-filtering

- geometric: $\alpha(\mathbf{v}) < \epsilon_{\text{free}} = 0.1$
- semantic: $\cos(\mathbf{f}_\alpha(\mathbf{v}), \mathbf{Z}_M) < \epsilon_{\text{task}}$

- Pose optimization

$$\mathcal{T} = \{T\}$$

$$\mathcal{J}_{\text{pose}}(\mathbf{T}) = -\cos(\mathbf{z}_{\mathbf{T}}, \mathbf{Z}_M) \quad (15)$$

Open-Text Language-Guided Manipulation

- Retrieving relevant demonstrations

$$\cos(\mathbf{q}, \mathbf{F}_d), \quad \mathbf{q} = \text{emb}_{\text{CLIP}}(L^+)$$

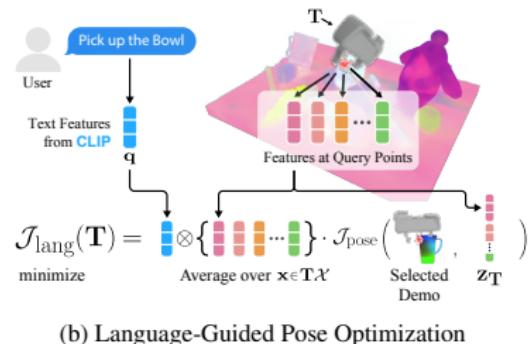
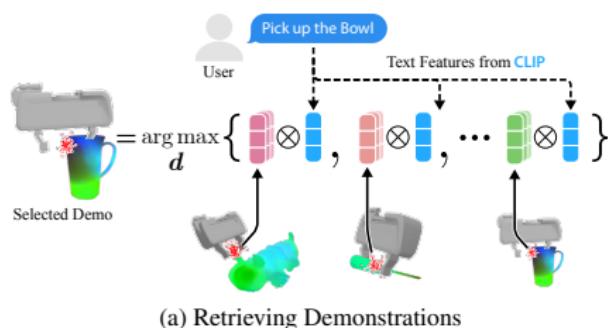
- Initializing grasp proposals

$$\mathbf{q}_i^- = \text{emb}_{\text{CLIP}}(L_i^-) \mid i \in \{i, \dots, n\}$$

- Language-guided grasp pose optimization

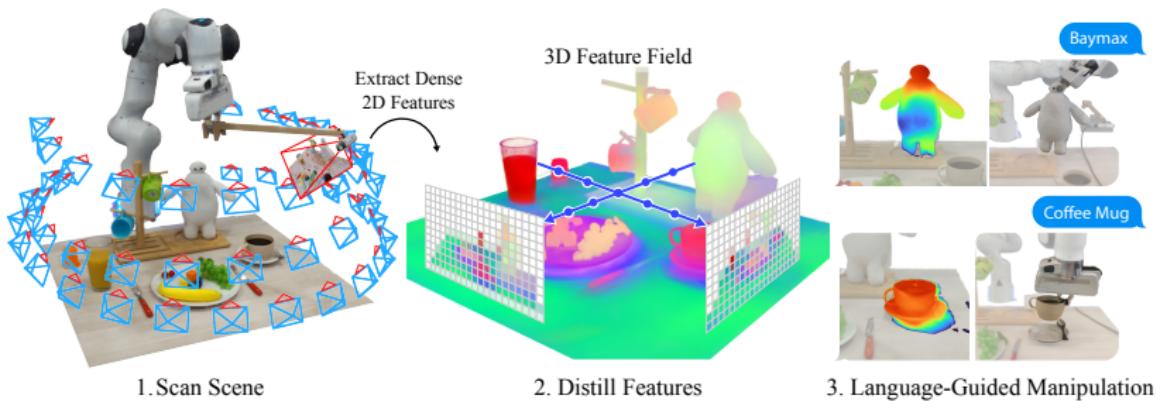
$$\mathcal{J}_{\text{lang}}(\mathbf{T}) = \text{mean}_{\mathbf{x} \in \mathbf{T}} \mathcal{X} \left[\mathbf{q} \otimes \mathbf{f}_\alpha(\mathbf{x}) \right] \cdot \mathcal{J}_{\text{pose}}(\mathbf{T}) \quad (16)$$

Pipeline for Language-Guided Manipulation



F3RM Overview

- Collecting multiple images of the scene
- Train a feature field
- Optimize grasp poses using language query and execute



Limitations

- Vision-language models may behave like bag-of-words [1]
- No incentives for model to learn compositional relations between objects.
- Lack of higher-level scene representation[2]

Limitations: Example

Similarity to language query "teddy bear"

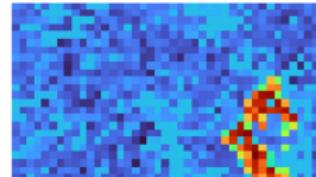
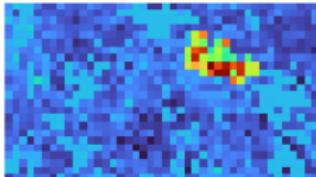
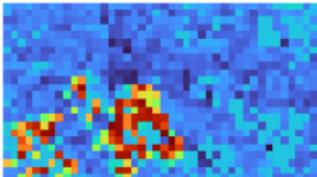
frame_1.png



frame_2.png



frame_3.png



Limitations: Example

Similarity to language query "mug that is next to teddy bear"

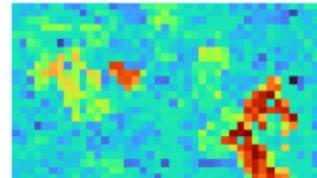
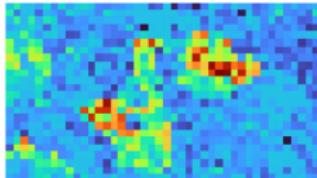
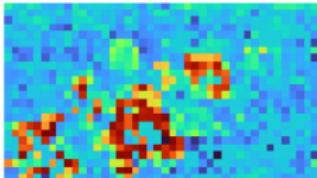
frame_1.png



frame_2.png



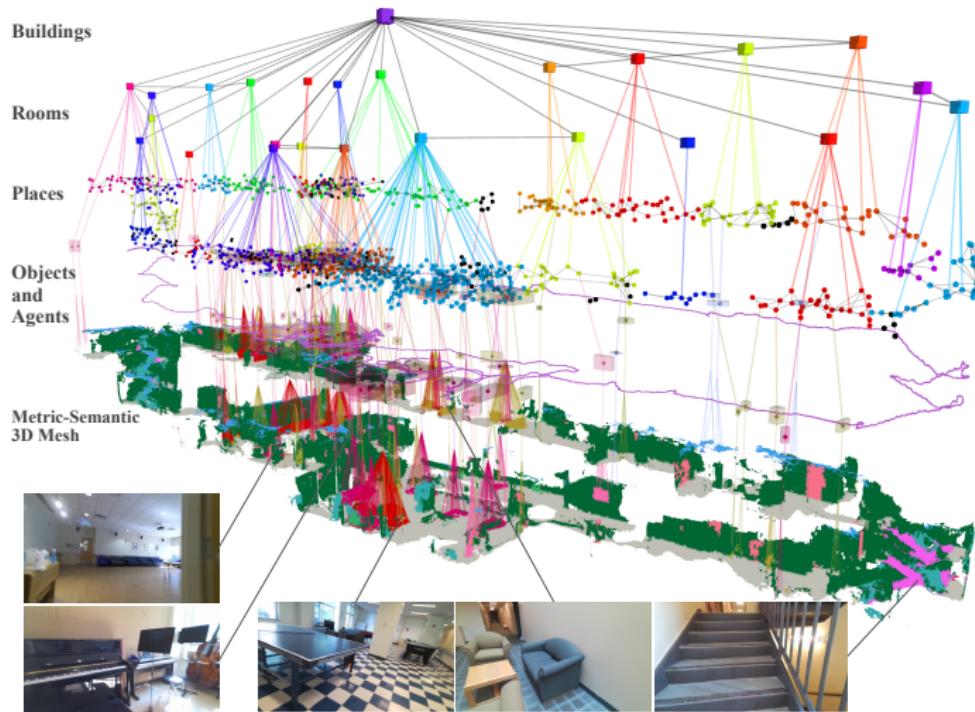
frame_3.png



Follow-up Questions

- Scene representation at different level [2]?
- Better **scene understanding**? Scene graph?
- Imitation or reinforcement learning?
- Datasets and experimental environment?

3D Scene Graph: A High-level Representation [3]



Simulation Environment

■ Available Physics Engine

- MuJoCo (**M**ulti-**J**oint dynamics with **C**ontact)
- MetaWorld
- PyBullet

■ Demo

References

- [1] **ICLR 2023** - When and why vision-language models behave like bag-of-words models, and what to do about it?
- [2] **CoRL 2022** - A Dual Representation Framework for Robot Learning with Human Guidance.
- [3] **RSS 2022** - Hydra: A real-time spatial perception system for 3D scene graph construction and optimization.

Thank you very much!
Q&A