# Counting Stacked Objects

Supplementary Material

## 7. Dataset Details

### 7.1. 3DC-Real dataset.

We capture 45 real scenes where the objects to count can be any stack of items that are at least partially visible. This includes stacked objects on a table or on the floor, objects in containers such as bowls or boxes, or objects still in their packaging.

**Cameras.** We use a regular RGB smartphone camera to capture 30-60 pictures of the scene from various angles, forming a semisphere surrounding the objects and their container. These images are downscaled to approximately 600 pixels wide to reduce memory usage and facilitate the processing with COLMAP [20]. Additionally, we take a measurement of an arbitrary object within the scene, allowing us to scale the camera measurements and align the unit distance of the scene with a meter in the real world.

Initially, we experimented with triangulation methods using two pairs of corresponding points across images. This would enable the calculation of a 3D distance and allow us to scale the scene. However, this approach proved to be unstable, as small inaccuracies in point matching led to significant variations in the scaling. Instead, we reconstructed the 3D scene with 3DGS [6] and measured the 3D distance directly within the reconstruction. This measurement allows us to rescale the scene to match the reference measurement. Note that the 3D point cloud generated by COLMAP [20], which is used as an initialization by 3DGS, is also scaled accordingly.

**Unit volume.** For each scene, we require the unit volume of the object being counted. For complex shapes, we determine the unit volume $v$ by 3D reconstruction, similar to the method described in our main paper. For many common food items, such as kidney beans or corn, this information is readily available online. For other scenes, the volume can be approximated, for example in the case of the beads in Fig. 1, by subtracting the volume of a cylinder from that of a sphere.

**Pre and Post-processing.** Using this method, we capture 45 scenes consisting of various items in different environments. The scenes vary in complexity, from simple quasi-spherical objects in containers to more challenging configurations, such as complicated shapes still in packaging (e.g., in the *pasta* scene). Some items are used in several scenes, in which case the container and location are modified to create a new setting.

### 7.2. 3DC-Synthetic Dataset.

To generate our large-scale synthetic dataset, we utilize Blender, a free and open-source 3D creation suite that supports Python scripting. This allows us to implement a fully automated generation pipeline, which is mainly composed of two steps: simulation and rendering.

**Simulation.** We drop batches of objects, arranged in a $4 \times 4 \times 5$ grid, into a box positioned at $(0, 0, 0.5)$. The box has a side length of $1$ and a thickness of $0.04$. Once the simulation converges, we check if the union of the objects intersects with an invisible cube placed directly on top of the box. If an intersection occurs, the simulation stops, and objects outside the box are deleted. If no intersection is detected, a new batch of objects is added, and the simulation is performed again.

We use the convex hull to compute collision between objects. Ideally, we would use the triangle mesh itself, however this becomes far too costly when physically simulating thousands of shapes with tens of thousands of triangles. We experimented with using convex hulls first, and then refining with additional frames using the triangle mesh, but this turned out to still be extremely costly and computationally very unstable, leading to objects being ejected outside the box due to the change in collision computation.

**Rendering.** For rendering, we use a texture randomly sampled from 3 possibilities for the box, five textures for the ground, and a random material for each model chosen from one of the following: a realistic grey metal texture, a red metallic texture, or a plastic material with a randomly selected color.

We always render the first view directly above the box, looking downwards, which we call the *nadir* view. For the validation dataset, we also generate 29 additional views on the unit sphere, each observing the box from different angles. The rendering is performed using Blender's Cycles rendering engine. Additionally, we generate ground-truth depth maps and masks that separate the ground, box, and objects in the images.

**Pre and Post-processing.** In addition to the simulation and rendering steps, we perform pre-processing to filter out unsuitable meshes, such as those with multiple connected components or excessive size. Since the physical simulation can sometimes be unstable or fail, we also remove a small fraction of results in post-processing. This includes cases where the unit volume is too small or where too few objects remain in the box in the final frame.

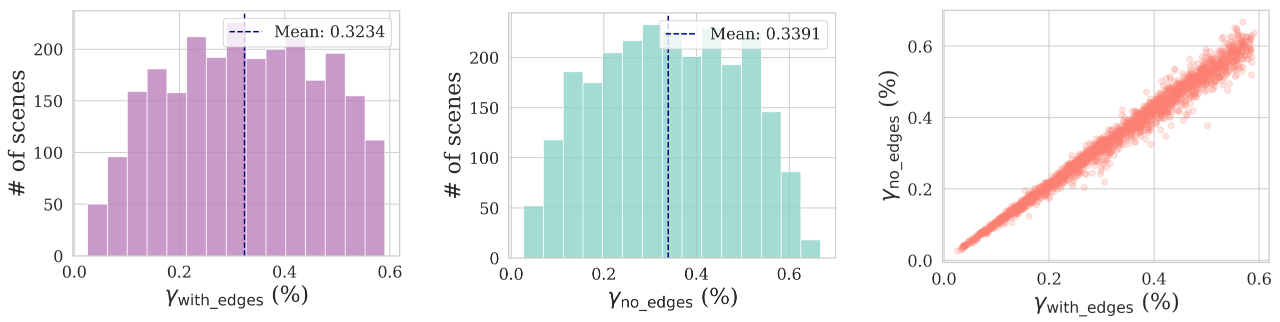

Figure 10. **Border effects.** Measuring the ground-truth $\gamma$ over the complete box or over only a smaller section makes little difference, indicating that our assumption of uniform $\gamma$ across the volume is justified.

Finally, we export the calibrated camera parameters in a format compatible with nerfstudio [25]. Since these cameras are not produced by COLMAP, they do not include a 3D point cloud that 3DGS can use as initialization. This poses a challenge, as a fully random initialization may generate distant Gaussian points outside the cameras' range, which are not removed and interfere with the volume estimation. To address this, we generate a set of 100 grey points within the unit cube, centered at $(0, 0, 0.5)$. This simple initialization proves sufficient to quickly produce a faithful 3D reconstruction and resolves the aforementioned issue.

## 8. Additional comparison distinguishing visible and invisible objects.

Our method is, to the best of our knowledge, proposing the first solution for this task. The methods used as comparison in 4 are initially designed to count only visible objects, and perform poorly on our dataset. In this section, we distinguish between visible and invisible objects in stacks and evaluate baseline methods against both counts. To perform this evaluation, we manually annotate the locations of visible objects in real scenes and provide the visible count and locations in our released datasets. Two examples are displayed in Fig. 11.

The results of this experiment can be found in Tab. 6. While the numbers for the 2D counting methods improve, it remains hard for them to distinguish similar objects clumped together. As a result, their performance re,ains much lower on these challenging scenes than on traditional 2D counting benchmarks.

## 9. Evaluation metrics

In this section, we provide the exact formula behind the metrics used in Sec. 4. As explained, the NAE and SRE are defined as:

$$\text{NAE} = \frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{\sum_{i=1}^{n} y_i}, \quad \text{SRE} \quad = \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} y_i^2} \ .$$

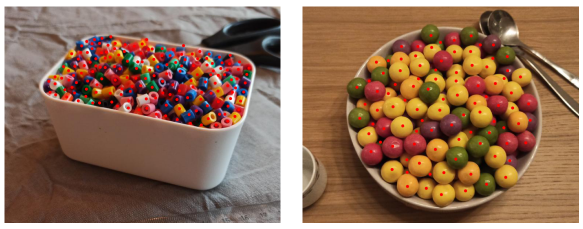

Figure 11. **Localization of visible objects.** Left: $\mathcal{N}_{vis} = 168$, Right: $\mathcal{N}_{vis} = 96$

| | NAE ↓ | SRE ↓ | sMAPE ↓ |
|---|---|---|---|
| *Visible objects only* | | | |
| BMNet+ [21] | **0.51** | **0.65** | 51.28 |
| SAM+CLIP [7,16] | 0.57 | 0.81 | **50.84** |
| *All objects* | | | |
| BMNet+ [21] | 0.93 | 0.98 | 131.44 |
| SAM+CLIP [7,16] | 0.94 | 0.99 | 124.31 |
| Ours | **0.36** | **0.06** | **53.31** |

Table 6. **Counting visible and invisble objects separately.**

We also report the Symmetric Mean Absolute Percentage Error (sMAPE), which can be considered a normalized percentage error. It is expressed as follows:

$$\text{sMAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{(y_i + \hat{y}_i)/2},$$

The formula of sMAPE ensures that errors are scaled symmetrically between the prediction and ground truth counts. Finally, the coefficient of determination, $R^2$, measures the proportion of variance in the ground-truth occupancy ratio $\gamma$ explained by our predictions

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \overline{y})^2},$$

where $\overline{y}$ is the mean of the ground truth counts. High values of $R^2$ indicate strong agreement between predictions and ground-truth values.

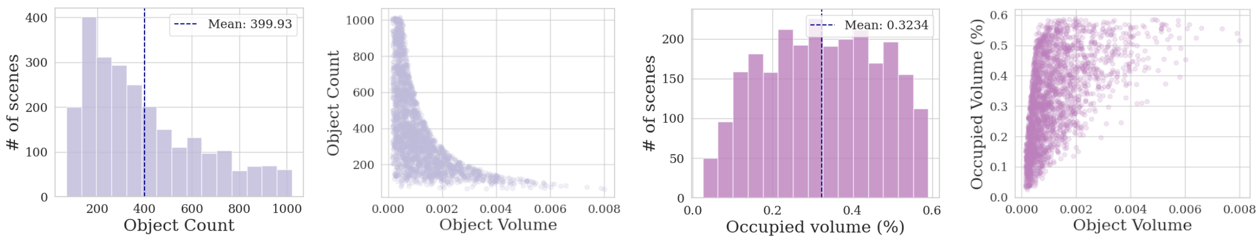

Figure 12. **Dataset statistics.** The histograms represent the distributions of object count and occupancy ratio, respectively, and each bar plots the number of scenes in a given bin. In scatter plots, each point represents a physically simulated 3D scene. In particular, the occupancy ratio $\gamma$ spans a large range between 1% and 65%

## 10. Additional Discussion on Border Effects

In our work, we assume that the occupancy ratio, $\gamma$, is approximately uniform throughout the container. This assumption generally holds as the number of stacked objects increases. However, it neglects the influence of container borders, where objects tend to occupy less volume due to the boundary.

To quantitatively evaluate the impact of border effects and verify the validity of our uniform $\gamma$ assumption, we analyze the ground-truth volume ratio in two distinct ways using our large-scale synthetic dataset. First, we compute $\gamma_{\text{with\_edges}}$ for the entire unit box, as described in the main paper. Additionally, we compute $\gamma_{\text{no\_edges}}$ by measuring the volume ratio in a smaller sub-box of side length 0.5, centered within the unit box. Intuitively, the difference between $\gamma_{\text{with\_edges}}$ and $\gamma_{\text{no\_edges}}$ reflects the influence of border effects, allowing us to evaluate whether this assumption is justifiable.

Figure 10 presents two histograms comparing the distributions of $\gamma_{\text{with\_edges}}$ and $\gamma_{\text{no\_edges}}$. The results indicate that both metrics follow highly similar distributions, with their mean values differing by less than 5%. Notably, the mean value of $\gamma_{\text{no\_edges}}$ is slightly higher than that of $\gamma_{\text{with\_edges}}$, consistent with the intuition that density decreases near borders.

To further investigate the relationship between these two values, we provide a scatter plot of $\gamma_{\text{with\_edges}}$ versus $\gamma_{\text{no\_edges}}$ in Fig. 10. The plot demonstrates a strong correlation between the two measures, particularly for objects with small volume ratios. For objects with high values of both $\gamma_{\text{with\_edges}}$ and $\gamma_{\text{no\_edges}}$, minor discrepancies are observed. These differences can be attributed to the relatively large size of these objects compared to the measurement box, which introduces noise in the estimation of $\gamma$.

Overall, these analyses confirm that $\gamma_{\text{with\_edges}}$ and $\gamma_{\text{no\_edges}}$ are highly consistent and can be used interchangeably without significant loss of accuracy. In our experiments, we rely on $\gamma_{\text{with\_edges}}$ to train our occupancy ratio estimation network.

## 11. Implementation details

We use the nerfstudio library [25] for 3D reconstruction, specifically the *splatfacto* method built on top of the gsplat library [32]. We thank the contributors of all the aforementioned libraries.

Our pipeline also uses pretrained models for depth estimation and mask generation. We employ the *vitl* model from Depth Anything v2 [30] for depth estimation and the *sam2.1_hiera_large* model from SAM2 [19] for mask generation. These state-of-the-art models ensure high-quality and robust outputs across diverse scenes.

The dataset is generated using CPUs only, greatly reducing its production cost and environmental impact. Other operations are fairly light and performed locally on a 4080 Mobile GPU, taking up only a few gigabytes of VRAM and being completed in a couple minutes.

## 12. Architecture details

Our architecture utilizes a DinoV2 [15] encoder model that produces pixel-aligned features. Since DinoV2 downscales the input image by 14, we feed it an image of size 448 x 448 to produce a 32 x 32 x 768 feature image. Specifically, we use the pretrained weights of the *dinov2_vitb14* model and freeze them during all subsequent learning.

To predict a scalar value from the $32 \times 32 \times 768$ feature image produced by DinoV2, we employ a series of convolutional layers to progressively reduce both the spatial dimensions and the number of channels. The convolutional layers successively reduce the channel dimension from the initial 768 down to 512, 256, 128, and finally 64. Concurrently, the spatial dimensions of the feature map are reduced from $32 \times 32$ to $16 \times 16$, $8 \times 8$, $4 \times 4$, and ultimately $2 \times 2$.

Following this, an adaptive average pooling layer compresses the spatial dimensions to a single pixel while preserving the 64-channel depth. The resulting $1 \times 1 \times 64$ tensor is passed through a fully connected linear layer to map it to a scalar output. Finally, a sigmoid activation function is applied to produce the final prediction in the $[0, 1]$ range.