A. Review of SurfEMB

Our noise model on RGB information builds on SurfEMB [1] which learns neural embeddings to establish dense 2D-3D correspondences via contrastive learning. For each object class $t \in \{1, \cdots, M\}$, SurfEMB learns a pair of neural networks: the query model $f_t : \{0, \cdots, 255\}^{H \times W \times 3} \mapsto \mathbb{R}^{H \times W \times E}$ and the key model $g_t : \mathbb{R}^3 \mapsto \mathbb{R}^E$. The query model transforms an observed RGB image $I$ into query embeddings $Q^t$, while the key model transforms a rendered object coordinate image $\tilde{X}$ into a set of key embeddings.

For a given 2D pixel location on the observed image with query embedding $q$, SurfEMB specifies a surface correspondence distribution $P_{\text{RGB}}(g_t(\tilde{x}) | q, t) \propto \exp(q^T g_t(\tilde{x}))$ for each object class $t$. To normalize this surface correspondence distribution, for each object class $t$, we subsample uniformly across the object’s surface to get a set $Z_t$ of surface 3D coordinates in object frame. Then, given a pixel with query embedding $q$, we calculate the probability that this pixel corresponds to a surface point $\tilde{x} \in Z_t$ on object class $t$ as:

$$
P_{\text{RGB}}(g_t(\tilde{x}) | q, Z_t, t) = \frac{\exp(q^T g_t(\tilde{x}))}{\sum_{\tilde{x} \in Z_t} \exp(q^T g_t(\tilde{x}))}$$

(1)

B. More details on the energy-based formulation of 3DNEL

B.1. Existence of the normalization constant for the energy-based formulation

Since we are working with an energy-based formulation (Equation (1)), to make the probability distribution properly defined we need to make sure the normalization constant, i.e. the sum of the energy function over all $I$ and $C$

$$
s \int I \prod_C \left( c \mathbb{P}_{\text{BG}}(c; B) + \frac{1 - \epsilon}{K} \sum_{\hat{c} : \hat{s} > 0} c \mathbb{P}_{\text{depth}}(c|\hat{c}; r) \mathbb{P}_{\text{RGB}}(g_{\hat{s}}(\tilde{x}) | q^\hat{s}, \hat{s}) \right)
$$

is finite and well-defined. For RGB images of size $H \times W$, since each pixel has only 256 values, there are at most $256^{H \times W}$ RGB images of size $H \times W$, which is a finite number. Since the value of the energy function is less than 1 for any given $I$ and $C$, summing over a finite number of $I$ and integrating over a bounded region for $C$ gives us a finite normalization constant, making the probability distribution well-defined.

*Equal contribution
B.2. JAX-based implementation of 3DNEL evaluation given rendering outputs

Given rendering outputs from OpenGL, we use JAX to develop a 3DNEL evaluation implementation that can run efficiently on modern GPUs. The implementation can be easily combined with \texttt{jax.vmap} to support 3DNEL evaluation of hundreds of 3D scene descriptions in parallel.

```python
@functools.partial(jax.jit, static_argnames="filter_shape")
def neural_embedding_likelihood(
    data_xyz: jnp.ndarray,
    query_embeddings: jnp.ndarray,
    log_normalizers: jnp.ndarray,
    model_xyz: jnp.ndarray,
    key_embeddings: jnp.ndarray,
    model_mask: jnp.ndarray,
    obj_ids: jnp.ndarray,
    data_mask: jnp.ndarray,
    r: float,
    p_background: float,
    p_foreground: float,
    filter_shape: Tuple[int, int],
):  
    ""
    Args:
    data_xyz: Array of shape (H, W, 3). Observed point cloud organized as an image.
    query_embeddings: Array of shape (H, W, n_objs, d).
        Query embeddings for each observed pixel using models from different objects.
        The log normalizers for each pixel given each object model
    model_xyz: Array of shape (H, W, 3). Rendered point cloud organized as an image.
    key_embeddings: Array of shape (H, W, d). Key embeddings organized as an image.
    obj_ids: Array of shape (H, W). The object id of each pixel.
    data_mask: Array of shape (H, W). Mask indicating the relevant set of pixels.
    r: Radius of the ball.
    p_background: background probability.
    p_foreground: foreground probability.
    filter_shape: used to restrict likelihood evaluation to a 2D neighborhood.
    ""
    obj_ids = jnp.round(obj_ids).astype(jnp.int32)
    padding = [
        for ii in range(len(filter_shape))
    ]
    model_xyz_padded = jnp.pad(model_xyz, pad_width=padding + [(0, 0)])
    key_embeddings_padded = jnp.pad(key_embeddings, pad_width=padding + [(0, 0)])
    model_mask_padded = jnp.pad(model_mask, pad_width=padding)
    obj_ids_padded = jnp.pad(obj_ids, pad_width=padding)

    @functools.partial(
        jnp.vectorize,
        signature="(m),(n),(o,d),(o)->()",
    )
    def log_likelihood_for_pixel(
        i: jnp.ndarray,
        data_xyz_for_pixel: jnp.ndarray,
        query_embeddings_for_pixel: jnp.ndarray,
        log_normalizers_for_pixel: jnp.ndarray,
    ):  
        ""
        Args:
        i: Array of shape (2,). The i, j index of the pixel.
        ""
        model_xyz_patch = jax.lax.dynamic_slice(
            model_xyz_padded,
            (i[0] - padding[0][0], i[1] - padding[1][0], 0, 0),
            (filter_shape[0] + 1, filter_shape[1] + 1, 3, 1),
        )
C. Details on coarse enumerative pose hypotheses generation

C.1. Formal description of the coarse enumerative pose hypotheses generation process

We develop a novel spherical voting procedure and a heuristic scoring using the query embeddings and observed point cloud image $C$ defined in Section 3.2, and use them in an enumerative procedure to efficiently generate pose hypotheses. We use the object center and $n_k$ points sampled using farthest point sampling from the object surface as our keypoints, and discretize the camera frame space into a $L_x \times L_y \times L_z$ voxel grid.

For a given keypoint, our spherical voting procedure aggregates information from the entire image to score how likely the keypoint is present at different voxel locations, and stores the scores in a voxel grid. Figure 1(a) visualizes spherical voting for the center $x^*$ of the mug object: for pixel location $(i, j)$ with camera frame coordinate $c \in \mathbb{R}^3$ and query embedding $q \in \mathbb{R}^E$, we identify its most likely corresponding point on the mug surface $x = \arg \max_{\tilde{x}} P_{\text{RGB}}(g_t(\tilde{x})|q, t)$, calculate the distance $r_x = ||x - x^*||_2$ from $x$ to $x^*$ in the object frame, and cast votes [3] with weight $p_{i,j} = \max_{\tilde{x}} P_{\text{RGB}}(g_t(\tilde{x})|q, t)$ towards all points on a sphere of radius $r_x$ centered at $c$. Figure 1(b) visualizes the 20 top-scoring voxels from the voxel grid for the mug center, and Figure 1(c) visualizes the 20 top-scoring voxels from the voxel grids for the $n_k$ keypoints on the mug surface.

We coarsely discretize the object pose space. We reuse the same camera frame space discretization into a voxel grid, and use the $L_x \times L_y \times L_z$ voxel centers to discretize the location space. We use a customized procedure (Appendix C.3) to generate $n_r$ rotations and discretize the rotation space. We use the voxel grid for the object center to identify top-scoring object locations, and score all $n_r$ rotations at these locations. We score a given object pose with the sum of the scores of the voxels the corresponding $n_k$ keypoints fall into. Figure 1(d) visualizes 3 example top pose hypotheses from the enumerative
Figure 1. Coarse Enumerative Pose Hypotheses Generation We visualize our coarse enumerative pose hypotheses generation process using a mug object as an example. Figure 1(a) visualizes spherical voting for identifying the mug center. The red points in the observed point cloud represent points associated with the mug. The 3 colored spheres illustrate how votes from points on the mug combine to identify the mug center. The same procedure can also be used to identify the \( n_k \) keypoints on the mug surface. Figure 1(b) visualizes 20 top-scoring voxels from the voxel grid associated with the mug center. Figure 1(c) visualizes 20 top-scoring voxels from the voxel grids associated with the \( n_k \) keypoints. Figure 1(d) show how we score pose hypotheses by summing the scores of the voxels the corresponding \( n_k \) keypoints fall into. The scores of the poses hypotheses decrease as we go from left to right. In Figures 1(b) to (d), light yellow represents low voxel scores, while dark red represents high voxel scores.

procedure. Figure 1(e) visualizes how poses far away from ground truth get low scores from our heuristic scoring.

In Algorithm 1 we present a detailed description of our coarse enumerative pose hypotheses generation process. In practice we also optionally apply non-max suppression in the TopPositions function in the algorithm to make sure the promising positions we identify are well spread out to cover different parts of the image and better represent uncertainty.

C.2. Taichi-based spherical voting

We use Taichi [2] to develop a spherical voting implementation that can run efficiently on modern GPUs.

```python
@ti.kernel
def taichi_spherical_vote(
    centers: ti.types.ndarray(element_dim=1),
    radiusses: ti.types.ndarray(),
    weights: ti.types.ndarray(),
    voxel_grid: ti.types.ndarray(),
    voxel_grid_start: ti.types.ndarray(element_dim=1),
    voxel_diameter: float,
    multipliers: ti.types.ndarray(),
):
    ""
    Args:
    centers: Array of shape \((\text{batch}_\text{size}, \text{n}_\text{centers}, \text{3})\). Coordinates of the centers of the spheres.
    radiusses: Array of shape \((\text{batch}_\text{size}, \text{n}_\text{centers})\). Radiiuses of the spheres.
    weights: Array of shape \((\text{batch}_\text{size}, \text{n}_\text{centers})\). Weights of votes from the spheres.
    voxel_grid: Array of shape \((\text{voxel}_\text{grid}_\text{shape})\). Coordinate of the center of voxel \((0, 0, 0)\)
    voxel_diameter: float. Diameter of a voxel.
    multipliers: fixed length-2 1D array with elements 1.0, -1.0
    ""
    for voxel in ti.grouped(voxel_grid):
        voxel_grid[voxel] = 0.0
```

Algorithm 1: Coarse Enumerative Pose Hypotheses Generation

/* Basic setups */

**Object-specific:** A set of surface points $Z_t$ for object class $t$, query model $f_t$, key model $g_t$, $n_k$ keypoints with object frame coordinates $x_{i1}^t, \ldots, x_{nk}^t \in \mathbb{R}^3$.

**Spatial discretization:** Camera frame coordinates of the center of the boundary voxel $y \in \mathbb{R}^3$, size of the voxel grid $(L_x, L_y, L_z)$, diameter of the voxels $d > 0$.

**Orientation discretization:** $n_t$ representative orientations $R_1, \ldots, R_{n_t} \in S(3)$.

**Parameters:** Number of pose hypotheses to generate $n_p$, number of top positions $n_t$.

/* Coarse Enumerative pose hypotheses generation */

**Input:** RGB image $I \in \mathbb{R}^{H \times W \times 3}$, observed point cloud $C \in \mathbb{R}^{H \times W \times 3}$.

**Output:** Top scoring pose hypotheses $P_1^t, \ldots, P_{n_p}^t \in \mathbb{SE}(3)$.

1./* Coarse Enumerative pose hypotheses generation */
   
   **Input:** RGB image $I \in \mathbb{R}^{H \times W \times 3}$, observed point cloud $C \in \mathbb{R}^{H \times W \times 3}$.

2. /* Coarse Enumerative pose hypotheses generation */
   
   **Output:** Top scoring pose hypotheses $P_1^t, \ldots, P_{n_p}^t \in \mathbb{SE}(3)$.

/* Voting and heuristic scoring */

**def** Voting $(Q, C, x^t)$ // Voting-based evidence aggregation

1. /* Voting and heuristic scoring */
   
   **def** Voting $(Q, C, x^t)$ // Voting-based evidence aggregation

/* Scoring */

**def** Scoring $(l, R, V^1, \ldots, V^{n_k})$ // Heuristic pose scoring

2. /* Scoring */
   
   **def** Scoring $(l, R, V^1, \ldots, V^{n_k})$ // Heuristic pose scoring

/* Local Utilities */

**def** Round $(x)$ // Round $x$ to the nearest integer.

3. /* Local Utilities */
   
   **def** Round $(x)$ // Round $x$ to the nearest integer.

/* Voting */

**def** Voting $(Q, C, x^t)$ // Voting-based evidence aggregation

4. /* Voting */
   
   **def** Voting $(Q, C, x^t)$ // Voting-based evidence aggregation

/* Heuristic Scoring */

**def** Scoring $(l, R, V^1, \ldots, V^{n_k})$ // Heuristic pose scoring

6. /* Heuristic Scoring */
   
   **def** Scoring $(l, R, V^1, \ldots, V^{n_k})$ // Heuristic pose scoring

/* Voting */

**def** Voting $(Q, C, x^t)$ // Voting-based evidence aggregation

7. /* Voting */
   
   **def** Voting $(Q, C, x^t)$ // Voting-based evidence aggregation

/* Heuristic Scoring */

**def** Scoring $(l, R, V^1, \ldots, V^{n_k})$ // Heuristic pose scoring

8. /* Heuristic Scoring */
   
   **def** Scoring $(l, R, V^1, \ldots, V^{n_k})$ // Heuristic pose scoring

/* Local Utilities */

**def** Round $(x)$ // Round $x$ to the nearest integer.

9. /* Local Utilities */
   
   **def** Round $(x)$ // Round $x$ to the nearest integer.

/* Coarse Enumerative pose hypotheses generation */

10. /* Coarse Enumerative pose hypotheses generation */

/* Voting and heuristic scoring */

11. /* Voting and heuristic scoring */

/* Voting */

12. /* Voting */

/* Heuristic Scoring */

13. /* Heuristic Scoring */

/* Local Utilities */

14. /* Local Utilities */

/* Coarse Enumerative pose hypotheses generation */

15. /* Coarse Enumerative pose hypotheses generation */

/* Voting and heuristic scoring */

16. /* Voting and heuristic scoring */

/* Voting */

17. /* Voting */

/* Heuristic Scoring */

18. /* Heuristic Scoring */

/* Local Utilities */

19. /* Local Utilities */

/* Coarse Enumerative pose hypotheses generation */

20. /* Coarse Enumerative pose hypotheses generation */

/* Voting and heuristic scoring */

21. /* Voting and heuristic scoring */

/* Voting */

22. /* Voting */

/* Heuristic Scoring */

23. /* Heuristic Scoring */

/* Local Utilities */

24. /* Local Utilities */

/* Coarse Enumerative pose hypotheses generation */

25. /* Coarse Enumerative pose hypotheses generation */

/* Voting and heuristic scoring */

26. /* Voting and heuristic scoring */

/* Voting */

27. /* Voting */

/* Heuristic Scoring */

28. /* Heuristic Scoring */

/* Local Utilities */

29. /* Local Utilities */

/* Coarse Enumerative pose hypotheses generation */

30. /* Coarse Enumerative pose hypotheses generation */

/* Voting and heuristic scoring */

31. /* Voting and heuristic scoring */

/* Voting */

32. /* Voting */

/* Heuristic Scoring */

33. /* Heuristic Scoring */

/* Local Utilities */

34. /* Local Utilities */

/* Coarse Enumerative pose hypotheses generation */

35. /* Coarse Enumerative pose hypotheses generation */

/* Voting and heuristic scoring */

36. /* Voting and heuristic scoring */
C.3. Discretizing the rotation space

We discretize $SO(3)$ into 6400 representative orientations. We generate these orientations by first picking 200 points roughly uniformly on the unit sphere using the Fibonacci sphere. The 6400 representative orientations are generated by first rotating the axis $(0,0,1.0)$ to point to one of the 200 points, followed by one of 32 in-plane rotations around the axis.

D. Details on inference pipeline implementation

D.1. Using additional 2D detection and mask prediction from SurfEMB

In the best performing setup, we leverage the same 2D detector used in SurfEMB as part of the pose hypotheses generation process. Although our spherical voting procedure can robustly aggregate information from the entire image to generate pose hypotheses, as demonstrated by the competitive performance of 3DNELEXSIGP (No 2D detection) (Abalations in Table 1), in practice the query embedding images for many objects are very noisy and tend to hurt performance. Empirically, we observe
that by additionally using 2D detections, we can focus the spherical voting process on regions of the observed image that is likely relevant for the objects, and further improve performance.

For each object class in the scene, there can be multiple 2D detections. For each 2D detection, we do spherical voting just within the detector crop and generate 80 pose hypotheses per detector crop. When there is a missing 2D detection, we obtain the query embeddings by upsampling the input RGB image by 1.5x, and do spherical voting on the whole image. In such cases, when we identify top positions, we additionally do non-max suppression with a filter size of 10 to spread the top-scoring positions out. For each such top-scoring position we identify top 2 orientation, and we consider all top-scoring positions and generate in total 30 pose hypotheses.

D.2. Implementation details and hyperparameters

We use OpenGL for rendering, use Taichi for spherical voting, and use JAX for 3DNEL evaluation. We refer the readers to the attached Python source code for a complete implementation of our pipeline.

As we describe in the main text, we pick hyperparameters by visually inspecting detection results on a small number of real training images that are outside the test set.

We select \( n_k = 8 \) keypoints from the surface of each object class, and use \( y = (-350.0, -210.0, 530.0), L_x = 129, L_y = 87, L_z = 168 \) and \( d = 5.0 \) to make the voxel grids large enough to cover all the keypoints that can be present in the camera frame. Here the units for the values in \( y \) and \( d \) are mm.

We use \( r = 5.0 \) in evaluating 3DNEL for all our experiments. When identifying points in the rendered point cloud \( \hat{C} \) (organized as an \( H \times W \times 3 \) image) that is within distance \( r \) from a point \( (i, j) \) in the observed point cloud \( C \), to further speed up 3DNEL evaluation, we only look at the points from the patch of size \( (10, 10) \) centered at \( (i, j) \).

In Figure 3 we include additional visualizations of SurfEMB and 3DNEL MSIGP predictions on YCB-V test images.

E. Additional robustness results

In Figure 2 we include additional robustness results.

F. Additional visualizations of SurfEMB and 3DNEL MSIGP predictions on YCB-V test images
Figure 3. Additional visualizations of SurfEMB and 3DNL MSIGP predictions on YCB-V test images.
References

