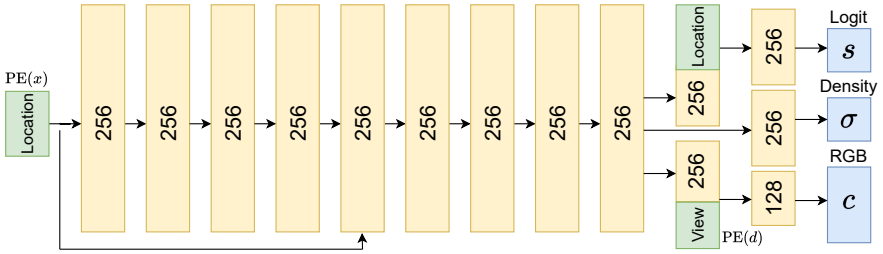

Fig. 7: The architecture used for LaTeRF is a multi layer perceptron which inspired by [54], extends the original NeRF model [23] to contain an additional output s to reason about the probability of different points in the space being part of the OOI. As evident in the figure, s is independent of the view d and only depends on the location x. Following the literature, both location x and view direction d are passed through a positional encoding function (PE).

A Additional Implementation Details

Our proposed method is implemented in PyTorch [29], and the network is optimized using the Adam optimizer [16] to learn the view-dependent radiance (color) and view-independent densities and objectness probabilities for points in the space. The network is randomly initialized and trained from scratch for each scene individually. Hyperparameters related to the NeRF model and the optimizer are set based on the PyTorch implementation [48] in the original NeRF paper [23]. Training is done on one NVIDIA GeForce RTX 3090 GPU. Due to the high GPU memory usage for the CLIP loss calculations, the object renders used for computing $\mathcal{L}_{\text{CLIP}}$ are $\frac{1}{3}$ of the size of the original input images, e.g., for 400×400 pixel synthetic views, the object is rendered at 133×133 pixels and the CLIP loss is defined over the downsized rendering. Moreover, in order to speed up training, the CLIP loss is only calculated every 10 steps. For test-time object renderings, instead of using the soft-partitioning approach, a hard-thresholding method is used to denoise the background. In the next section, we introduce the details of this denoising mechanism. An overview of the architecture of the MLP with the additional objectness score s as output is shown in Figure 7.

For real-world scenes, a minimal user interface is designed to capture the pixel annotations from an end-user. In this procedure, instead of limiting the user to give visual cues pixel-by-pixel, we allow brush size changes to enable the selection of areas corresponding to either the OOI or non-objects. This allows for quick annotation of points far from the object boundaries, resulting in a better non-object removal and object discovery in the scene. Using dynamic brush sizes, we were able to collect millions of pixel-level annotations in just a few minutes. By being able to change the brush size, the end-user can coarsely label the points that are not close to the boundaries, and then reduce the annotation area as they get closer to the boundaries to finely label the more important pixel-level data

(i.e., the boundary of the object and the foreground/background, as shown in the experiments).

B Denoising the Views

As mentioned in section 5.1, we use a post-processing approach to denoise the rendered images of the objects. Noise is mostly caused by small particles that emerge in the training of the NeRF model and that blend in with the background in the training views, but become visible as the non-objects (including the background) are removed from the scene. In addition, for the points in the space that are inside of a dense object or behind the background, the objectness score is not trained well since the densities of the surface points block the training signals. All these reasons contribute to noisy renderings of the OOI when using LaTeRF without additional denoising (see soft threshold results in Fig 8). Our first step to mitigate this issue is to smooth the densities. Because the noisy particles are mostly steep 'jumps' in density compared to the neighbouring points, substituting the value of every density with the average of its neighbours, including itself, will smooth these bumps. We repeat this averaging for 5 steps. Afterward, we filter the points with densities lower than a small threshold (which was set to 0.2 in the example in Fig 8). We call this approach hard thresholding and it is evident in Fig 8 that it has helped to reduce the noise, but that there are still some unpleasant gray artifacts in the renderings. However, we do not directly use hard thresholding to render the RGB images of the object; we only use it to render the silhouette of the object to mask it out from the soft threshold results. After applying the hard threshold, we render the object mask by substituting the color with the objectness scores in Eq. 10 and applying the sigmoid function:

$$\hat{P}_{\text{obj}}(r) = \text{Sigmoid}\left(\sum_{i=1}^{N} T_i^{\text{obj}}\big(1 - \exp(-\sigma_i p_i \delta_i)\big) s_i\right). \tag{16}$$

Note that we assume a background with 0 objectness probability when rendering the object so that only object points with high densities will dominate this background and, as such, we obtain the object mask as an output. The rendered mask is then applied to the soft-threshold results to yield the rendered images of the object without background artifacts (final results are shown in Figure 8).

C Additional Real-world Results

Novel-view renderings of additional objects (partly borrowed from [23,4]) are shown in Figure 9. These examples include objects with detailed textures and geometries and objects with challenging shiny surfaces.

D Relighting the Object

It is possible to leverage the dependence of color of a point on the view direction to 'trick' the learned object radiance field to render the OOI under novel lighting

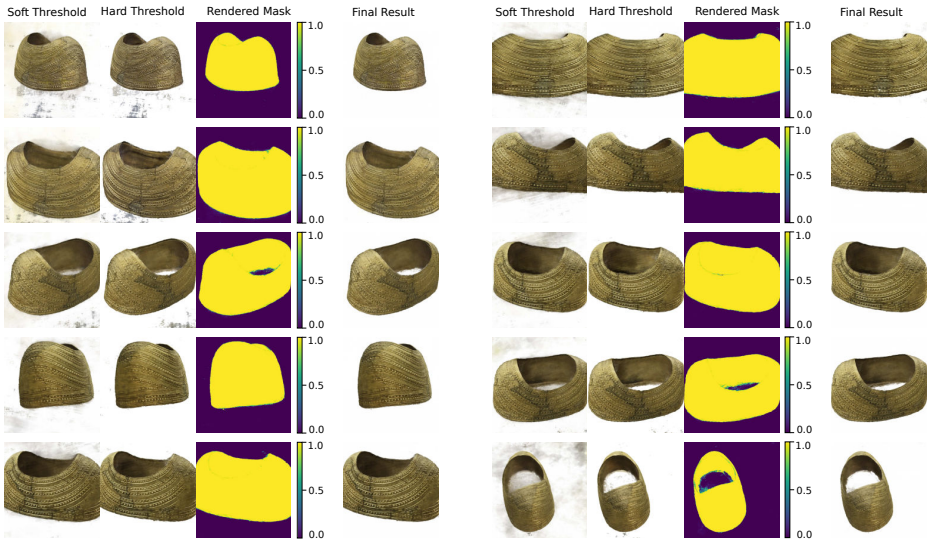

Fig. 8: More examples of the effectiveness of our denoising approach for removing the background artifacts using rendered objectness probabilities which act as object masks on the goldcape scene [30].

conditions [23]. The view direction fed to the network to find the radiance (color) of points in the space can be manually rotated while keeping the camera fixed. The effect caused by this alteration is similar to changing the lighting of the scene, and it is possible to fit the novel lighting to be consistent with certain lighting conditions. Figure 10 shows some of the real-world objects with three different illumination choices for a given, fixed view.

D.1 Blending the Object in Novel Scenes

The lighting setting can later be optimized with respect to the desired lighting in a novel scene, making an inserted object look consistent in a new scene. Figure 11 shows an example of placing a 3D asset extracted by LaTeRF into a scene under two different illumination conditions.

Fig. 9: Additional real-world object extraction results.

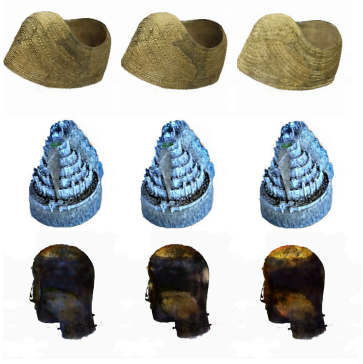

Fig. 10: Relighting objects under three different illumination conditions.

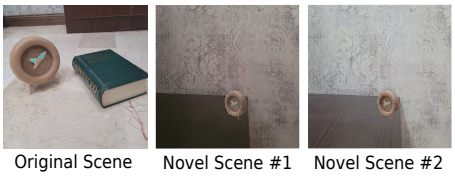

Fig. 11: An example of placing an extracted object in a scene with two different lighting conditions.