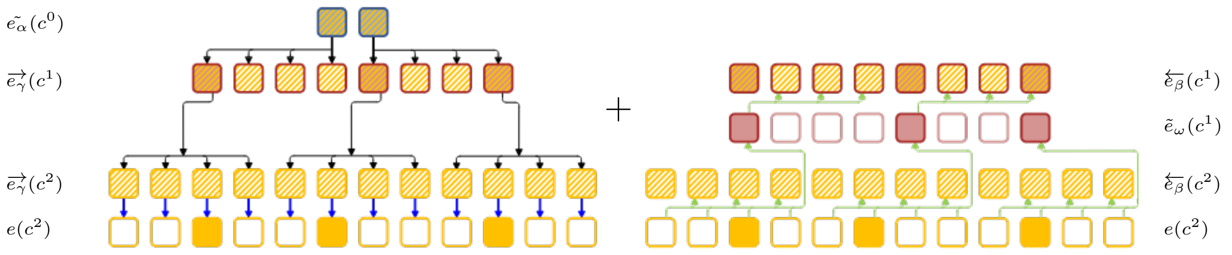

$\tilde{e_\alpha}(c^0)$
$\overrightarrow{e_\gamma}(c^1)$
$\overrightarrow{e_\gamma}(c^2)$
$e(c^2)$
$+$
$\overleftarrow{e_\beta}(c^1)$
$\tilde{e}_\omega(c^1)$
$\overleftarrow{e_\beta}(c^2)$
$e(c^2)$

Figure 8. A more detailed version of figure 4 showing the two distinct directions of information flow from the root downwards to the leafs (left) and from the leafs upwards (right). Information on the same level is added up to obtain $\hat{e}(c)$ The blue arrows signify sampling with subsequent encoding and are only necessary during inference.

A. Compression Scheme

Here we give more details for the compression and decompression algorithms presented in sections 3.2 and 3.3. For all formulas and examples we assume we want to compress an entire subtree starting at node c^0.

A.1. Encoding

In the encoding stage each node which has children gathers their information.

$$\tilde{e}_\alpha(c^l) = \begin{cases} e(c^l), \text{if } c^l \text{ is full or empty or at finest layer} \\ f_\alpha(\tilde{e}_\alpha(c_0^{l+1}), \dots, \tilde{e}_\alpha(c_{k-1}^{l+1})), \text{else} \end{cases} \tag{1}$$

Here c^l is a cell in layer l where we start counting from the root of the currently compressed subtree. Furthermore $e(c)$ is the token embedding as introduced in section 2. and $\tilde{e_\alpha}(c)$ is the compressed embedding, which is defined recursively on the children of node c^l : $C(c^l) = c_0^{l+1}, \dots, c_{k-1}^{l+1}$ (for an octree $k = 8$). f_α is an arbitrary learnable compression function. We use a single linear function with non-linearity, but of course more complex functions are possible. The weights α for this function differ for each layer l, but we omit a subscript for readability. The compressed feature vector of the subtree is then $\tilde{e}_\alpha(c^0)$.

A.2. Decoding

The decoding stage has two ”directions”. Information flows downwards from the root and upwards from the already sampled leaf nodes. The upward direction is given by the formula:

$$\overleftarrow{e_\beta}(c_i^l) = f_\beta(\tilde{e}_\omega(c_0^l), \dots, \tilde{e}_\omega(c_j^l)) \tag{2}$$

where $j < i$ and $c_i^l, c_j^l \in C(c^{l-1})$. Note that $\tilde{e}_\omega$ is the same function used in the compression, but with different learnable weights. Therefore $\overleftarrow{e_\beta}(c_i^l)$ depends on all its already generated siblings (if we are at leaf level) or their descendants otherwise.

The downward direction is given by:

$$\overrightarrow{e_\gamma}(c_i^{l+1}) = \begin{cases} f_\gamma^i(\tilde{e}_\alpha(c_l)), c_i^{l+1} \in C(c^l) & l \text{ is root level} \\ f_\gamma^i(\hat{e}(c_l)), c_i^{l+1} \in C(c^l) & \text{else} \end{cases} \tag{3}$$

This function can be seen as a transposed convolution, where all children depend on their parent, but with different weights (denoted with the superscript i in f_γ^i). Note that in this step we already integrate the information from the upward flow through $\hat{e}(c_l))$, which is defined as:

$$\hat{e}(c_i) = \overrightarrow{e_\gamma}(c_i) + \overleftarrow{e_\beta}(c_i) \tag{4}$$

During training the upward direction can be computed before the downward direction, as all information is already present. During inference these computations need to be interleaved.

B. Open Source

The research code is open source and can be accessed at `https://github.com/GregorKobsik/Octree-Transformer`.

C. Samples

Additional examples from both our unconditional (Fig. 9) and conditional (Fig. 10) sampling schemes at resolutions of 64^3, 128^3, 256^3.

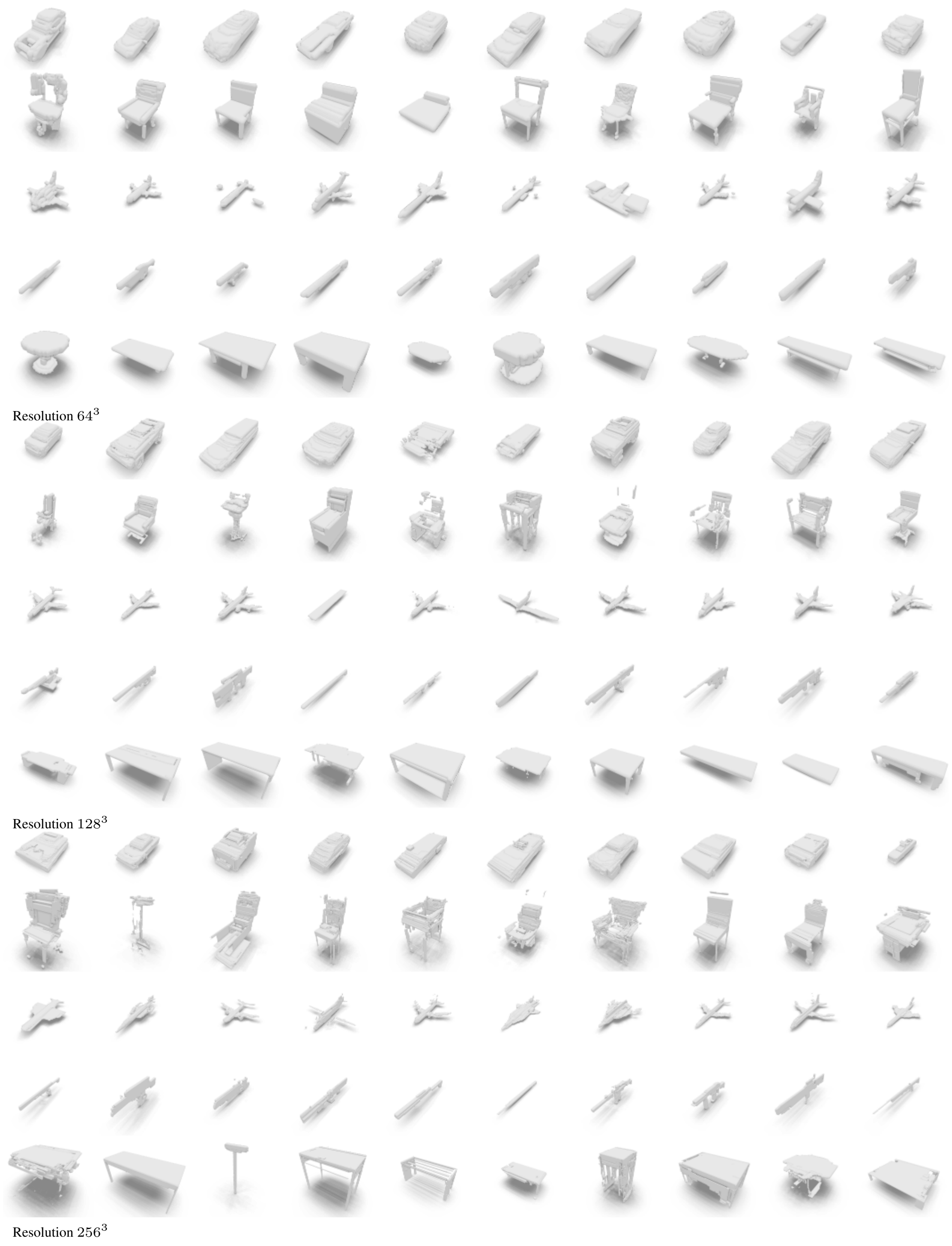


Figure 9. Unconditional samples

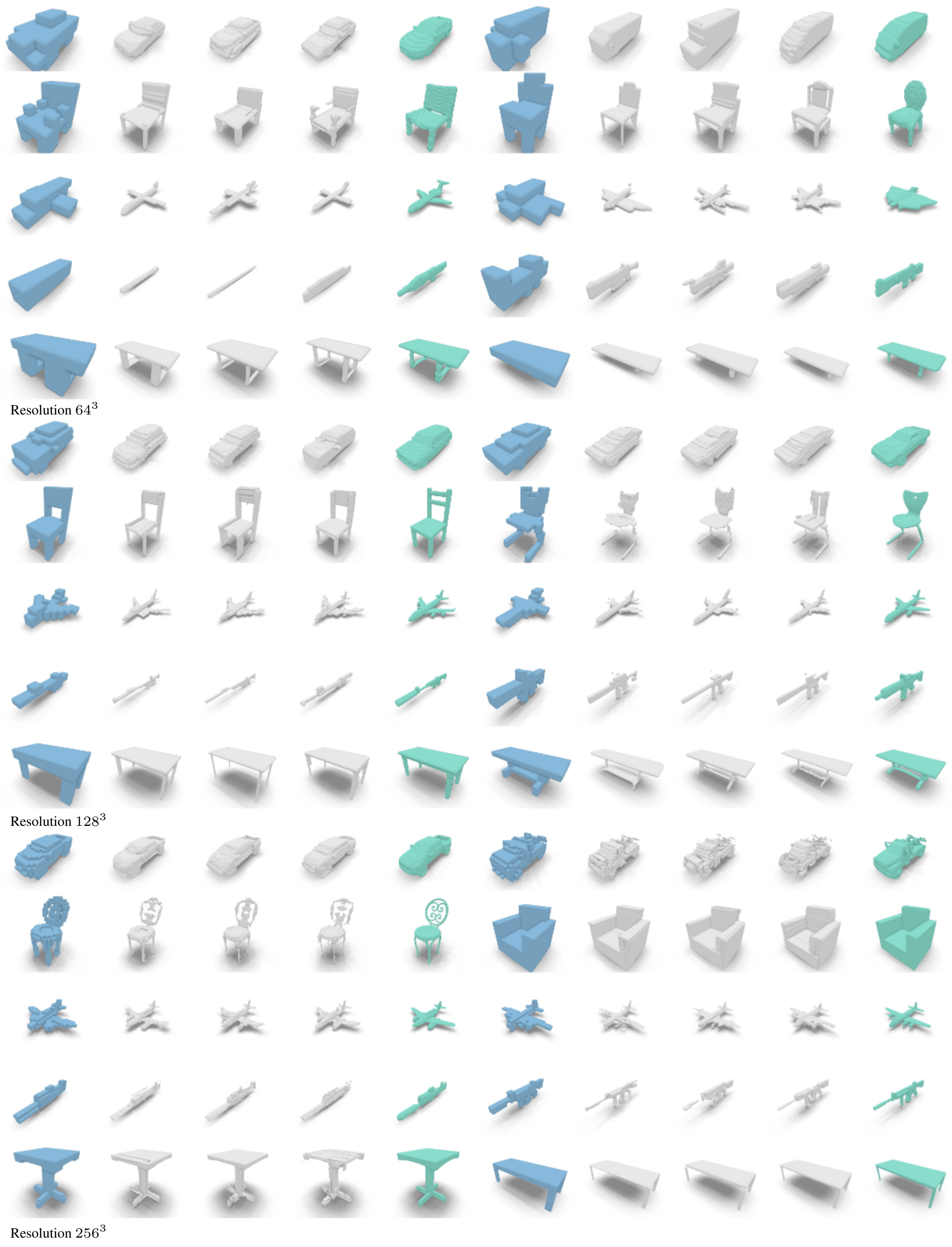
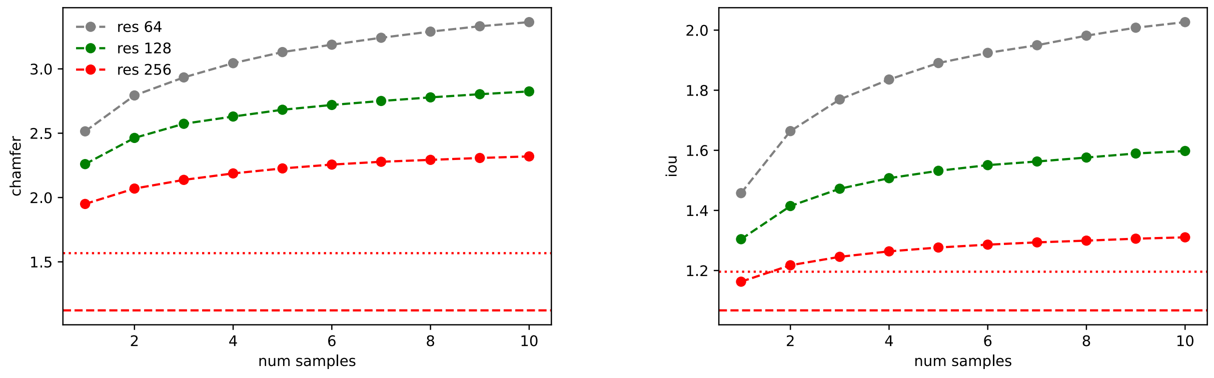


Figure 10. Conditional samples



(a) Chamfer Distance

(b) Intersection over Union

Figure 11. Relative improvements to the baseline, evaluated for the chair subset of ShapeNet. The dashed line denotes OccNet and the dotted line ConvONet, which both only are evaluated with an input resolution of 32 similar to our network, that outputs a resolution of 256.

D. Evaluation Superresolution

Unfortunately we cannot do a direct comparison against OccNet (Mescheder *et al*. [18]) and ConvONet (Peng *et al*. [23]), as they work on different data. However, we both compare against the same baseline (the input voxelization), so we can compare the relative improvements to this baseline. This is however still not a fair comparison due to several reasons. For our method we use a higher resolution voxelization as ground truth instead of the original meshes (like OccNet and ConvONet do), both for training and evaluation. Furthermore, our input voxelization is defined differently, as for us a voxel is ”full”, if it contains any geometry at all (usually a voxel is full if its center is within the shape). Lastly we use different train/test splits.