Appendix

Figure 11. Examples of data augmentation and randomized rendering conditions. For each input mesh we create 50 augmentations, and render each while varying lighting, camera and material properties.

A. Data Augmentation

For each input mesh from the ShapeNet dataset we create 50 augmented versions which are used during training (Figure 11). We start by normalizing the meshes such that the length of the long diagonal of the mesh bounding box is equal to 1. We then apply the following augmentations, performing the same bounding box normalization after each. All augmentations and mesh rendering are performed prior to vertex quantization.

Axis scaling. We scale each axis independently, uniformly sampling scaling factors s_x, s_y and s_z in the interval $[0.75, 1.25]$.

Piecewise linear warping. We define a continuous, piecewise linear warping function by dividing the interval $[0, 1]$ into 5 even sub-intervals, sampling gradients $g_1, \ldots, g_5$ for each sub-interval from a log-normal distribution with variance 0.5, and composing the segments. For x and y coordinates, we ensure the warping function is symmetric about zero, by reflecting a warping function with three sub-intervals on $[0, 0.5]$ about 0.5. This preserves symmetries in the data which are often present for these axes.

Planar mesh decimation. We use Blender's planar decimation modifier (https://docs.blender.org/manual/en/latest/modeling/modifiers/generate/decimate.html) to create n-gon meshes. This merges adjacent faces where the angle between surfaces is greater than a certain tolerance. Different tolerances result in meshes of different sizes with differing connectivity due to varying levels of decimation. We use this property for data augmentation and sample the tolerance degrees uniformly from the interval $[1, 20]$.

B. Rendering

We use Blender to create rendered images of the 3D meshes in order to train image-conditional models (Figure 11). We use Blender's Cycles (https://docs.blender.org/manual/en/latest/render/cycles/index.html) path-tracing renderer, and randomize the lighting, camera, and mesh materials. In all scenes we place the input meshes at the origin, scaled so that bounding boxes are 1m on the long diagonal.

Lighting. We use an 20W area light located 1.5m above the origin, with rectangle size 2.5m, and sample a number of 15W point lights uniformly from the range $[0, 1, \ldots, 10]$. We choose the location of each point light independently, sampling the x and y coordinates uniformly in the intervals $[-2, -0.75] \cup [0.75, 2]$, and sampling the z coordinate uniformly in the interval $[0.75, 2]$.

Camera. We position the camera at a distance d from the center of the mesh, where d is sampled uniformly from $[1.25, 1.5]$, at an elevation sampled between $[0, 1]$, and sample a rotation uniformly between $[0, 360]$. We sample a focal length for the camera in $[35, 36, \ldots, 50]$. We also sample a filter size (https://docs.blender.org/manual/en/latest/render/cycles/render_settings/film.html) in $[1.5, 2]$, which adds a small degree of blur.

Object materials. We found the ShapeNet materials and textures to be applied inconsistently across different examples when using Blender, and in many cases no textures loaded at all. Rather than use the inconsistent textures, we randomly generated materials for the 3D meshes, in order to produce a degree of visual variability. For each texture group in the mesh we sampled a new material. Materials were constructed by linking Blender nodes (https://docs.blender.org/manual/en/latest/render/shader_nodes/introduction.html#textures). In particular we use a noise shader with detail = 16, scale $= \sqrt{100 * u}, u \sim \mathcal{U}(0, 1)$, and scale draw from the interval $[0, 20]$. The noise shader is used as input to a color ramp node which interpolates between the input color, and white. The color ramp node then sets the color of a diffuse BSDF material https://docs.blender.org/manual/en/latest/render/shader_nodes/shader/diffuse.html, which is applied to faces within a texture group.

C. Transformer blocks

We use the improved Transformer variant with layer normalization moved inside the residual path, as in (Child et al., 2019; Parisotto et al., 2019). In particular we compose the Transformer blocks as follows:

$$\mathbf{R}^{(l)}_{\text{MMH}} = \text{MaskedMultiHead}(\text{LN}(\mathbf{H}^{(l-1)}_{\text{FC}})) \quad (12)$$

$$\mathbf{H}^{(l)}_{\text{MMH}} = \mathbf{H}^{(l-1)}_{\text{FC}} + \mathbf{R}^{(l)}_{\text{MMH}} \quad (13)$$

$$\mathbf{R}^{(l)}_{\text{FC}} = \text{Linear}(\text{ReLU}(\text{Linear}(\text{LN}(\mathbf{H}^{(l)}_{\text{MMH}})))) \quad (14)$$

$$\mathbf{H}^{(l)}_{\text{FC}} = \mathbf{H}^{(l)}_{\text{MMH}} + \mathbf{R}^{(l)}_{\text{FC}} \quad (15)$$

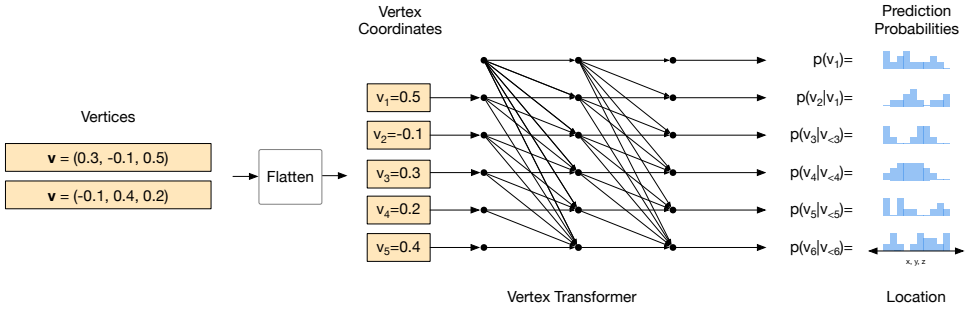

Figure 12. The vertex model is a masked Transformer decoder that takes as input a flattened sequence of vertex coordinates. The Transformer outputs discrete distributions over the individual coordinate locations, as well as the stopping token s. See Section 2.2 for a detailed description of the vertex model.

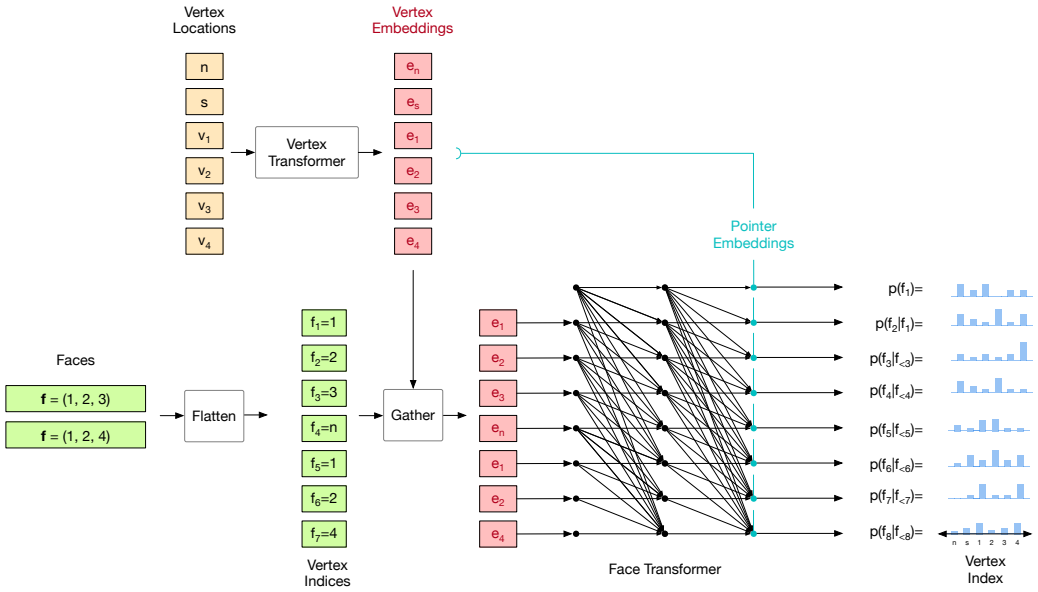

Figure 13. The face model operates on an input set of vertices, as well as the flattened vertex indices that describe the faces. The vertices as well as the new face token n and stopping token s are first embedded using an un-masked Transformer encoder (Vertex Transformer). A gather operation is then used to identify the embeddings associated with each vertex index. The index embeddings are processed with a masked Transformer decoder (Face Transformer) to output distributions over vertex indices at each step, as well as over the next-face token and the stopping token. The final layer of the Transformer outputs pointer embeddings which are compared to the vertex embeddings using a dot-product and then passed through a softmax to produce the desired distributions. See Section 2.3 for a detailed description of the face model and Figure 5 in particular for a detailed depiction of the pointer network mechanism.

Where $\mathbf{R}^{(l)}$ and $\mathbf{H}^{(l)}$ are residuals and intermediate representations in the l'th block, and the subscripts FC and MMH denote the outputs of fully connected and masked multi-head self-attention layers respectively. We apply dropout immediately following the ReLU activation as this performed well in initial experiments.

Conditional models. As described in Section 2.5 For global features like class identity, we project learned class embeddings to a vector that is added to the intermediate Transformer representations $\mathbf{H}_{\text{MMH}}$ following the self-attention layer in each block:

$$\mathbf{r}^{(l)}_{\text{global}} = \text{Linear}\left(\mathbf{h}_{\text{global}}\right) \tag{16}$$

$$\mathbf{H}^{(l)}_{\text{global}} = \mathbf{H}^{(l)}_{\text{MMH}} + \text{Broadcast}\left(\mathbf{r}^{(l)}_{\text{global}}\right) \tag{17}$$

For high dimensional inputs like images, or voxels, we jointly train a domain-appropriate encoder that outputs a sequence of context embeddings. The Transformer decoder performs cross-attention into the embedding sequence after the self-attention layer, as in the original machine translation

Transformer model:

$$\mathbf{R}_{\text{seq}}^{(l)} = \text{CrossMultiHead}\left(\mathbf{H}_{\text{MMH}}^{(l)}, \mathbf{H}_{\text{seq}}\right) \quad (18)$$

$$\mathbf{H}_{\text{seq}}^{(l)} = \mathbf{H}_{\text{MMH}}^{(l)} + \mathbf{R}_{\text{seq}}^{(l)} \quad (19)$$

The image and voxel encoders are both pre-activation resnets, with 2D and 3D convolutions respectively. The full architectures are described in Table 4.

D. AtlasNet

We use the same image and voxel-encoders (Table 4) as for the conditional PolyGen models. For consistency with the original method, we project the final feature maps to 1024 dimensions, before applying global average pooling to obtain a vector shape representation. As in the original method, the decoder is an MLP with 4 fully-connected layers of size 1024, 512, 256, 128 with ReLU non-linearities on the first three layers and tanh on the final output layer. The decoder takes the shape representation, as well as 2D points as input, and outputs a 3D vector. We use 25 patches, and train with the same optimization settings as PolyGen (Section 3) but for $5e5$ steps.

Chamfer distance. To evaluate the chamfer distance for AtlasNet models, we first generate a mesh by passing 2D triangulated meshes through each of the AtlasNet patch models as described in (Groueix et al., 2018). We then sample points on the resulting 3D mesh.

E. Alternative Vertex Models

In this section, we provide more details for the more efficient vertex model variants mentioned in Section 2.2.

In the first variant, instead of processing x, y and z coordinates in sequence we concatenate their embeddings together and pass them through a linear projection. This forms the input sequence for a 22-layer Transformer which we call *the torso*. Following (Salimans et al., 2017) we output the parameters of a mixture of 40 discretized logistics describing the joint distribution of a full 3D vertex. The main benefit of this model is that the self-attention is now performed for sequences which are 3 times shorter. This manifests in a much improved training time (see 2). Unfortunately, the speed-up comes at a price of significantly reduced performance. This may be because the underlying continuous components are not well suited to the peaky and multi-modal vertex distributions.

In the second variant we lift the parametric distribution assumption and use a MADE-style masked MLP (Germain et al., 2015) with 2 residual blocks to decode each output of a 18-layer torso h_n into a sequence of three conditional discrete distributions:

$$p(v_n|h_n) = p(z_n|h_n)p(y_n|z_n, h_n)p(x_n|z_n, y_n, h_n) \quad (20)$$

As expected, this change improves the test data likelihood while simultaneously increasing the computation cost. We notice that unlike the base model the MADE decoder has direct access only to the coordinate components within a single vertex and must rely on the output of the torso to learn about the components of previously generated vertices.

We let the decoder attend to all the generated coordinates directly in the third alternative version of our model. We replace the MADE decoder with a 6-layer Transformer which is conditioned on $\{h_n\}_n$ (this time produced by a 14-layer torso) and operates on a flattened sequence of vertex components (similarly to the base model). The conditioning is done by adding h_n to the embeddings of z_n, y_n and x_n. While slower than the MADE version, the resulting network is significantly closer in performance to the base model.

Finally, we make the model even more powerful using a 2-layer Transformer instead of simple concatenation to embed each triplet of vertex coordinates. Specifically, we sum-pool the outputs of that Transformer within every vertex. In this variant, we reduce the depth of the torso to 10 layers. This results in test likelihood similar to the that of the base model.

F. Masking Invalid Predictions

As mentioned in Section 2.2 we mask invalid predictions when evaluating our models. We identify a number of hard constraints that exist in the data, and mask the model's predictions that violate these constraints. The masked probability mass is uniformly distributed across the remaining valid values. We use the following masks:

Vertex model.

- The stopping token can only occur after an x-coordinate:

$$v_k = s \implies v_k \bmod 3 = 1 \quad (21)$$

- z-coordinates are non-decreasing:

$$z_k \geq z_{k-1} \quad (22)$$

- y-coordinates are non-decreasing if their associated z-coordinates are equal:

$$y_k \geq y_{k-1} \text{ if } z_k = z_{k-1} \quad (23)$$

- x-coordinates are increasing if their associated z and y-coordinates are equal:

$$x_k > x_{k-1} \text{ if } y_k = y_{k-1} \text{ and } z_k = z_{k-1} \quad (24)$$

Face model.

- New face tokens n can not be repeated:

$$f_k \neq n \quad \text{if} \quad f_{k-1} = n \tag{25}$$

- The first vertex index of a new face is not less than the first index in the previous face:

$$f_1^{(k)} \geq f_1^{(k-1)}, \quad k = 1, \ldots, N_f \tag{26}$$

- Vertex indices within a face are greater than the first index in that face:

$$f_j^{(k)} > f_1^{(k)} \tag{27}$$

- Vertex indices within a face are unique:

$$f_i^{(k)} \neq f_j^{(k)}, \quad \forall i, j \tag{28}$$

- The first index of a new face is not greater than the lowest unreferenced vertex index:

$$f_1^{(k)} \leq \min \left[\{v : v \leq N_V\} \setminus \{f_1^{(j)}, \ldots, f_{N_j}^{(j)}\}_{j=1}^{k-1} \right] \tag{29}$$

G. Draco Compression Settings

We compare our model in Table 1 to Draco (Google), a performant 3D mesh compression library created by Google. We use the highest compression setting, quantize the positions to 8 bits, and do not quantize in order to compare with the 8-bit mesh representations that our model operates on. Note that the quantization performed by Draco is not identical to our uniform quantization, so the reported scores are not directly comparable. Instead they serve as a ballpark estimate of the degree of compression obtained by existing methods.

H. Unconditional Samples

Figure 14 shows a random batch of unconditional samples generated using PolyGen with nucleus sampling and top-$p = 0.9$. We observe that the model learns to mostly output objects consistent with a shape class. In addition, the samples contain a large proportion of certain object classes, including tables, chairs and sofas. This reflects the significant class-imbalance of the ShapeNet dataset, with many classes being underrepresented. Finally, certain failure modes are present in the collection. These include meshes with disconnected components, meshes that have produced the stopping token too early, producing incomplete objects, and meshes that don't have a distinct form that is recognizable as one of the shape classes.

layer name	output size	layer parameters
conv1	128×128×64	7×7, 64, stride 2
conv2_x	64×64×64	3×3 max pool, stride 2
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times1$
conv3_x	32×32×128	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$
conv4_x	16×16×256	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$
	256×256	spatial flatten (or)
	1×256	average pool

(a) Image encoder

layer name	output size	layer parameters
embed	28×28×28×8	embed, 8
conv1	14×14×14×64	7×7×7, 64, stride 2
conv2_x	14×14×14×64	$\begin{bmatrix} 3\times3\times3, 64 \\ 3\times3\times3, 64 \end{bmatrix}\times1$
conv3_x	7×7×7×256	$\begin{bmatrix} 3\times3\times3, 256 \\ 3\times3\times3, 256 \end{bmatrix}\times2$
	343×256	spatial flatten (or)
	1×256	average pool

(b) Voxel encoder

Table 4. Architectures for image and voxel encoders. Pre-activation residual blocks are shown in brackets, with the numbers of blocks stacked. Downsampling is performed by conv3_1, conv4_ for image encoders, and by conv3_ for voxel encoders, with a stride of 2. For AtlasNet models, we perform an additional linear projection up to 1024 dimensions before average pooling to obtain a vector shape representation.

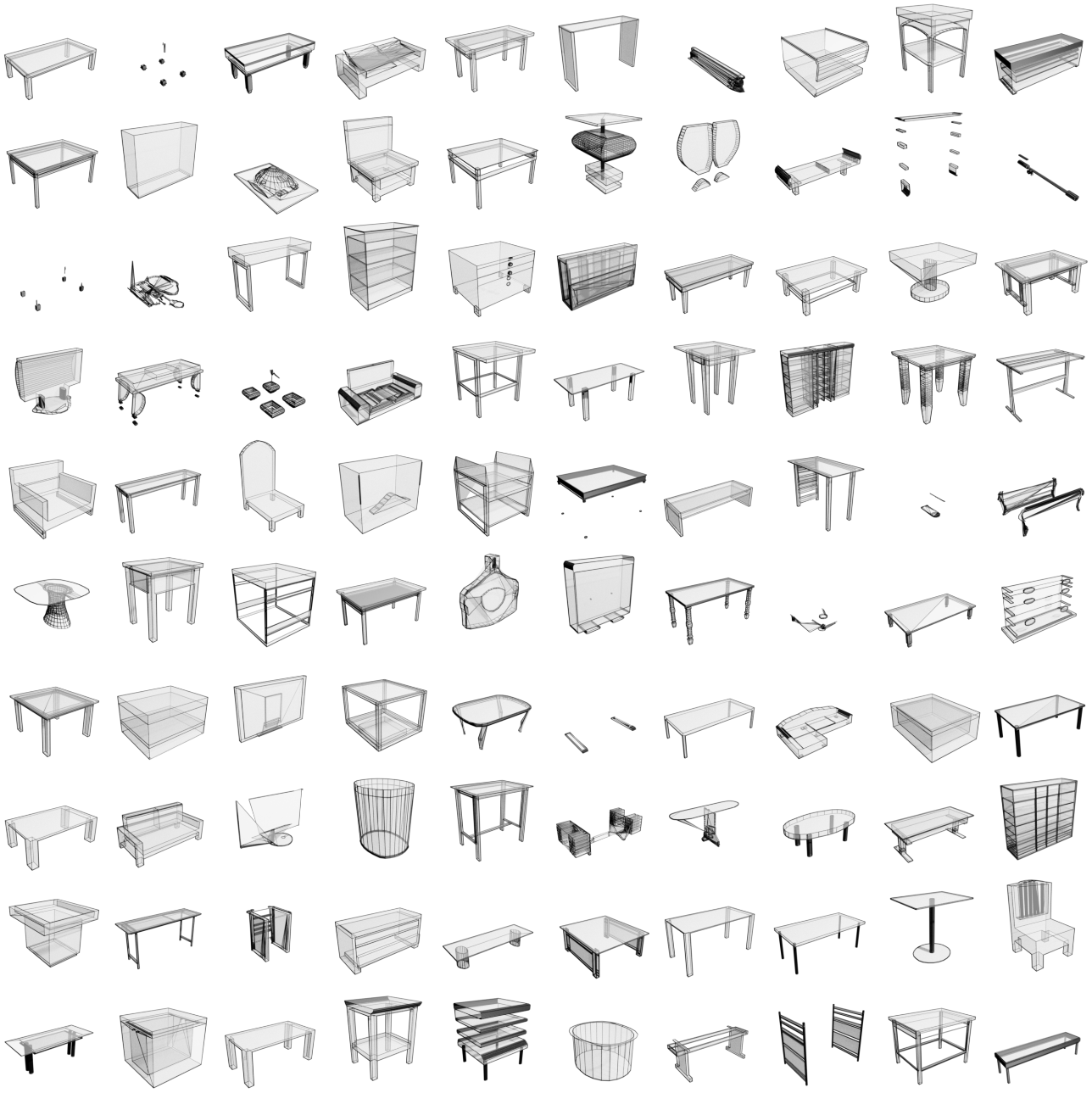

Figure 14. Random unconditional samples using nucleus sampling with top-$p = 0.9$.