This CVPR workshop paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the accepted version;
the final published version of the proceedings is available on IEEE Xplore.

Octree Transformer: Autoregressive 3D Shape Generation on Hierarchically Structured Sequences

Moritz Ibing Gregor Kobsik Leif Kobbelt
Visual Computing Institute, RWTH Aachen University

Abstract

Autoregressive models have proven to be very powerful in NLP text generation tasks and lately have gained popularity for image generation as well. However, they have seen limited use for the synthesis of 3D shapes so far. This is mainly due to the lack of a straightforward way to linearize 3D data as well as to scaling problems with the length of the resulting sequences when describing complex shapes. In this work we address both of these problems. We use octrees as a compact hierarchical shape representation that can be sequentialized by traversal ordering. Moreover, we introduce an adaptive compression scheme, that significantly reduces sequence lengths and thus enables their effective generation with a transformer, while still allowing fully autoregressive sampling and parallel training. We demonstrate the performance of our model by performing superresolution and comparing against the state-of-the-art in shape generation.

1. Introduction

Autoregressive models have become the standard paradigm for generating texts and have gained popularity for image generation as well. This is because they can explicitly model the likelihood of the data and thus can be trained to maximize it in a stable manner, a big benefit compared to *e.g.* GANs where training is generally fickle.

A problem with applying this family of models to other fields is that they can only generate sequences, as each generated element depends on all previous ones. This leads to issues when sequences become very long, as is the case *e.g.* with images. When using recurrent networks, the training and sampling processes take a lot of time, as it can be done only sequentially. At least for the training this can be alleviated by the use of parallelizable models like CNNs [32, 33], but here the receptive field is limited, meaning the generation of a new element can only depend on part of the previous sequence.
The advent of transformers [23, 35] somewhat alleviated this problem, as they can be trained in parallel and deal with much larger sequence lengths than CNNs. However, they cannot process arbitrary sequences, as the memory consumption grows quadratically with the length. There is ongoing research to reduce this memory requirement by approximating the attention matrix [31], but these approaches are agnostic to the underlying structure of the sequence.
When generating 3D data these problems are even more pronounced, as the trivial approach of sequencing a voxel grid (similar to images) would lead to a sequence length that grows cubically with resolution and quickly become intractable already at low resolutions. Thus, transformers have so far been seldomly adapted for the generation of 3D shapes [21, 29, 41].

In our approach we do generate voxel grids, but to handle the cubic memory complexity, we use an octree representation, where the sequence length grows roughly quadratically with the resolution for natural shapes. In the worst ("fractal") case, where even in the finest resolution every voxel is near the shape boundary, the octree does not achieve any memory reduction at all (we even increase the memory by a factor of 4/3, due to the overhead of saving the tree), but in practice we never encountered such a case. However, even a sequenced octree would still be too large to be processed with a transformer, requiring us to compress it further.
Compressing a sequence for transformer-based generation is not trivial, as in order to work on shorter sequences we not only need to define a compression method but we have to be able to expand this compacted sequence again in a fully autoregressive manner. This means we need to make sure, that not only the compressed embeddings depend on each other, but a token generated as part of a block in an expansion has to depend both on elements inside and outside of its block.
For this we make use of the octree structure, summarizing different subtrees with the help of convolutions. When generating new nodes of the tree, we again use convolutions for decompression, taking care, that the process stays fully autoregressive, while still being parallelizable and memory efficient.

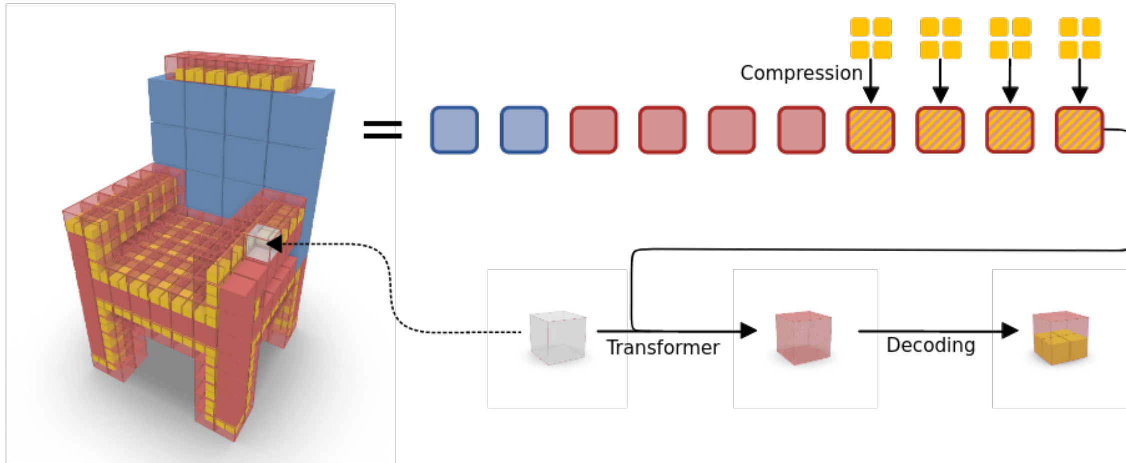


Figure 1. Our general approach: The given shape is represented as an octree (left) which we sequence, while compressing subtrees representing fine geometry (top right). Given a partial (compressed) sequence a transformer predicts the next token which is then decoded back to voxels.

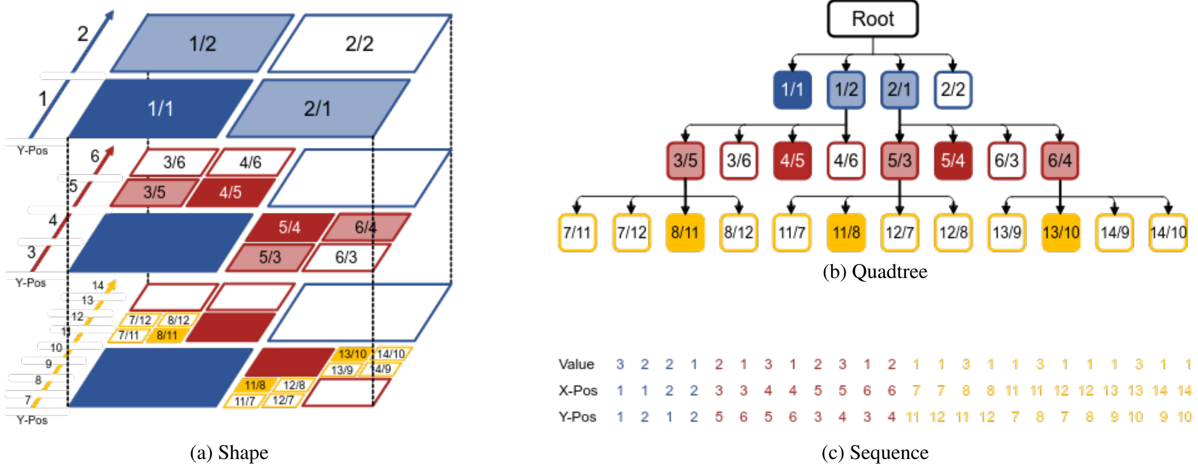
The use of an autoregessive setup easily allows us to condition the generation process on class labels, whereas the octree encoding enables not only a structured compression scheme, but also the straightforward implementation of voxel superresolution. The earlier parts of an octree sequence represent a coarse shape representation, while finer details are encoded in the later parts, thus superresolution can be formulated as sequence continuation.

2. Related Work

Transformer Autoregressive models in general and transformers in particular have often been used in the field of natural language processing *e.g.* for text generation [3, 25], as they are well suited for data that comes in the form of token sequences. More recently they have been applied to the field of vision as well to generate images. Early models here include PixelRNN, PixelCNN and the image transformer [23, 32, 33]. When using transformers to encode images Dosovitskiy *et al.* [8] summarize blocks to reduce the sequence length and Wu *et al.* [39] introduce convolutions for this compression. However, these are only employed on the encoding side and not in a generative context. Closest to our approach come van den Oord *et al.* [34] and Esser *et al.* [9] who compress an image by convolutions into a discrete set of codewords, which then can be generated in autoregressive fashion. However, here the compression can be seen as a preprocessing step and is not part of the generative model itself.

Tree structures Octrees [17] are a hierarchical spatial data structure that reduces the cubic memory complexity of voxel grids for storing 3D shapes, by recursively partitioning the space close to the surface, while saving information at a coarse resolution where it suffices. A downside is the higher complexity when implementing operations such as convolutions [26, 37], making their usage in neural network based processing relatively seldom. For generating 3D shapes with the help of octrees we are only aware of few methods, who all define convolutions on octrees and are either deterministic or based on VAEs [10, 30, 38]. This formulation does not allow autoregressive generation as in our case. Another line of work [15, 20] uses trees with semantic meaning (each node represents a coherent part) to encode shapes. The geometry within a node then of course needs to be encoded as well (*e.g.* as a point cloud). Thus, contrary to octrees the tree is not the actual geometry representation.

3D generation Different neural networks have been developed to generate all kinds of 3D shape representations: point clouds [1, 14, 22], voxels [2, 28, 40], functions [7] as well as those functions arranged in grid structures [11, 16]. However, all of these employ latent variable models like VAEs or GANs to generate novel shapes. The realm of autoregressive models is slowly emerging. PointGrow [29] and PolyGen [21] generate point clouds and meshes respectively, using architectures based mainly on self-attention. However, as they do not use any sequence compression, they are limited in the size of point clouds or meshes, that they can represent. Recently, VQVAEs [34] have been adapted to 3D, where the codewords represent implicit functions in a coarse [19, 41] or irregular [42] 3D grid. The models use this representation for shape completion or generation, by applying a transformer to predict the codebook ids. However, they do not use any hierarchical structure.



(a) Shape

(b) Quadtree

(c) Sequence

Figure 2. An example of our octree encoding (for simplicity we depict the encoding for a 2D quadtree). Position values are represented as X/Y for the x- and y-coordinates. Colors are used to indicate the depth level. Filled cells represent full (value=3), white cells empty (value=1) and transparent (value=2) cells mixed voxels. Below are the resulting value and positional sequences.

3. Octree Transformer

Our network consists of three stages (Fig. 1): The encoding stage takes as input a sequenced octree (Sec. 2) and compresses it into a shorter sequence of latent vectors with an increasing compression factor for finer octree levels (Sec. 3.2). This sequence is used, to train a standard generative transformer decoder. We will not describe this transformer and rather refer to to Vaswani *et al.* [35]. The generated sequence of latent vectors produced by the transformer is then uncompressed and decoded into a sequence of octree nodes (Sec. 3.3).

3.1. Octree Sequencing

To construct an octree from a binary voxel grid, we start by setting up the bounding cube as the root cell. We then recursively subdivide a cell whenever it contains both empty and full voxels, splitting it into 8 cubes, called the cells "children". This process is stopped when all cell's contain only empty or full voxels (this happens at the latest, when we have reached the resolution of the original voxel grid). Each cell in the tree can thus be either empty, full or mixed (only intermediate nodes).

Next, this octree needs to be linearized into a sequence of tokens, from which the octree can be reconstructed. In the following we will discuss how we approach this problem. All examples in this section and the following will be on quadtrees, the 2D variant of octrees, for easier visualization. Figure. 2b shows the tree structure together with all the information we need from our octree.

Simply enumerating the values of each cell (empty (1), mixed (2), full (3)) in breadth-first manner would give us a sequence, that fully characterizes the octree and allows its reconstruction (since we know, that only mixed cells have children and these have exactly 8, we can infer exactly how many cells we have per layer). However, this way the spatial location for each cell can only be inferred from global context. Instead, we add a spatial encoding, that is unique over all depth levels. For this we enumerate all cells along each dimension individually from coarse to fine (Fig. 2a), resulting in an unique id for each dimension. This spatial encoding replaces the 1D positional encoding usually employed, when working with transformers. Together with the value of the cell c we thus have 4 IDs (3 in 2D). For each possible value of these we learn a discrete embedding v, p_x, p_y, p_z, which we save for every cell: $v(c), p_x(c), p_y(c), p_z(c)$. Note that the positional encodings do not need to be predicted by the generative model as they can be inferred from the values themselves.

As our positional encoding depends on the spatial location of a cell and not the position of its token in the sequence, it is not possible to infer the encoding of the next token without global context. We therefore embed each token as the sum of its own value and positional encoding plus its successor's positional encoding: $e(c_i) = v(c_i) + p_x(c_i) + p_y(c_i) + p_z(c_i) + p_x(c_{i+1}) + p_y(c_{i+1}) + p_z(c_{i+1})$. This way when sampling a new token, the information of its position is easy to retrieve from its predecessor. The necessary context for this is always available, as the octree is generated from the root downwards. We do not need an end-of-sequence token as we always know how many tokens are needed to generate a valid octree, and can stop the sampling process when this number is reached. If we reach the maximum resolution and still have mixed tokens as leaves, those are treated as full cells.

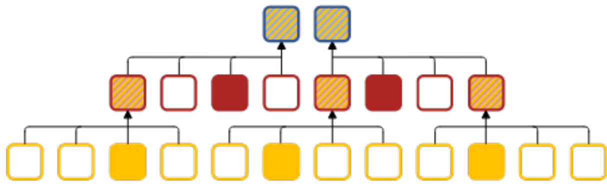


Figure 3. By compressing siblings we can achieve compression rates of up to 8 (4 in the quadtree example). For higher compression rates, we consider cousins (of higher order) by compression on the parent level. When compressing parent nodes we replace mixed tokens, by the representation of their children. For tokens that do not have any children (full/empty) we use their original embedding. The result of the compression is a latent vector for each compressed subtree.

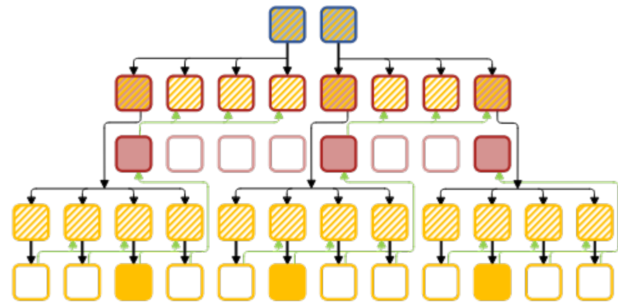


Figure 4. When generating a subtree from a sequence of compressed embeddings, we need to both upsample information from the embedding vectors (black arrows), as well as pass information from all previously generated tokens in the same subtree (green arrows).

3.2. Sequence Compression

Although we make use of an octree encoding to circumvent the cubic complexity of voxel grids, we still encounter prohibitively large sequence length when representing shapes in a high resolution. Therefore, we need to further compress our sequence in order to be able to fit the transformer's attention computation into memory. This compression should not treat our sequence simply as "flat" 1D data, but instead take the hierarchical octree structure into account. On the one hand this means, that not all tokens of the sequence should be treated equally, as cells from early depth layers are responsible for bigger parts of the shape (and are few), whereas later cells only encode details (and are many). On the other hand the compression itself should take the structure of the octree into account and only compress cells together, that are spatially close, but not necessarily close in the sequence.
For compression rates of 2,4,8, this means that we combine the features of siblings in a tree. This can be done easily with strided convolutions (stride and kernel size equal to the compression factor).
For higher compression rates we encode entire subtrees into a single latent vector. For this we first compress all siblings and place their combined feature vector at the parent node. If this parent does have any siblings without children of their own, these will keep their own feature vector. We then repeat the compression at the parent generation (Fig. 3).

3.3. Sequence Decoding

For generation we train the transformer decoder on compressed sequences, so that all feature vectors only depend on vectors positioned earlier in the sequence (when generating novel shapes, this sequence will be constructed one by one). However, to obtain a valid octree, we still have to undo the compression, and obtain logits from which to sample our cell information.
When no compression has been applied a linear layer with softmax non-linearity is enough to obtain logits. When compression has been applied we have to undo it, using the reverse operations of what has been presented before. Just doing a simple upsampling is not sufficient, as we would loose the autoregressive property of our generation. The probabilities of tokens generated this way would depend only on all previous tokens *outside of the compressed subtree*. These dependencies are taken care of by the transformer. However, within the block the probabilities (of siblings and cousins) would not depend on each other. To model these dependencies efficiently we again make use of the octree structure (Fig. 4).

More concretely, we need to combine information from two different sources. Information about everything outside of the current subtree is encoded in the latent vector at the root and needs to be passed down to the leaves (Fig. 4 black arrows), whereas information about already generated leaves within the tree needs to be passed to successor leaves on the same level (Fig. 4 green arrows).
Let us start with the simple case of a compression factor $c \leq 8$. We first need to undo the compression, by upsampling the latent vector, which simply can be done with a transposed convolution of the same stride and kernel size as was used for the compression, resulting in blocks of size c. Next, we need to model the dependency within these blocks. Each entry should depend on all entries occurring earlier in the sequence (within the same block). This can be done with what we call a masked block convolution. This passes information from one node to all successors within the same block. For this we just need to accumulate the respective latent vectors, which can be efficiently modeled with convolutions.
This operation can be seen as a sort of attention, where the attention weights are not input dependent, but instead only

Res	Octree tokens	0/4	0/8	1/8	2/8
2	8	2	1	-	-
4	64	16	8	1	-
8	224	56	28	8	1
16	816	204	102	28	8
32	3352	838	419	102	28
64	13264	3316	1658	419	102
128	48530	12132	6066	1658	419
256	144096	36024	18012	6066	1658

Table 1. Sequence lengths achieved with different compression schemes. The Octree tokens are from the 90 percentile per depth layer. The compression is denoted as a/b, where a indicates the depth of the subtrees we collapse and b how many tokens at the root level are encoded together.

depend on the position. This is sensible in our case, as the position of our information (*where* is the filled/mixed/empty cell positioned) contains more information than the value (which is only one of three options).

When we have compression factors higher than 8, we have to apply this strategy recursively. On the one hand the latent vector at the root is upsampled several times, taking the position of mixed tokens into account (only at mixed tokens do we upsample further, as only mixed tokens have children). On the other hand information about leaf nodes needs to be propagated further. For this we pass it up a level (to the parent nodes) and there repeat the block convolution. Again, information is only passed to nodes later in the sequence. Nodes in the parent generation that do not have any children, do not contribute. This information is then added to the corresponding latent vectors in the same level and passed down again by the already described transposed convolution.

Note that we generally could model both the sequence encoding and decoding with additional smaller transformer networks as well. However, this would lead to a much larger memory consumption, counteracting the goal of the compression, which is why we have opted for this more lightweight approach.

Note that our compression algorithm is fully differentiable and we therefore can train the encoding, transformer and decoder together end to end.

4. Evaluation

All evaluations are performed on the ShapeNet Core dataset (v1) [4]. We use the training split from Chen and Zhang [7] (80% training and 20% testing) and randomly pick another 5% from the training set for validation. The voxelized models are obtained from Hane *et al*. [10].

All experiments are done on a single RTX2080Ti. Our transformer decoder consists of 8 layers, uses 8 heads and a latent dimension of 256. We trained all models using the Adam optimizer [13] and a training rate of $3 \cdot 10^{-5}$. Furthermore the first 10% of the training steps are used for warm-up, where we linearly increase the learning rate from 0. We trained with a batch size of 1 for either 500 epochs on a single class or 100 epochs for class conditional models on the entire dataset. To increase sampling speed we used the fast transformer framework [12, 36]. Even though our compression scheme is fully parallelizable and decreases the effective sequence length for the transformer, during sampling we still need to sample each token of the uncompressed sequence one after the other. We do not need to apply the transformer for each token, but still this can lead to long sampling times (several minutes) for complex shapes at high resolution. We sampled with a temperature of 0.8.

As we compare against GANs, which do not allow to compute a log-likelihood for test shapes, we adopt their evaluation scheme. For this we sampled a set of shapes 5 times the size of the test set, used the Light Field Descriptor [5] to compare single shapes and coverage (COV) and minimum matching distance (MMD) [1] as well as edge count difference (ECD) [6, 11] to compare the respective sets.

Data Augmentation In all experiments we use data augmentation inspired by PolyGen [21], randomly scaling the shape independently along the different axes with a piecewise linear warp. Note that even small changes in the voxel grid can lead to big changes in the octree sequence, making this augmentation very important as shown in Tab. 3.

Compression In Tab. 1 we show which compression we achieve with a collection of different compression factors. From these numbers we can infer how much we need to compress in order to fit the sequence into memory, and what compression factor might be reasonable for what layer. In Tab. 3 we show different compression schemes. After generating the octree, we filter out all sequences, that would have a length of more than 3300 after compression, in order to fit the training onto a single GPU.

Loss Function Autoregressive models are usually trained by directly minimizing the negative log-likelihood, and we do the same here. However, one might think that in an octree representation getting the earlier elements in the sequence correct is much more important than for later elements, as they cover bigger regions. We therefore tried weighting the loss function according to the depth layer an octree element resides in. Starting with a weight of one for the first layer and then decrease by a factor of α for each subsequent layer. In Tab. 2 we evaluate different factors. In order to be able to compare fairly, we normalize the weights to an average of one over a shape. To compare the results, we evaluate COV, MMD and ECD. As can be seen,

factor α	↑COV	↓MMD	↓ECD
1	76.47	2958	1889
0.5	75.00	2921	1886
0.25	74.78	2933	1767
0.125	76.03	3011	2057

Table 2. Evaluation of different loss functions

model	↓bits/token
uniform random	1.5850
sliding window	0.3056
full model	0.0727
- augmentation	0.1227
+ later compression A	0.0762
+ stronger compression B	0.0864

Table 3. Evaluating different design decisions. To put the numbers into context we report a baseline of uniform random sampling as well as results for a transformer with sliding-window attention.

differences are only marginal. For simplicity we therefore decided against any weighting.

Ablation In this section we motivate several of the design decisions we made. For this we evaluate different configurations of our model and report the negative log-likelihood after training in Tab. 3. For runtime reasons this ablation is done on the chair dataset for a resolution of 64. We compare against a simple baseline, where we do not use any compression, but instead limit the attention to a local neighbourhood [27,41], so that the computation fits into memory.

As can be seen easily, data augmentation contributes a big factor in improving our results as it counteracts overfitting. In fact using data augmentation, we did not observe any overfitting at all, suggesting that the model size could be further increased given enough memory.

Lastly, we want to compare several choices regarding our compression scheme. As mentioned, the sequence can be compressed with a different factor at each depth level. Trying out every possible combination of compression is not feasible, therefore we restricted to three intuitive possibilities. As coarser levels are more important for reconstruction they should generally be compressed less than higher levels. However, at what rate the compression should increase is unclear. We evaluate two different schemes. In our baseline we increase the compression more or less linearly using factors of $(0/1, 0/1, 0/2, 0/4, 0/8, 1/4)$. In another experiment, we evaluate a "steeper" compression A. leaving earlier layers unchanged, and focusing the compression on the finer scales $A = (0/1, 0/1, 0/1, 0/1, 0/8, 1/8)$. The idea was, that this way we might lose information in the details, but keep the coarse structure better. Both compression schemes lead to roughly similar sequence lengths. As can be seen the former scheme is better able to reduce the loss, suggesting that a linear increase in compression is desirable (Tab. 3).

Furthermore, we want to check if the compression scheme has any adverse effect at all, or if maybe a shorter sequence is even beneficial for training. For this we tested a stronger compression scheme $B = (0/1, 0/1, 0/4, 0/8, 1/4, 1/8)$ leading to sequences of roughly half the size. As can be seen, this leads to worse results. We conclude that the compression should be chosen in a way that leaves the sequence as long as memory permits.

Comparison After having found the best configuration for our model, we want to compare it against other generative approaches. For this there are mostly GANs available. As their evaluation is carried out on a resolution of 64, we do the same, using the evaluation criteria described earlier in this section. The numbers for previous approaches are taken from Chen and Zhang [7] and Ibing *et al.* [11]. Although we do not fully reach the state of the art, our method is competitive with the presented GANs (Tab. 4).

We would have liked to compare against autoregressive approaches as well. But as the only available options PointGrow [29] and PolyGen [21] work on different shape representations have different input requirements and evaluated their models with different metrics on different subsets of the ShapeNet dataset, we considered a fair comparison impossible.

In Fig. 5 we compare shapes generated at different resolutions sampled from class conditional models with previous approaches. For each resolution we trained a different model (although in principle higher resolution models can generate shapes in lower resolutions as well). To fit the models into memory we used different compression strategies. The model for resolution 64 is the same as presented earlier as our baseline. For a resolution of 128 we use a compression of $(0/1, 0/1, 0/4, 0/8, 1/4, 1/8, 2/4)$ and for resolution 256 $(0/1, 0/1, 0/4, 0/8, 1/4, 1/8, 2/4, 2/8)$. We achieve our best results at a resolution of 64 where we can generate more diverse and less noisy samples than 3DGAN, the only other voxel based method. We then see a decrease in quality at higher resolutions. Although our level of detail is comparable to the function based approaches (IMGAN, Grid-IMGAN), we do not reach their smoothness, even though our resolution is higher, as the voxels we add are noisy. This might be due to the higher compression factor that is necessarily employed, as well as due to the limited size of our model (a bigger network with more heads might be able to focus better on the coarse shape as well as on fine details). On all scales our network is able to correctly model the symmetries inherent in the shapes. Generally we have not noticed overfitting in any

		Plane	Car	Chair	Rifle	Table	Average
↑COV %	3DGAN [40]		12.13	25.07	62.32	18.80	
	PC-GAN [1]	73.55	61.40	70.06	61.47	77.50	68.80
	IM-GAN [7]	70.33	69.33	75.44	65.26	86.43	73.36
	Grid IM-GAN [11]	81.58	80.67	82.08	81.47	86.19	82.80
	Octree Transformer	73.05	60.26	76.47	60.21	80.55	70.11
	Train Set	85.04	85.67	84.73	84.00	87.13	85.13
↓MMD	3DGAN		1993	4365	4476	5208	
	PC-GAN	3737	1360	3143	3891	2822	2991
	IM-GAN	3689	1287	2893	3760	2527	2831
	Grid IM-GAN	3226	1225	2768	3366	2396	2607
	Octree Transformer	3664	1363	2958	3582	2496	2813
	Train Set	2225	984	2317	3085	2066	2135
↓ECD	3DGAN		28855	26279	6495	32116	
	PC-GAN						
	IM-GAN	6543	20606	2553	3288	1018	
	Grid IM-GAN	355	1062	144	94	188	
	Octree Transformer	2573	8563	1889	1835	1098	
	Train Set	1	11	1	2	5	

Table 4. Comparison of different generative methods synthesizing 3D shapes. Results from previous methods are taken from [7] and [11]. Each subcategory was evaluated on a shape resolution of 64^3 against a voxelized test set of the same resolution. 3DGAN was not trained on the plane subcategory and no ECD values are reported for PC-GAN, thus some entries remain blank. For ECD no averages are reported, as values for different datasets are not comparable.

of our runs despite not using any regularization techniques (only data augmentation) and therefore conclude that the network capacity could still be significantly increased. Furthermore, our hyper-parameters were optimized for a resolution of 64.

Superresolution To perform superresolution, we provide the model with truncated sequences from the test set, describing the shape only up to a given resolution and let our model complete these sequences. For a quantitative evaluation we use similar metrics as Mescheder *et al*. [18] and Peng *et al*. [24]. We report the chamfer distance as well as the intersection over union between the sampled shape and the ground truth (Fig. 7). As our model is probabilistic, we show how the result improves when increasing the number of samples, by reporting the best value out of the set. As a baseline we compare the input (low resolution) voxelization to the ground truth. A direct comparison to Mescheder *et al*. [18] and Peng *et al*. [24] is unfortunately not possible, as they train and evaluate on different data. Again, we show results at three different resolutions using the same trained models as in the last task. In all examples we increased the resolution by a factor of 8 (3 levels). In Fig. 6 we show the input shape, some generated samples, and the original shape from the test set. With higher resolutions, we see the same quality problems as in the unconditional case. Furthermore, the diversity decreases (compared to *e.g*. resolution 64, where a wide range of valid shapes are generated), which can be explained by the more restrictive precondition and the comparatively small dataset.

5. Conclusion

In this work we introduced a new deep generative model, based on the transformer architecture, for the generation of 3D shapes in the form of octrees. In order to deal with long sequences we developed a compression scheme, that significantly reduces sequence length, while still allowing fully autoregressive generation and might be of interest in other fields that deal with long sequence lengths, like the domain of image generation. Our model can be applied for the generation of novel shapes, as well as for increasing the resolution of existing ones, as it naturally supports a coarse-to-fine generation process, with easily adjustable level of detail.

As autoregressive models are widely used in text generation etc. one can now pose 3D problems in this framing, as we demonstrated for the case of superresolution, which was formulated similar to next word prediction. This should be interesting for further tasks, like treating shape completion as a translation problem (both partial and complete shapes

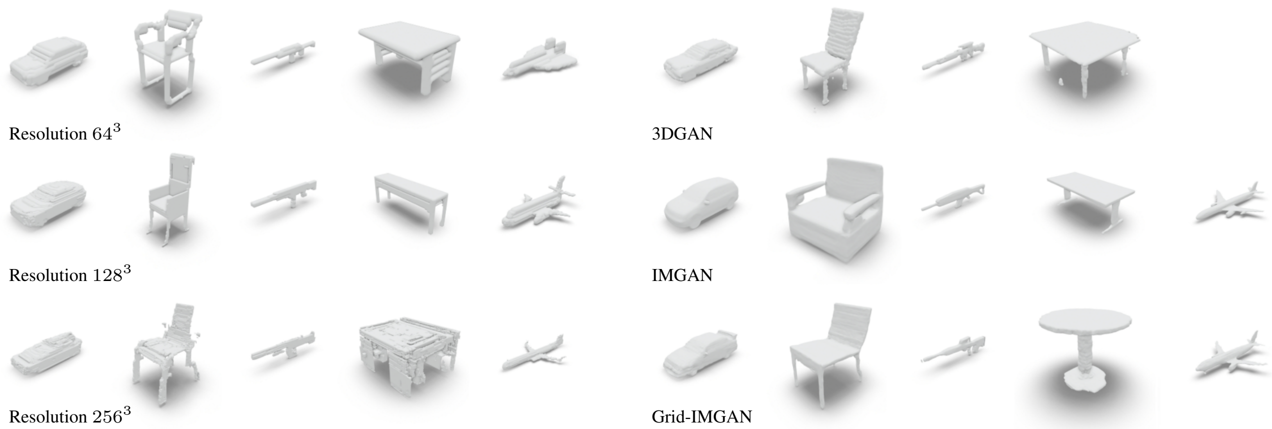


Figure 5. Samples synthesized by our class conditional Octree Transformer at different resolutions and comparable approaches. For 3DGAN the plane model was omitted, as the model was not trained on the plane dataset.

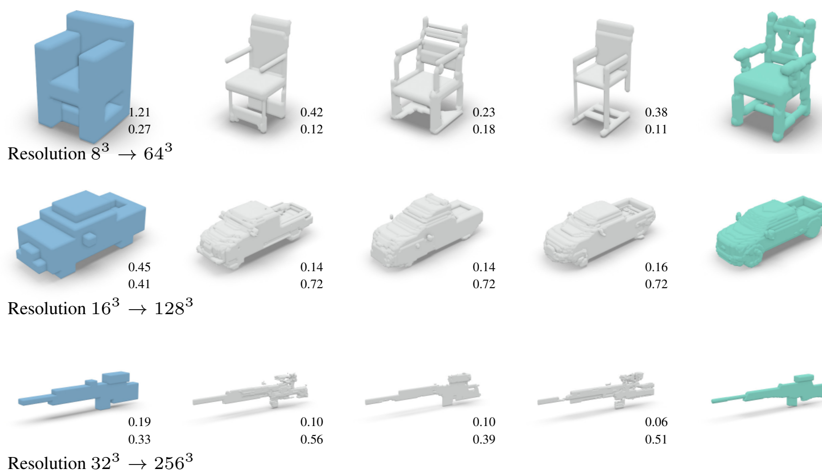
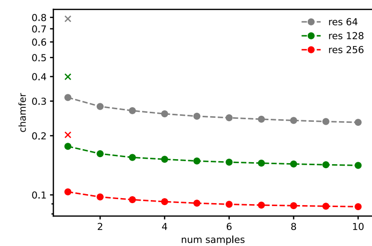


Figure 6. Superresolution samples synthesized by the Octree Transformer at different resolutions. Our model computes an upsampling of factor eight. The leftmost shape is the preconditioning and the rightmost shape the actual model from the dataset. In grey we show three different upsamplings. We report chamfer distance (upper number) and IoU (lower number) to the ground truth shape.



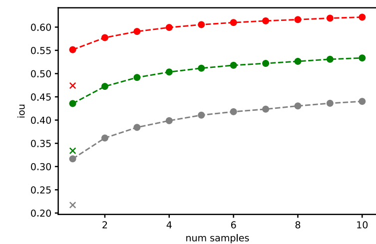


Figure 7. Evaluating superresolution on the chair subset (Chamfer distance and Intersection over Union). x denotes baseline.

can be encoded akin to sentences).

Even though our method greatly reduces the size of the sequence the transformer needs to process to generate shapes, we are still mainly limited by its length and the resulting memory constraints (small model sizes, high compression factors, slow sampling). An interesting direction of research and possible solution to these problems would be an increase of the complexity of geometry represented by a single token. Currently we only use three tokens (empty, mixed, full) to encode the octree. By adding tokens with additional information, we could transfer complexity from the sequence length to the token variety (language models usually deal with thousands of different tokens). In these tokens we could encode voxel arrangements or even go a step further and store implicit functions in the form of (quantized) latent vectors, thus improving the smoothness of our results as well.

Acknowledgements This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) within the Gottfried Wilhelm Leibniz programme and project number 449823330.

References

[1] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas. Learning representations and generative models for 3d point

clouds. In *International conference on machine learning*, pages 40–49. PMLR, 2018. 2, 5, 7

[2] A. Brock, T. Lim, J. M. Ritchie, and N. Weston. Generative and discriminative voxel modeling with convolutional neural networks. *arXiv preprint arXiv:1608.04236*, 2016. 2

[3] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020. 2

[4] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015. 5

[5] D.-Y. Chen, X.-P. Tian, Y.-T. Shen, and M. Ouhyoung. On visual similarity based 3d model retrieval. In *Computer graphics forum*. Wiley Online Library, 2003. 5

[6] H. Chen and J. H. Friedman. A new graph-based two-sample test for multivariate and object data. *Journal of the American statistical association*, 112(517):397–409, 2017. 5

[7] Z. Chen and H. Zhang. Learning implicit fields for generative shape modeling. In *CVPR*, 2019. 2, 5, 6, 7

[8] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021. 2

[9] P. Esser, R. Rombach, and B. Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12873–12883, 2021. 2

[10] C. Häne, S. Tulsiani, and J. Malik. Hierarchical surface prediction for 3d object reconstruction. In *2017 International Conference on 3D Vision (3DV)*, pages 412–420. IEEE, 2017. 2, 5

[11] M. Ibing, I. Lim, and L. Kobbelt. 3d shape generation with grid-based implicit functions. In *CVPR*, 2021. 2, 5, 6, 7

[12] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020. 5

[13] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *ICLR*, 2015. 5

[14] C. Li, M. Zaheer, Y. Zhang, B. Póczos, and R. Salakhutdinov. Point cloud GAN. In *Deep Generative Models for Highly Structured Data, ICLR 2019 Workshop*, 2019. 2

[15] J. Li, K. Xu, S. Chaudhuri, E. Yumer, H. Zhang, and L. Guibas. Grass: Generative recursive autoencoders for shape structures. *ACM Transactions on Graphics (TOG)*, 36(4):1–14, 2017. 2

[16] I. Lim, M. Ibing, and L. Kobbelt. A convolutional decoder for point clouds using adaptive instance normalization. *Comput. Graph. Forum*, 38(5):99–108, 2019. 2

[17] D. Meagher. Geometric modeling using octree encoding. *Comput. Graph. Image Process.*, 19(1):85, 1982. 2

[18] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. Occupancy networks: Learning 3d reconstruction in function space. *CVPR*, 2019. 7

[19] P. Mittal, Y.-C. Cheng, M. Singh, and S. Tulsiani. AutoSDF: Shape priors for 3d completion, reconstruction and generation. In *CVPR*, 2022. 2

[20] K. Mo, P. Guerrero, L. Yi, H. Su, P. Wonka, N. Mitra, and L. J. Guibas. Structurenet: Hierarchical graph networks for 3d shape generation. *ACM Trans. Graph.*, 2019. 2

[21] C. Nash, Y. Ganin, S. M. A. Eslami, and P. W. Battaglia. Polygen: An autoregressive generative model of 3d meshes. In *Int. Conf. on Machine Learning,*, pages 7220–7229, 2020. 1, 2, 5, 6

[22] C. Nash and C. K. I. Williams. The shape variational autoencoder: A deep generative model of part-segmented 3d objects. *Comput. Graph. Forum*, 36(5):1–12, 2017. 2

[23] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran. Image transformer. In *Int. Conf. on Machine Learning,*, pages 4052–4061, 2018. 1, 2

[24] S. Peng, M. Niemayer, L. Mescheder, M. Pollefeys, and A. Geiger. Convolutional occupancy networks. *ECCV*, 2020. 7

[25] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019. 2

[26] G. Riegler, A. Osman Ulusoy, and A. Geiger. Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3577–3586, 2017. 2

[27] A. Roy, M. Saffar, A. Vaswani, and D. Grangier. Efficient content-based sparse attention with routing transformers. *Transactions of the Association for Computational Linguistics*, 9:53–68, 2021. 6

[28] A. Sharma, O. Grau, and M. Fritz. Vconv-dae: Deep volumetric shape learning without object labels. In *ECCV*, pages 236–250. Springer, 2016. 2

[29] Y. Sun, Y. Wang, Z. Liu, J. E. Siegel, and S. E. Sarma. Pointgrow: Autoregressively learned point cloud generation with self-attention. In *IEEE Winter Conference on Applications of Computer Vision, WACV 2020, Snowmass Village, CO, USA, March 1-5, 2020*, pages 61–70, 2020. 1, 2, 6

[30] M. Tatarchenko, A. Dosovitskiy, and T. Brox. Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2088–2096, 2017. 2

[31] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler. Efficient transformers: A survey. *ACM Computing Surveys*, 55(6):1–28, 2022. 1

[32] A. van den Oord, N. Kalchbrenner, L. Espeholt, K. Kavukcuoglu, O. Vinyals, and A. Graves. Conditional image generation with pixelcnn decoders. In *NeurIPS*, pages 4790–4798, 2016. 1, 2

[33] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. In *Int. Conf. on Machine Learning,*, pages 1747–1756, 2016. 1, 2

[34] A. van den Oord, O. Vinyals, and K. Kavukcuoglu. Neural discrete representation learning. In *NeurIPS*, pages 6306–6315, 2017. 2

[35] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017. 1, 3

[36] A. Vyas, A. Katharopoulos, and F. Fleuret. Fast transformers with clustered attention. In *Proceedings of the International Conference on Neural Information Processing Systems (NeurIPS)*, 2020. 5

[37] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong. O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions on Graphics (TOG)*, 36(4):1–11, 2017. 2

[38] P.-S. Wang, C.-Y. Sun, Y. Liu, and X. Tong. Adaptive o-cnn: A patch-based deep representation of 3d shapes. *ACM Transactions on Graphics (TOG)*, 37(6):1–11, 2018. 2

[39] H. Wu, B. Xiao, N. Codella, M. Liu, X. Dai, L. Yuan, and L. Zhang. Cvt: Introducing convolutions to vision transformers. *CoRR*, 2021. 2

[40] J. Wu, C. Zhang, T. Xue, W. T. Freeman, and J. B. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *NeurIPS*, pages 82–90, 2016. 2, 7

[41] X. Yan, L. Lin, N. J. Mitra, D. Lischinski, D. Cohen-Or, and H. Huang. Shapeformer: Transformer-based shape completion via sparse representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6239–6249, 2022. 1, 2, 6

[42] B. Zhang, M. Nießner, and P. Wonka. 3DILG: Irregular latent grids for 3d generative modeling. In *Thirty-Sixth Conference on Neural Information Processing Systems*, 2022. 2