

Leveraging 2D Data to Learn Textured 3D Mesh Generation

Paul Henderson
IST Austria
paul@pmh47.net

Vagia Tsiminaki
IBM Research Zürich
tsi@zurich.ibm.com

Christoph H. Lampert
IST Austria
chl@ist.ac.at

Abstract

Numerous methods have been proposed for probabilistic generative modelling of 3D objects. However, none of these is able to produce textured objects, which renders them of limited use for practical tasks. In this work, we present the first generative model of textured 3D meshes. Training such a model would traditionally require a large dataset of textured meshes, but unfortunately, existing datasets of meshes lack detailed textures. We instead propose a new training methodology that allows learning from collections of 2D images without any 3D information. To do so, we train our model to explain a distribution of images by modelling each image as a 3D foreground object placed in front of a 2D background. Thus, it learns to generate meshes that when rendered, produce images similar to those in its training set.

A well-known problem when generating meshes with deep networks is the emergence of self-intersections, which are problematic for many use-cases. As a second contribution we therefore introduce a new generation process for 3D meshes that guarantees no self-intersections arise, based on the physical intuition that faces should push one another out of the way as they move.

We conduct extensive experiments on our approach, reporting quantitative and qualitative results on both synthetic data and natural images. These show our method successfully learns to generate plausible and diverse textured 3D samples for five challenging object classes.

1. Introduction

Learning the structure of a 3D object class is a fundamental task in computer vision. It is typically cast as learning a probabilistic generative model, from which instances of the class may be sampled. The last five years have seen dramatic progress on this task [53, 52, 36, 32, 27, 41, 8, 45, 54, 9, 2, 15, 28], enabled by new, large-scale training sets [6, 53]. However, existing methods generate only shapes, without any associated textures to capture the surface appearance. This is a major shortcoming since the surface appearance of an object strongly influences how we

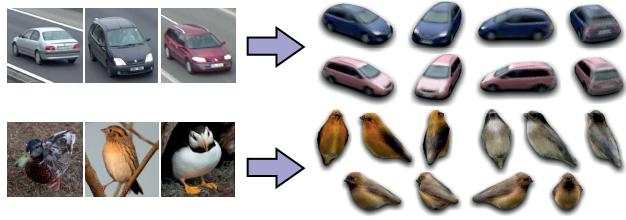


Figure 1. We propose a method to learn a generative model of textured 3D shapes (right), from collections of images (left)

perceive and interact with it—consider for example the difference between a red and a green tomato, a police car and a taxi, a book and a brick, or a zebra and a horse. As such, textures are vital for many practical uses of generated shapes, such as visual effects and games.

We hypothesise that the lack of research on methods that learn to generate textured shapes is in part due to a lack of textured 3D data for training them. Of the two large-scale datasets of 3D shapes, ShapeNet [6] lacks detailed textures for most instances, while ModelNet [53] lacks color information entirely.

We propose an alternative paradigm: rather than learning such a model from 3D data, we learn it from a large collection of 2D images (Figure 1). This lets us leverage existing weakly-annotated image datasets, at the price of solving a very challenging learning problem. It is challenging for three reasons: (i) infinitely many 3D shapes may project to give the same 2D image; (ii) we cannot rely on having multiple views of each object instance; (iii) objects appear in front of cluttered backgrounds, and segmentation masks may not be available. We must therefore learn to isolate foreground objects from background clutter, at the same time as learning their space of valid shapes and textures.

Our first contribution is a **new generative model over textured 3D shapes** (Section 3). Our second and most significant contribution is a **method to train this model to match a distribution of images** (Section 5), overcoming the difficulties mentioned. Specifically, our model learns to *reconstruct* its training images in terms of the physical process by which they were formed. This is achieved by augmenting the generative model with an image-formation model—we place the generated 3D object in front of a cam-

era, and render it over some background to give an image. An encoder network then predicts the latent parameters that give rise to any given image. Thus, the model must explain the distribution of training images in terms of a distribution of 3D foreground objects over 2D backgrounds (Figure 2). By modelling the variability among object instances using a limited-capacity latent space, we ensure that our method generates complete, coherent objects, rather than unrealistic shapes that could explain each training image in isolation. Informally, this works because it takes more bits to encode a distribution over many partial, viewpoint-dependent object appearances than over the variability of the true global appearance model.

We choose meshes as our representation of textured shapes, similar to some recent works on single-image 3D reconstruction [22, 30, 24, 49]. Meshes are the dominant shape representation in computer graphics, and have several benefits over alternatives such as voxels and point-clouds: (i) their computational cost scales (at worst) with surface area not volume; (ii) they can represent arbitrarily-oriented surfaces; (iii) they can directly represent solid surfaces with a well-defined normal and interior/exterior.

Training a model to output meshes that correctly reconstruct its training images suffers one potential failure mode—correct-looking images may be obtained by rendering even highly-irregular, self-intersecting meshes, due to the ambiguity of the projection operation. This is problematic as many downstream use-cases, such as physical simulations, geometry-processing algorithms, and 3D printing, require meshes that are *non-intersecting*—that is, no triangular faces intersect with any others. For smooth, regular object classes such as cars, non-intersection can be encouraged by careful regularization of the local surface geometry [24, 22]. However, for angular, non-convex object classes with elongated parts, such as chairs and airplanes, a sufficiently strong regularizer results in overly-smoothed surfaces lacking in detail.

As a further contribution, we therefore propose a **novel mesh parametrization, that necessarily yields non-intersecting surfaces even without regularization** (Section 4). It nonetheless has great representational flexibility, and can faithfully capture complex shapes such as chairs and airplanes. Our parametrization has a simple physical intuition—as faces move, they push others out of the way rather than intersecting them; we show how to formalise this idea and efficiently incorporate it in gradient-based training.

We conduct extensive experiments illustrating the performance of our method (Section 6). We first validate its performance on synthetic data from four diverse object classes, then show that it also performs well on two challenging classes of natural images. In all cases, we show both quantitatively and qualitatively that our method successfully learns to generate samples that are diverse and

realistic, even when ground-truth segmentation masks are not available. Moreover, we show that our novel mesh parametrization eliminates problematic self-intersections, yet allows representing angular and concave classes such as chairs, airplanes, and sofas.

2. Related work

The vast majority of methods for automatically generating 3D shapes are based on variational autoencoders (VAEs) [25] or generative adversarial networks (GANs) [11]. However, almost all such models (*e.g.* [53, 54, 45, 59, 3, 9, 2]) must learn from large datasets of 3D shapes [6, 53]; this stands in contrast to our own, which instead learns from collections of images. We now discuss existing works that do learn (untextured) 3D generative models from 2D datasets, and also methods for single-image reconstruction that are trained with only 2D data.

Generative 3D models learnt from 2D data. The first work to learn a generative model of 3D shapes from 2D data was [8]. The authors train a GAN that produces voxels, with a discriminator that ensures these project to silhouettes matching a distribution of ground-truth segmentation masks. MP-GAN [28] extends this to use multiple discriminators, corresponding to different viewpoints, simultaneously training a classifier to predict the viewpoint. Reliance on silhouettes is a severe limitation when ground-truth masks are not available; our method avoids it by reconstructing the image pixels themselves. The closest work in spirit to ours is [15], which learns single-image reconstruction and mesh generation from untextured renderings under known lighting. Unlike us, they do not learn to generate or reconstruct textures, so their method still cannot work on natural images. Moreover, their method has no mechanism to ensure that it produces meshes without self-intersections. Finally, HoloGAN [33] is a GAN over images, that incorporates a 3D latent feature space. This allows manipulation of 3D pose parameters somewhat-independently of object identity and background. However, it cannot output an explicit 3D shape representation such as a voxel grid or mesh.

Single-image reconstruction learnt from silhouettes. Several recent works learn single-image 3D reconstruction from 2D silhouettes [36, 56, 51, 47, 24, 46, 22, 19, 57, 14, 15, 30, 23, 16]. These are trained discriminatively to map images to 3D representations (voxels, meshes, or point-clouds); they use losses that ensure the reprojected, reconstructed silhouette matches one or more masks provided as supervision. Four of these works also consider colors: [30] has a post-processing stage that predicts the texture for a reconstructed mesh; [47] shows an example of reconstructing colored voxels, assuming training with multiple views per instance; [16] does the same even when only a single view per instance is available; [22] learns single-image tex-

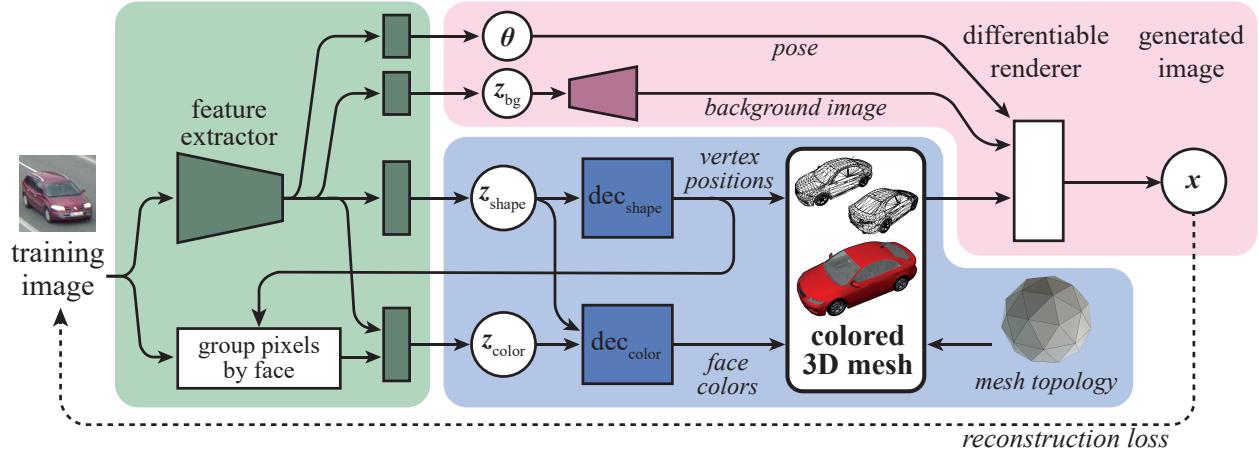


Figure 2. We propose a probabilistic generative model of textured 3D meshes (blue; see Section 3). We show how to train it using only 2D data (Section 5), by adding additional components (pink) that model the process of forming an image from a 3D foreground mesh rendered over 2D background clutter. We train the model to maximise the likelihood of a dataset of images, by adding an encoder model (green) that predicts the posterior distribution on latent variables for a given image. White circles represent random variables; colored boxes are densely-connected networks; trapezoids are convolutional networks

tured mesh reconstruction, using mask and keypoint annotations. Unlike ours, none of these methods allow sampling new meshes *a priori*—they do not learn an explicit prior.

Texture generation and reconstruction. Other recent works learn to generate or reconstruct textures given full supervision, which limits them to classes for which extensive textured 3D data is available. [35] defines textures implicitly, as a function of 3D position; they show that this representation allows sampling textures given a 3D shape, or reconstructing texture from a single image and shape. [59] generates images of textured 3D shapes, by first sampling textureless shapes and projecting them to a silhouette and depth-map; the texture is then generated in image space. [43] performs textured single-image reconstruction, using colored voxels as the output representation. [29, 38] learn priors on textures to improve multi-image reconstruction.

Non-intersecting mesh parametrization. One of our contributions is a method to parametrize meshes such that the resulting surface is highly flexible, yet guaranteed not to intersect itself. Methods for single-image 3D reconstruction use local smoothness regularizers such as Laplacian regularization [42], TV-L1 regularization [58], and regularizing the angles between adjacent faces [24, 30]. However, these only prevent local surface irregularity, and do not penalise two smooth surfaces passing through each other. The related problem of detecting, characterising and removing mesh self-intersections has received some attention in the graphics literature [55, 5, 20]. Unfortunately, none of these methods allow us to construct meshes that do not intersect *a priori*, which is necessary when we predict their shape directly from a neural network.

3. Modelling textured meshes

We begin by defining our probabilistic generative model for textured 3D meshes (Figure 2, blue background). Each mesh consists of N_V vertices, and N_F triangular faces to which we assign colour values \mathbf{c} . We assume fixed topology for all meshes, *i.e.* N_V , N_F , and the mapping between faces and vertices, do not vary between instances. To generate a mesh, the model must therefore sample the positions \mathbf{v} of all vertices, and the colors \mathbf{c} of all faces.

We draw low-dimensional latent code variables from standard Gaussian distributions, then pass these through *decoder* networks producing the required attributes:

$$\mathbf{z}_{\text{shape}} \sim \text{Normal}(\mathbf{0}, \mathbf{1}) \quad (1)$$

$$\mathbf{z}_{\text{color}} \sim \text{Normal}(\mathbf{0}, \mathbf{1}) \quad (2)$$

$$\mathbf{v} = \text{dec}_{\text{shape}}(\mathbf{z}_{\text{shape}}) \quad (3)$$

$$\mathbf{c} = \text{dec}_{\text{color}}(\mathbf{z}_{\text{color}}, \mathbf{z}_{\text{shape}}) \quad (4)$$

Here, $\mathbf{z}_{\text{shape}}$ captures the object’s 3D shape; the shape decoder $\text{dec}_{\text{shape}}$ can be as simple as a densely-connected ELU network with $3N_V$ outputs \mathbf{v} , corresponding to the 3D position of each vertex (we discuss a more sophisticated option later). Similarly, $\mathbf{z}_{\text{color}}$ captures texture; $\text{dec}_{\text{color}}$ is a densely-connected ELU network with $3N_F$ outputs \mathbf{c} , corresponding to the color of each face represented as red/green/blue values.

Note that separating $\mathbf{z}_{\text{shape}}$ from $\mathbf{z}_{\text{color}}$ lets us reconstruct the shape of an instance before its color is known, which will be important for our training process (Section 5). However, by passing the shape code into $\text{dec}_{\text{color}}$, we still allow dependencies between color and shape, *e.g.* to capture the fact that a bird with the shape of an eagle should never have the colors of a robin. Detailed network architectures

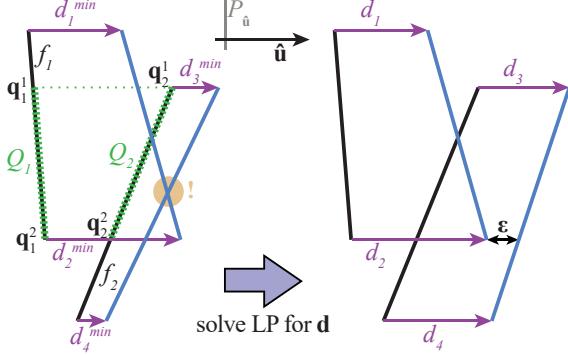


Figure 3. A 2D example of pushing faces (left is before pushing, right is after); $\hat{\mathbf{u}}$ lies in the plane of the page, with $P_{\hat{\mathbf{u}}}$ perpendicular. Black lines are the initial faces f_1, f_2 ; blue lines are after shifting by \mathbf{d}^{\min} (left) and \mathbf{d} (right). f_1, f_2 do not initially intersect, but overlap in $P_{\hat{\mathbf{u}}}$; the proposed distances \mathbf{d}^{\min} of motion along $\hat{\mathbf{u}}$ would create an intersection in the orange circle. \mathbf{d}^{\min} is mapped to \mathbf{d} such that the intersection is prevented, by solving an LP; note that $d_v > d_v^{\min}$ for $v \in \{3, 4\}$, but $d_v = d_v^{\min}$ for $v \in \{1, 2\}$.

are given in Appendix B.

Calculating vertex locations with a neural network typically results in highly irregular meshes with many self-intersections (*e.g.* Figure 5). This is undesirable for many use-cases, and in general, difficult to avoid with regularization alone. The next section describes a more sophisticated structure for $\text{dec}_{\text{shape}}$, that produces vertex locations which are guaranteed not to create intersecting surfaces.

4. Non-intersecting mesh parametrization

Our goal is a decoder network that produces vertex locations such that no surface intersections can occur, but highly concave, angular surfaces (*e.g.* chairs) can still be represented.

Physical motivation. When playing with a deflated balloon, we can deform it quite drastically without introducing any intersections, simply because when attempting to push one surface through another, the second will instead be pushed out of the way. It is not computationally feasible to simulate the physical dynamics of such a system during training. Nonetheless, we can make use of this insight, by combining a careful choice of parametrization with a simple, efficient model of surface collisions.

Parametrization. Instead of producing the final set of vertex positions in a single shot, we perform a sequence of N_s simpler deformation steps, starting from an initial, non-learnt mesh with spherical topology (and thus no intersections). In each step, we will move all vertices in the same direction $\hat{\mathbf{u}}$, but by differing (non-negative) distances \mathbf{d} . A densely-connected network outputs $\hat{\mathbf{u}}$, and *lower bounds* \mathbf{d}^{\min} on the distances, but these are modified to give \mathbf{d} such that each face pushes others rather than intersecting them.

Pushing faces. We map the initial distances \mathbf{d}^{\min} to pushed ones \mathbf{d} that minimise $\sum_v d_v$, where v indexes vertices. This minimisation is subject to two sets of constraints: (i) $d_v \geq d_v^{\min} \forall v$, *i.e.* each vertex moves at least as far as specified by the decoder network, and (ii) no intersections should arise.

To impose (ii), we first find all pairs of faces that *could* intersect, depending on the distances their vertices move. As all vertices move in the same direction, this problem can be considered in 2D: we project all vertices and faces into a plane $P_{\hat{\mathbf{u}}}$ perpendicular to the direction of motion $\hat{\mathbf{u}}$, and find all pairs of faces for which the corresponding triangles in $P_{\hat{\mathbf{u}}}$ intersect with non-zero area. For each such pair of faces (f_1, f_2), their intersection is a polygonal region of $P_{\hat{\mathbf{u}}}$; we can re-project this region back onto each face in 3D space, giving planar polygons (Q_1, Q_2) respectively (Figure 3).

As there are initially no intersections, we can order Q_1 and Q_2 according to their projection onto $\hat{\mathbf{u}}$: if \mathbf{q}_i^j is the position of the j^{th} corner of Q_i , then either $\mathbf{q}_1^j \cdot \hat{\mathbf{u}} < \mathbf{q}_2^j \cdot \hat{\mathbf{u}} \forall j$, or $\mathbf{q}_1^j \cdot \hat{\mathbf{u}} > \mathbf{q}_2^j \cdot \hat{\mathbf{u}} \forall j$. We assume without loss of generality that the former holds; to avoid intersections, this must remain true even after moving each vertex by $d_v \hat{\mathbf{u}}$.

Let β_i^j be the barycentric coordinates of \mathbf{q}_i^j w.r.t. the triangular face f_i in which it lies. The distance by which \mathbf{q}_i^j moves is then given by $\beta_i^j \cdot \mathbf{d}|_{f_i}$, where $\mathbf{d}|_{f_i}$ contains the elements of \mathbf{d} corresponding to the vertices of face f_i . The constraint that f_1 and f_2 do not intersect, *i.e.* that none of the points \mathbf{q}_i^j change ordering, then becomes

$$\mathbf{q}_1^j \cdot \hat{\mathbf{u}} + \beta_1^j \cdot \mathbf{d}|_{f_1} + \varepsilon \leq \mathbf{q}_2^j \cdot \hat{\mathbf{u}} + \beta_2^j \cdot \mathbf{d}|_{f_2} \quad \forall j \quad (5)$$

where ε is a small buffer distance.

All the constraints we have defined are linear in \mathbf{d} . Thus, minimising our objective $\sum_v d_v$ under them defines a linear programming problem (LP), which can be solved efficiently using the simplex algorithm [34]. In practice, we use the efficient off-the-shelf solver of [31].

Propagating derivatives. We have now defined \mathbf{d} as the solution to an optimisation problem that has \mathbf{d}^{\min} and the vertex locations as inputs. In order to incorporate this in our decoder network, we need to propagate gradients back through the optimisation process from \mathbf{d} to these inputs. Note that the solution to an LP always lies at a corner of the polytope defined by its constraints. At this corner, equality holds for some subset of *active* constraints. These constraints define a system of linear equations, whose solution equals \mathbf{d} . Thus, back-propagating gradients from \mathbf{d} is exactly equivalent to back-propagating them through the process of solving this linear system, *e.g.* by Cholesky decomposition, allowing direct implementation in TensorFlow [1].

5. Training from images

Our goal is to train the generative model of Section 3 using only images, without any 3D data. We assume access to a training set of images, each containing exactly one instance of the target object class, and consider two training settings:

- (MASK) We have access to (i) the approximate camera calibration; (ii) a segmentation mask for each target object instance; and (iii) the background image, *i.e.* a view of the same environment but without the foreground object present. For our experiments, we estimate these automatically from weakly-annotated data, *e.g.* by running a segmentation algorithm on unannotated data and inpainting the background, and estimating the camera calibration from keypoints. We found the model is robust to even quite large errors in the camera calibration. This setting is similar to some weakly-supervised methods for single-image 3D reconstruction [22, 23] and untextured 3D generation [8, 15].
- (NO-MASK) We have only the (approximate) camera calibration available. This second setting is much more challenging, and goes beyond all prior works on weakly-supervised reconstruction and generation.

To allow training in these settings, we augment the generative model with additional components to model the entire image formation process (Figure 2, pink background). Specifically, after sampling the mesh itself, we position it in 3D space in front of a perspective camera, and render it over a background image. The final, observed image \mathbf{x} is an isotropic Gaussian random variable with mean equal to the rendered pixels, and fixed variance. We then introduce an encoder (or inference) network, that predicts the latent variables corresponding to a given image. This lets us train our model to match a distribution of *images* (rather than meshes), by learning to reconstruct each in terms of a foreground mesh in front of background clutter. We now describe each aspect of this training methodology in detail.

Background image. In setting (MASK), the background is provided as input to the model along with the training image. In setting (NO-MASK), we explicitly model the background, *i.e.* our generative process samples both the 3D foreground object, and the 2D pixels that are ‘behind’ it. The background is generated by sampling a low-dimensional latent code vector \mathbf{z}_{bg} , and passing it through a convolutional decoder network. Whereas the decoder in a VAE or GAN is typically designed to be as powerful as possible, we need to avoid the background model being *too powerful*, and thus able to model the foreground object as well. We therefore set the dimensionality of \mathbf{z}_{bg} to be just

16, use only three transpose-convolutional layers, and upsample the resulting image $4\times$ or $6\times$ to the desired resolution. This ensures the model cannot capture high-frequency details such as the edge of the foreground object.

Rendering. To render the generated mesh over the background image, we first place it in 3D space relative to a camera at the origin, according to a pose θ . This captures the fact that the object may not be centered nor at known, constant depth in the original image, and we do not wish the shape model to have to account for this variability. We then project and rasterise the mesh, using the publicly-available differentiable mesh renderer *DIRT* [13, 15]. We use direct illumination and Lambertian reflectance [26], and do not attempt to disentangle albedo from shading.

Variational training. Let \mathbf{z} denote all the latent variables ($\mathbf{z}_{\text{shape}}, \mathbf{z}_{\text{color}}, \mathbf{z}_{\text{bg}}, \theta$). Our full model defines a joint distribution $P(\mathbf{x}, \mathbf{z}) = P(\mathbf{z})P(\mathbf{x}|\mathbf{z})$ over these and the resulting image \mathbf{x} . We would ideally train it to maximise the likelihood of a training dataset of images; however, this is intractable due to the latent variables. We therefore adopt a variational approach [21], adding an encoder network that predicts parameters of a variational posterior distribution $Q(\mathbf{z}|\mathbf{x})$. The following is then a lower bound (the *ELBO*) on the data log-likelihood [37, 25]:

$$\mathcal{L} = \mathbb{E}_{Q(\mathbf{z}|\mathbf{x})} \log P(\mathbf{x}|\mathbf{z}) - D_{\text{KL}}[Q(\mathbf{z}|\mathbf{x}) || P(\mathbf{z})] \leq \log P(\mathbf{x}). \quad (6)$$

In practice, rather than maximising \mathcal{L} directly, we make two modifications. First, following [18], we multiply the KL-divergence term by a constant factor. Second, we replace the Gaussian log-likelihood $\log P(\mathbf{x}|\mathbf{z})$ by the multi-scale structural similarity (MS-SSIM) metric [50], which (i) allows gradients to propagate across longer distances in the image, and (ii) tends to preserve fine details better. The generative model and encoder network are trained jointly to maximise this objective—thus they learn to reconstruct input images, while ensuring the posterior distribution on the latents is close to the prior.

Encoder. The encoder network (Figure 2, green background) takes an image as input, and predicts the mean and variance of Gaussian posterior distributions on $\mathbf{z}_{\text{shape}}, \mathbf{z}_{\text{color}}, \mathbf{z}_{\text{bg}}$, and θ . We use a small CNN (similar to [51, 15]) to extract features from the image; the mean and log-variance of $\mathbf{z}_{\text{shape}}, \mathbf{z}_{\text{bg}}$, and θ are then computed by densely-connected layers. Detailed network architectures are given in Appendix B.

We experimented with the same approach for $\mathbf{z}_{\text{color}}$. However, this gives rise to a challenging learning task: the network must map an image to the colors of all faces, without explicitly knowing which faces would be visible in the reconstruction, nor where in the image they project to. We achieved better results using a novel architecture that explicitly incorporates this information. We group pixels of



Figure 4. Textured meshes sampled from our model, trained on renderings from ShapeNet. Each row of five images shows the same sampled mesh from different viewpoints. Cars and sofas are trained in setting (NO-MASK) with (DENSE) parametrization; chairs and airplanes are trained in setting (MASK) with (PUSHING) parametrization, and thus are free of self-intersections in spite of the finely detailed geometry. Note the samples are diverse and realistic, both in terms of the mesh geometry and textures.

the input image (Figure 2, ‘group pixels by face’) according to which (if any) face of the reconstructed mesh defined by $\mathbf{z}_{\text{shape}}$ they would lie in (this is possible as the shape does not depend on $\mathbf{z}_{\text{color}}$, hence can be reconstructed first; see Section 3). For each group (including one for background pixels), we calculate the mean RGB values of the pixels that were assigned to it. This defines a matrix of size $(N_F + 1) \times 3$, which we flatten and encode with a small densely-connected network. The resulting code then captures the input pixel colors in a way that accounts for

where they lie with respect to the surface of the reconstructed mesh. Thus, the encoder network does not have to learn this invariance itself from data. Finally, the mean and variance for $\mathbf{z}_{\text{color}}$ are estimated by another dense layer taking this code and the image features as input.

Regularization. In setting (MASK), we additionally maximise the intersection-over-union (IOU) between the ground-truth mask and the reconstructed silhouette. This ensures the foreground mesh does not overlap into background regions of the image. For easier optimisation, we take the mean of the IOU at multiple scales, using smoothed and downsampled masks/silhouettes. We also found it useful to add local mesh smoothness regularizers, similar to [22, 24, 30].

6. Experiments

We conduct experiments on five diverse object classes: birds, cars, airplanes, chairs, and sofas, which have also been the focus of related works on weakly-supervised reconstruction and generation [22, 28, 15, 46]. In Section 6.1, we validate our method in a controlled setting on renderings of ShapeNet meshes [6], analysing its performance under different settings and parametrizations. Then, in Section 6.2, we show that it successfully learns models from two challenging datasets of natural images. Finally, in Section 6.3, we show that the trained model can also be used for single-image 3D reconstruction on natural images.

All hyperparameters (*e.g.* regularizer strengths and mesh resolution) were selected manually for perceived quality of generation; we did not directly optimise them w.r.t. our quantitative metrics. Our code, hyperparameters and pre-processed datasets will be made available online very soon.

Metrics. We are not aware of any existing evaluation metric for generative models of textured 3D shapes. We therefore propose a new evaluation protocol using established 2D metrics: inception score (IS) [39], Fréchet inception distance (FID) [17] and kernel inception distance (KID) [4]. All these metrics pass a large set of generated images through the CNN of [44], recording output logits and feature activations. IS measures how similar the logits are to a uniform distribution, in terms of KL divergence (larger is better). FID and KID pass a test set of ground-truth images through the same CNN, then measure how similar their feature activations are to those of the generated images (smaller is better). To apply these metrics in our setting, we render 25600 meshes sampled from our model, each over a ground-truth background, and report the IS/FID/KID values for these images¹.

¹use of ground-truth backgrounds avoids confounding the quality of mesh generation with that of background generation in setting (NO-MASK)

	(MASK)			(NO-MASK)		
	IS	FID	KID	IS	FID	KID
airplane	4.0	73.5	0.063	3.2	56.5	0.044
car	4.1	154.0	0.123	3.5	165.4	0.136
chair	5.8	111.1	0.088	5.2	82.6	0.061
sofa	4.3	58.3	0.037	4.3	63.8	0.041

Table 1. Quantitative measures of generation for four ShapeNet classes, in settings (MASK) and (NO-MASK) (training with and without ground-truth masks respectively) For IS, larger is better; for FID/KID, smaller is better.

6.1. Validation on ShapeNet

For our experiments on synthetic data, we use four ShapeNet [6] classes: car, chair, airplane, and sofa. These have very different characteristics—cars have a smooth, largely-convex shape and areas of high-frequency texture; chairs and airplanes have much more angular shapes with multiple elongated parts, but are often of more uniform texture; sofas are typically concave with large, flat surfaces. We use 80% of the instances in each class for training, and 20% for evaluation.

Our model is trained using renderings rather than the meshes themselves. For car and sofa, we use the renderings of [7]; for chair and airplane, we obtained better results using those of [12], which have greater pose diversity. Although these datasets contain several images per mesh, we shuffle the images randomly, so there is only a very small chance of the same object appearing twice in one minibatch. As a form of data augmentation, we alpha-composite the renderings over random solid color backgrounds.

Generation with and without mask annotations. We train separate models on each of the four classes, in each of the two supervision settings (MASK) and (NO-MASK). In Figure 4, we see that our method has learnt to generate plausible samples for all four classes. The samples are reasonably diverse in terms of both texture and shape—note the different colors of car, different styles of chair, etc. More examples are shown in Appendix A. Even in setting (NO-MASK), the model has learnt to sample meshes that represent a complete instance of the foreground object, without including any sections of background—in spite of training without segmentation annotations. This is particularly impressive for chairs, which have narrow legs that could easily be ignored. Quantitatively, Table 1 shows that (NO-MASK) does give slightly poorer results than (MASK) in terms of IS, which is expected as it is a significantly more challenging setting. For FID and KID, performance is similar in the two settings—car and sofa are better with (MASK), and airplane and chair with (NO-MASK).

Non-intersecting mesh parametrization We now evaluate our non-intersecting mesh parametrization (PUSHING), comparing it to a simple dense decoder (DENSE) that di-

	(DENSE)		(PUSHING)		
	int. faces	int. faces	IS	FID	KID
airplane	56.1 %	0 %	3.7	76.3	0.067
chair	58.8 %	0 %	5.4	145.8	0.127
sofa	77.0 %	0 %	4.0	82.6	0.058

Table 2. Mean fraction of intersecting faces per generated mesh (int. faces) in setting (MASK) using (DENSE) and (PUSHING) parametrizations. We also give the generation metrics for (PUSHING); see Table 1 for the same using (DENSE).

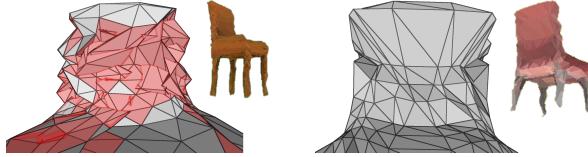


Figure 5. Examples of chair mesh structure and renderings using densely-connected (DENSE) (left) and non-intersecting (PUSHING) (right) mesh parametrizations. Faces that intersect others are highlighted in red. Note that the right-hand mesh is free of intersecting faces, in contrast to the left. Consistency of the renderings with an input image is not sufficient to enforce this, as both renders appear reasonable. In both cases, we used the same strengths for the local smoothness regularizers, so these do not influence the result.

rectly outputs vertex locations. Qualitative examples with (PUSHING) are given in Figure 4 for classes chair and airplane; we see that this parametrization is powerful enough to produce realistic shapes, in spite of it having considerably fewer degrees of freedom than (DENSE). Figure 5 shows details of two chairs for illustration, one sampled with (DENSE) and the other with (PUSHING). While both render to give reasonable images, the sample with (DENSE) has numerous intersecting faces visible, while that with (PUSHING) does not.

For quantitative comparison, we use the classes chair, airplane, and sofa, for which local surface regularizers struggle to prevent self-intersections. As well as the three generation metrics, we also measure the mean fraction of faces that intersect another face. We see (Table 2) that (DENSE) typically produces shapes with substantial numbers of self-intersections, while (PUSHING) avoids these. Thus, meshes from (PUSHING) may be used in downstream tasks which require a non-intersecting surface, in contrast to those from (DENSE). Conversely, all three generation metrics indicate slightly worse performance with (PUSHING). We believe this is because although (PUSHING) is highly expressive, the decoder network is less free to make arbitrary movements to individual vertices in response to gradients, as vertices are tied together through the shared directions \hat{u} . The space of shapes is therefore more difficult to traverse by gradient descent. Thus, there is a trade-off to be made depending on the desired application—when non-intersection is important, (PUSHING) should be used, but when it is not

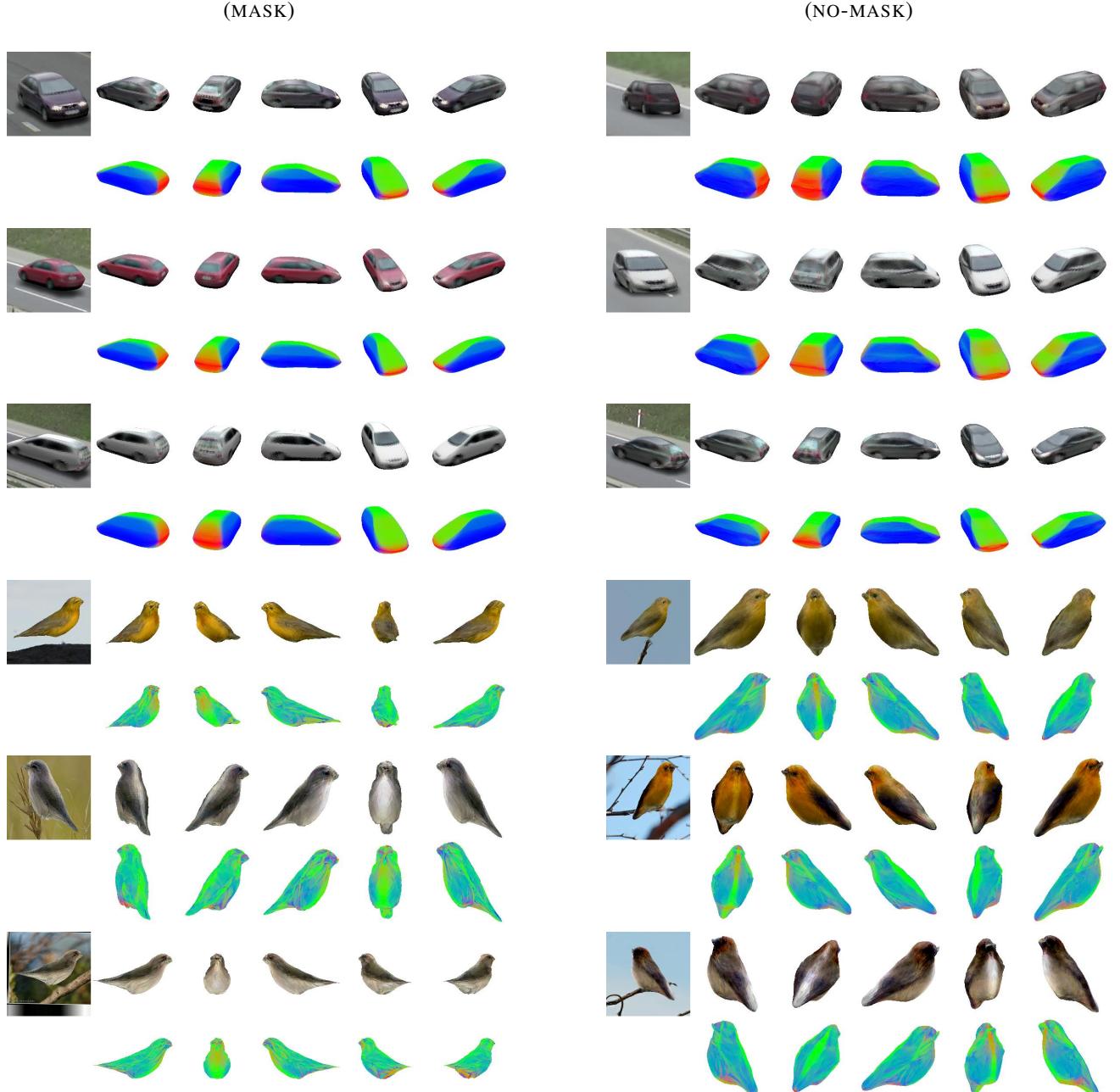


Figure 6. Cars and birds generated by our model. Each group of 11 images shows different views of one textured 3D mesh sampled from our model. The left-hand image shows the sampled mesh rendered with a background and pose drawn randomly from the test dataset. The lower five images in each group are normal maps, which reveal the 3D structure more clearly. Samples in the left column use setting (MASK); those in the right column use (NO-MASK).

a requirement, (DENSE) may be more appropriate.

6.2. Generation results on natural images

For our experiments on natural images, we use two classes: bird and car. The images of birds are taken from the CUB-200-2011 dataset [48]; we use the preprocessed version from [22], including their approximate camera calibrations, and discard the class labels. Each image contains

one bird, which we crop out using the provided segmentation masks, applying a random offset for data augmentation. For setting (MASK), we synthesise a background by inpainting the region corresponding to the bird, replacing each pixel inside the mask with the nearest non-foreground pixel. The images of cars are taken from the BrnoComp-Speed dataset [40], and we use their approximate camera calibration. We segment and crop out individual cars us-

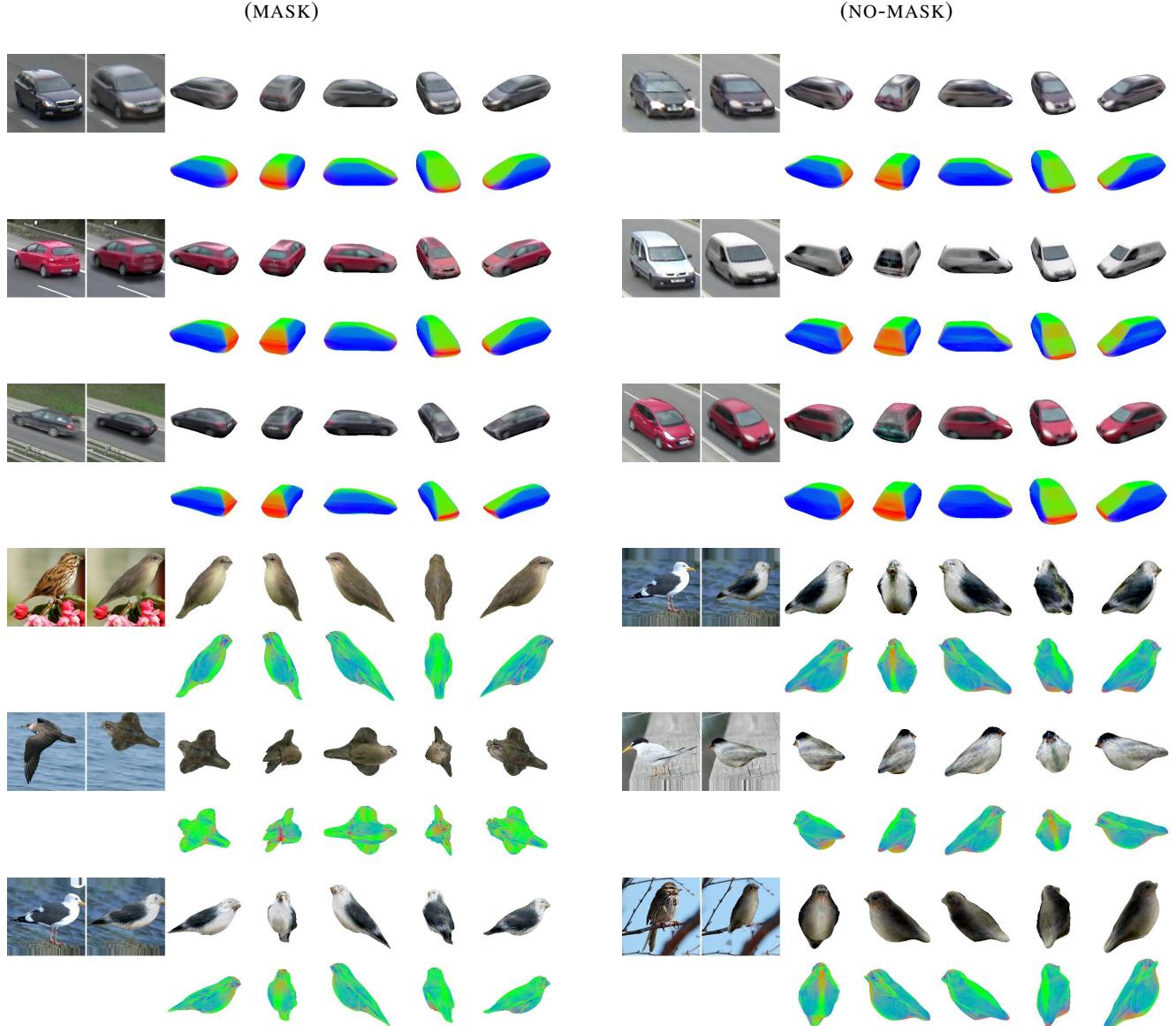


Figure 7. Textured 3D reconstructions of cars and birds. The left-hand image in each group is the input to our model; the next is our reconstruction, rendered over the ground-truth background; the remaining five columns are different views of the reconstructed mesh, with normal-maps below. Samples in the the left column use setting (MASK); those in the right column use (NO-MASK).

ing [10], and use the median frame as the background. Although the original data is video, we sample frames sparsely and shuffle them, treating them as independent images.

We see (Figure 6) that our approach yields realistic 3D meshes for both classes (more samples are shown in Appendix A). In setting (MASK), the sampled cars have diverse shapes and colors; in the more challenging setting (NO-MASK), the shapes are a little less diverse, and the colours a little more blurry, but all samples are still clearly identifiable as cars. For birds, the results are even stronger; there is no perceptible decrease in performance when we do not have mask annotations. This is perhaps because the highly varied backgrounds of the bird images are difficult for the

foreground model to (incorrectly) incorporate.

A similar pattern is apparent in the quantitative results (Table 3), where we see that (NO-MASK) birds in fact perform significantly better than any other combination. Meanwhile, for car, we see that (NO-MASK) performs slightly worse than (MASK), in accordance with the qualitative results. We also give results for a conventional (2D) deep convolutional VAE, with similar latent capacity, encoder architecture, and training time as our models, to give a reference for the ranges of the different metrics. Our 3D models give significantly better results than this baseline for birds, and somewhat worse for cars; this possibly reflects the high degree of background complexity in CUB.

	(MASK)			(NO-MASK)		
	IS	FID	KID	IS	FID	KID
<i>Our model</i>						
car	2.8	191.4	0.211	3.0	182.1	0.197
bird	4.5	104.3	0.090	3.8	75.4	0.060
<i>2D VAE</i>						
car	-	-	-	3.0	157.4	15.4
bird	-	-	-	3.3	213.8	21.0

Table 3. Quantitative measures of generation, training on natural images of cars and birds, in settings (MASK) and (NO-MASK). For IS, larger is better; for FID/KID, smaller is better. For reference, we also include results from a 2D deep convolutional VAE.

6.3. Single-image 3D reconstruction on natural images

While trained for generation, our model learns single-image 3D reconstruction ‘for free’—we can obtain a textured mesh reconstruction by running the encoder and decoder networks (green and blue parts of Figure 2) on an image. Specifically, the encoder network yields a variational posterior distribution on the latent variables; we take the mode of this distribution, and pass it through the decoders $\text{dec}_{\text{shape}}$ and $\text{dec}_{\text{color}}$ to produce a textured 3D mesh. Thus, we perform single-image 3D reconstruction, in spite of our model never having received any 3D supervision, nor even segmentation masks in (NO-MASK) setting. Note also that we did *not* tune our models for reconstruction quality.

Figure 7 shows examples for each class in both training settings, on held-out validation images; more examples are given in Appendix A. For the bird images, our setting (MASK) matches that of [22], allowing qualitative comparison with their approach².

We see that in almost all cases, the network faithfully reconstructs the visible part of the object, including details of texture and shape. Notably, occluded regions are also reconstructed plausibly, even though they do not influence the photometric reconstruction loss. This is because the model must learn to produce textured shapes that reproject well for all training images, while representing their variability in a low-dimensional latent space. Moreover, it must do so subject to the KL term in the loss (6) that explicitly limits the latent capacity. This discourages solutions where the predicted shapes explain each image in isolation, but occluded parts have appearances lying outside the space of variability that is observed in images where that part is visible.

²the background we use to illustrate the reconstruction for birds is generated from the original image, with all points inside the (ground-truth) bird mask replaced by the nearest non-masked pixel

7. Conclusion

We have presented a new generative model of textured 3D meshes, and shown how to train this from images alone, by augmenting it to capture the entire image formation process. We train the model to explain its training images, by reconstructing each in terms of a foreground mesh rendered over a background. We have shown that this approach allows us to generate diverse and realistic textured meshes of five object classes. Importantly, our method can use natural images, not just renderings, as training data. Moreover, it does not rely on multiple views of the same instance, nor on ground-truth segmentation masks. The resulting meshes are textured, and so may be used immediately in downstream applications such as visual effects and games.

We have also presented a new mesh parametrization, that avoids intersections *a priori*. This is useful whenever we need to generate a mesh from a neural decoder with the guarantee that it does not contain any self-intersections, necessary for example if it is to be used for physical simulation or 3D printing. However, this comes at the expense of producing samples that score slightly lower than a naïve parametrization in terms of IS/FID/KID metrics.

References

- [1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. 4
- [2] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3D point clouds. In *ICML*, 2018. 1, 2
- [3] Elena Balashova, Vivek Singh, Jiangping Wang, Brian Teixeira, Terrence Chen, and Thomas Funkhouser. Structure-aware shape synthesis. In *3DV*, 2018. 2
- [4] Mikolaj Bińkowski, Dougal J. Sutherland, Michael N. Arbel, and Arthur Gretton. Demystifying MMD GANs. In *ICLR*, 2018. 6
- [5] Marcel Campen and Leif Kobbelt. Exact and robust (self-)intersections for polygonal meshes. In *Eurographics*, 2010. 3
- [6] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. *arXiv preprint*, arXiv:1512:03012, 2015. 1, 2, 6, 7

- [7] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3D-R2N2: A unified approach for single and multi-view 3D object reconstruction. In *ECCV*, 2016. 7
- [8] Matheus Gadelha, Subhransu Maji, and Rui Wang. 3D shape induction from 2D views of multiple objects. In *3DV*, 2017. 1, 2, 5
- [9] Matheus Gadelha, Rui Wang, and Subhransu Maji. Multiresolution tree networks for 3D point cloud processing. In *ECCV*, 2018. 1, 2
- [10] Ross Girshick, Ilya Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron. <https://github.com/facebookresearch/detectron>, 2018. 9
- [11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *NIPS*, 2014. 2
- [12] Christian Häne, Shubham Tulsiani, and Jitendra Malik. Hierarchical surface prediction for 3D object reconstruction. In *3DV*, 2017. 7
- [13] Paul Henderson. DIRT: a fast differentiable renderer for TensorFlow. <https://github.com/pmh47/dirt>, 2018. 5
- [14] Paul Henderson and Vittorio Ferrari. Learning to generate and reconstruct 3D meshes with only 2D supervision. In *BMVC*, 2018. 2
- [15] Paul Henderson and Vittorio Ferrari. Learning single-image 3D reconstruction by generative modelling of shape, pose and shading. *IJCV*, 2019. 1, 2, 5, 6
- [16] Philipp Henzler, Niloy J. Mitra, and Tobias Ritschel. Escaping platos cave: 3D shape from adversarial rendering. In *ICCV*, 2019. 2
- [17] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *NIPS*, 2017. 6
- [18] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. β -VAE: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017. 5
- [19] Eldar Insafutdinov and Alexey Dosovitskiy. Unsupervised learning of shape and pose with differentiable point clouds. In *NIPS*, 2018. 2
- [20] Alec Jacobson, Ladislav Kavan, and Olga Sorkine-Hornung. Robust inside-outside segmentation using generalized winding numbers. *Trans. on Graphics*, 32(4):33:1–33:12, 2013. 3
- [21] M. Jordan, Z. Ghahramani, T. Jaakkola, and L. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37:183–233, 1999. 5
- [22] Angjoo Kanazawa, Shubham Tulsiani, Alexei A. Efros, and Jitendra Malik. Learning category-specific mesh reconstruction from image collections. In *ECCV*, 2018. 2, 5, 6, 8, 10
- [23] Hiroharu Kato and Tatsuya Harada. Learning view priors for single-view 3D reconstruction. In *CVPR*, 2019. 2, 5
- [24] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3D mesh renderer. In *CVPR*, 2018. 2, 3, 6
- [25] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *ICLR*, 2014. 2, 5
- [26] Johann Heinrich Lambert. *Photometria*. Eberhard Klett Verlag, 1760. 5
- [27] Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas Guibas. GRASS: Generative recursive autoencoders for shape structures. *Trans. on Graphics*, 36(4), 2017. 1
- [28] Xiao Li, Yue Dong, Pieter Peers, and Xin Tong. Synthesizing 3D shapes from silhouette image collections using multi-projection generative adversarial networks. In *CVPR*, 2019. 1, 2, 6
- [29] Yawei Li, Vagia Tsiminaki, Radu Timofte, Marc Pollefeys, and Luc van Gool. 3D appearance super-resolution with deep learning. In *CVPR*, 2019. 3
- [30] Shichen Liu, Weikai Chen, Tianye Li, and Hao Li. Soft rasterizer: Differentiable rendering for unsupervised single-view mesh reconstruction. In *ICCV*, 2019. 2, 3, 6
- [31] Gurobi Optimization LLC. Gurobi optimizer reference manual, 2019. 4
- [32] Charlie Nash and Christopher K. I. Williams. The shape variational autoencoder: A deep generative model of part-segmented 3D objects. *Computer Graphics Forum*, 36(5):1–12, 2017. 1
- [33] Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. HoloGAN: Unsupervised learning of 3D representations from natural images. In *ICCV*, 2019. 2
- [34] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer-Verlag, 2006. 4
- [35] Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. Texture fields: Learning texture representations in function space. In *ICCV*, 2019. 3
- [36] Danilo J. Rezende, S. M. Ali Eslami, Shakir Mohamed, Peter Battaglia, Max Jaderberg, and Nicolas Heess. Unsupervised learning of 3D structure from images. In *NIPS*, 2016. 1, 2
- [37] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, 2014. 5
- [38] Audrey Richard, Ian Cherabier, Martin R. Oswald, Vagia Tsiminaki, Marc Pollefeys, and Konrad Schindler. Learned multi-view texture super-resolution. In *3DV*, 2019. 3
- [39] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. Improved techniques for training GANs. In *NIPS*, 2016. 6
- [40] Jakub Sochor, Roman Juránek, Jakub Špaříhel, Lukas Maršík, Adam Široký, Adam Herout, and Pavel Zemčík. Comprehensive data set for automatic single camera visual speed measurement. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–11, 2018. 8
- [41] Amir A. Soltani, Haibin Huang, Jiajun Wu, Tejas D. Kulkarni, and Joshua B. Tenenbaum. Synthesizing 3d shapes via modeling multi-view depth maps and silhouettes with deep generative networks. In *CVPR*, 2017. 1
- [42] Olga Sorkine. Laplacian mesh processing. In *Eurographics*, 2005. 3

- [43] Yongbin Sun, Ziwei Liu, Yue Wang, and Sanjay E. Sarma. Im2Avatar: Colorful 3D reconstruction from a single image. *arXiv preprint*, arXiv:1804.06375, 2018. 3
- [44] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, and Jon Shlens. Rethinking the inception architecture for computer vision. In *CVPR*, 2016. 6
- [45] Qingyang Tan, Lin Gao, and Shihong Xia Yu-Kun Lai. Variational autoencoders for deforming 3d mesh models. In *CVPR*, 2018. 1, 2
- [46] Shubham Tulsiani, Alexei A. Efros, and Jitendra Malik. Multi-view consistency as supervisory signal for learning shape and pose prediction. In *CVPR*, 2018. 2, 6
- [47] Shubham Tulsiani, Tinghui Zhou, Alexei A. Efros, and Jitendra Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *CVPR*, 2017. 2
- [48] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011. 8
- [49] Nanyang Wang, Yinda Zhang, Zhuwen Li, Yanwei Fu, Wei Liu, and Yu-Gang Jiang. Pixel2Mesh: Generating 3D mesh models from single RGB images. In *ECCV*, 2018. 2
- [50] Zhou Wang, Eero P. Simoncelli, and Alan C. Bovik. Multiscale structural similarity for image quality assessment. In *The Thirty-Seventh Asilomar Conference on Signals, Systems & Computers*, volume 2, pages 1398–1402, 2003. 5
- [51] Olivia Wiles and Andrew Zisserman. SilNet: Single- and multi-view reconstruction by learning from silhouettes. In *BMVC*, 2017. 2, 5
- [52] Jiajun Wu, Chengkai Zhang, Tianfan Xue, William T Freeman, and Joshua B Tenenbaum. Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. In *NIPS*, 2016. 1
- [53] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Liguang Zhang, Xiaou Tang, and Jianxiong Xiao. 3D ShapeNets: A deep representation for volumetric shape modeling. In *CVPR*, 2015. 1, 2
- [54] Jianwen Xie, Zilong Zheng, Ruiqi Gao, Wenguan Wang, Song-Chun Zhu, and Ying Nian Wu. Learning descriptor networks for 3d shape synthesis and analysis. In *CVPR*, 2018. 1, 2
- [55] Soji Yamakawa and Kenji Shimada. Removing self intersections of a triangular mesh by edge swapping, edge hammering, and face lifting. In Brett W. Clark, editor, *Proc. 18th Intl Meshing Roundtable*, pages 13–29, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. 3
- [56] Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. Perspective transformer nets: Learning single-view 3D object reconstruction without 3D supervision. In *NIPS*, 2016. 2
- [57] Guandao Yang, Yin Cui, Serge Belongie, and Bharath Hariharan. Learning single-view 3D reconstruction with limited pose supervision. In *ECCV*, 2018. 2
- [58] H. Zhang, C. Wu, J. Zhang, and J. Deng. Variational mesh denoising using total variation and piecewise constant function space. 21(7):873–886, 2015. 3
- [59] Jun-Yan Zhu, Zhoutong Zhang, Chengkai Zhang, Jiajun Wu, Antonio Torralba, Josh Tenenbaum, and Bill Freeman. Visual object networks: Image generation with disentangled 3D representations. In *NIPS*, 2018. 2, 3

A. Additional qualitative results

In this section, we present additional qualitative results from our models using different parametrizations and training settings, on each of the five object classes.

Random (uncurated) samples generated by the models trained on ShapeNet renderings of cars are given in Figure 8 and Figure 9; chairs in Figure 10, Figure 11 and Figure 12; airplanes in Figure 13, Figure 14 and Figure 15; and sofas in Figure 16, Figure 17 and Figure 18. See Section 6.1 for discussion of these results.

Random (uncurated) samples from the models trained on natural images of cars are given in Figure 19 and Figure 20, and birds in Figure 21 and Figure 22. See Section 6.2 for discussion of these results.

Additional reconstruction results on natural images of cars are given in Figure 23, and birds in Figure 24. See Section 6.3 for discussion of these results.

B. Network architectures

In this section, we describe the neural network architectures for each component of our model.

Notation. We use the following notation for network layers:

- Convolutional layers are denoted by $\text{Conv}(\text{channels}, \text{filter size})$; stride is one unless otherwise specified
- Densely-connected layers are denoted by $\text{Dense}(\text{channels})$
- Bilinear upsampling layers are denoted by $\text{Upsampling}(\text{factor})$
- $\text{Reshape}(\text{shape})$ denotes reshaping the input tensor to the given shape
- \oplus denotes vector concatenation
- σ denotes the logistic sigmoid function

When the input to a layer is not just the output of the previous layer, we indicate this input in a second pair of parentheses.

Encoders. The encoders consist of a shared CNN that extracts features from the input image, followed by densely-connected layers operating on the resulting features to give predictions for each latent variable. The feature extractor is as follows (all convolution/dense layers use relu activation and group-normalization):

- *input: RGB image*
- Conv($32, 3 \times 3$, stride = 2)
- Conv($64, 3 \times 3$)
- MaxPool(2×2)
- Conv($96, 3 \times 3$)
- MaxPool(2×2)
- Conv($128, 3 \times 3$)
- MaxPool(2×2)
- Conv($128, 4 \times 4$)
- Dense(128)
- *output: 128D feature vector f*

The shape encoder is as follows:

- *input: 128D feature vector f*
- Dense(32×2)
- *output: mean and stddev of 32D shape embedding $\mathbf{z}_{\text{shape}}$, latter with softplus activation*

The texture encoder is as follows:

- *input: 128D feature vector f and vector c of mean pixel colors clipping each face*
- Dense(96, activation = relu, group-normalization)(\mathbf{c}) $\oplus \mathbf{f}$
- Dense(128×2)
- *output: mean and stddev of 128D texture embedding $\mathbf{z}_{\text{color}}$, latter with softplus activation*

The background encoder is as follows:

- *input: 128D feature vector f*
- Dense($64, \text{activation} = \text{relu}$, group-normalization)
- Dense(16×2)
- *output: mean and stddev of 16D background embedding \mathbf{z}_{bg} , latter with softplus activation*

The pose encoder is as follows:

- *input: 128D feature vector f*
- Dense(5)
- *output: 2D offset in xz-plane; 3D log-scale*

Decoders. The decoders consist of densely-connected networks, taking the latent variables as input. The shape decoder $\text{dec}_{\text{shape}}$ is as follows:

- *input: 32D shape embedding $\mathbf{z}_{\text{shape}}$*
- Dense($32, \text{activation} = \text{elu}$)
- Dense($3N_V$)
- Reshape($N_V, 3$)
- *output: 3D offset vectors to be added to each of the N_V vertices of the base mesh*

The texture decoder $\text{dec}_{\text{color}}$ is as follows:

- *input: 128D texture embedding $\mathbf{z}_{\text{color}}$ and 32D shape embedding $\mathbf{z}_{\text{shape}}$*

- Dense(128, activation = elu)($\mathbf{z}_{\text{color}} \oplus \mathbf{z}_{\text{shape}}$) $+ \mathbf{z}_{\text{color}}$
- Dense(192, activation = elu)
- Dense($3N_F$)
- Reshape($N_F, 3$) / $10 + \frac{1}{2}$
- *output: RGB colors for each of the N_F faces of the mesh*

The background decoder, used only in setting (NO-MASK), is as follows (all convolution layers use elu activation, except the last):

- *input: 16D background embedding*
- Reshape($1 \times 1 \times 16$)
- Upsample($4 \times$)
- Conv($64, 3 \times 3$)
- Upsample($2 \times$)
- Conv($32, 3 \times 3$)
- Upsample($2 \times$)
- Conv($16, 3 \times 3$)
- Upsample($2 \times$)
- $\sigma(\text{Conv}(3, 3 \times 3) / 2)$
- Upsample($4 \times$ or $6 \times$)
- *output: RGB background image*

Baseline VAE. The encoder for the baseline VAE uses the same feature extractor as the main model. To convert the features to the mean and variance of the latent variables, the following architecture is used:

- *input: 128D feature vector f*
- Dense(160×2)
- *output: mean and stddev of 160D image embedding, latter with softplus activation*

The decoder is as follows (all layers use elu activation, except the last):

- *input: 160D image embedding*
- Dense(256)
- Reshape($4 \times 4 \times 16$)
- Conv($128, 3 \times 3$)
- Upsample($2 \times$)
- Conv($64, 3 \times 3$)
- Upsample($2 \times$)
- Conv($32, 3 \times 3$)
- Upsample($2 \times$)
- Conv($24, 3 \times 3$)
- Upsample($2 \times$)
- Conv($16, 3 \times 3$)
- Upsample($2 \times$)
- Conv($3, 3 \times 3$) $+ \frac{1}{2}$
- *output: RGB image*

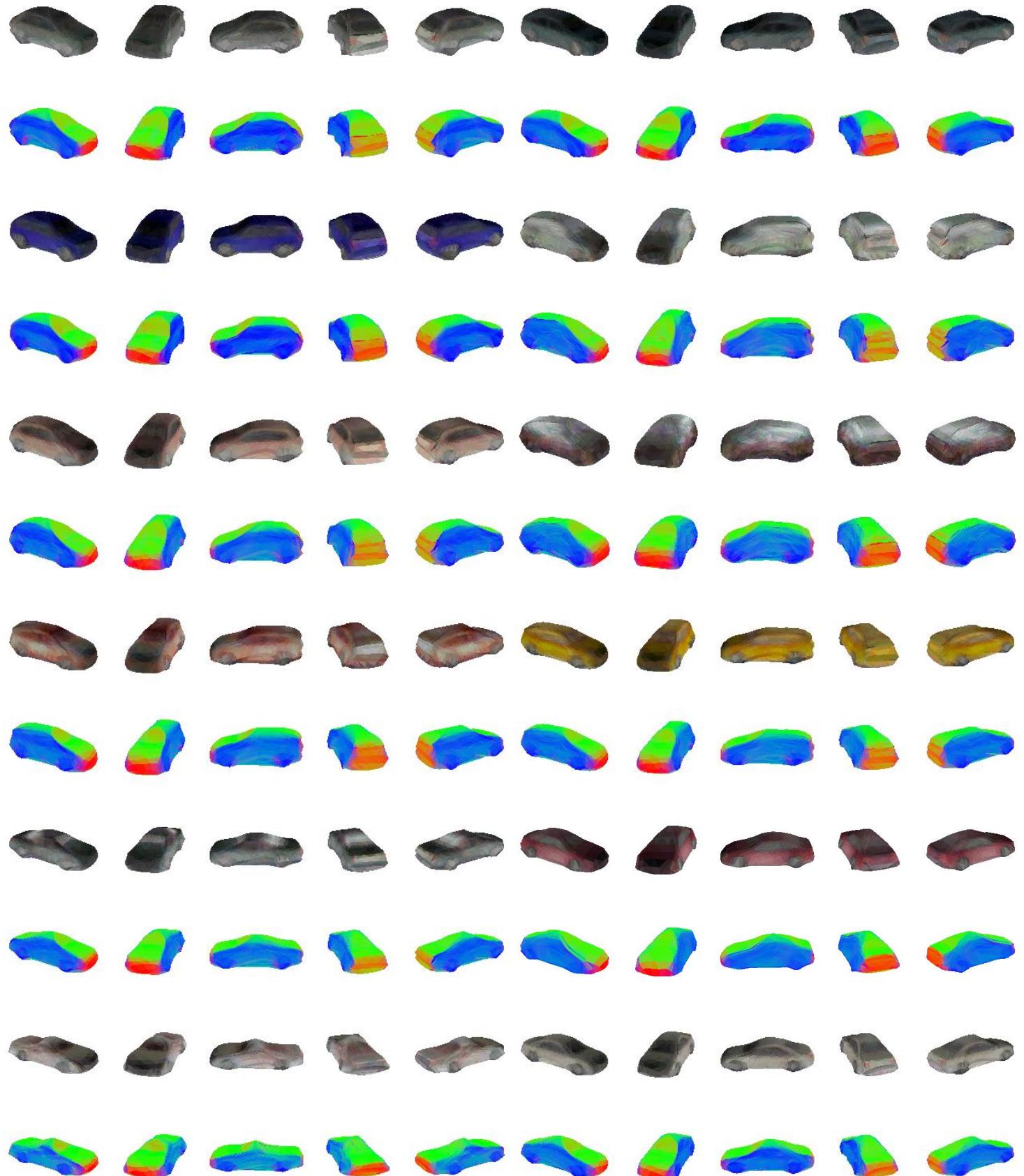


Figure 8. Examples of cars generated by our model, in setting (MASK) with parametrization (DENSE).

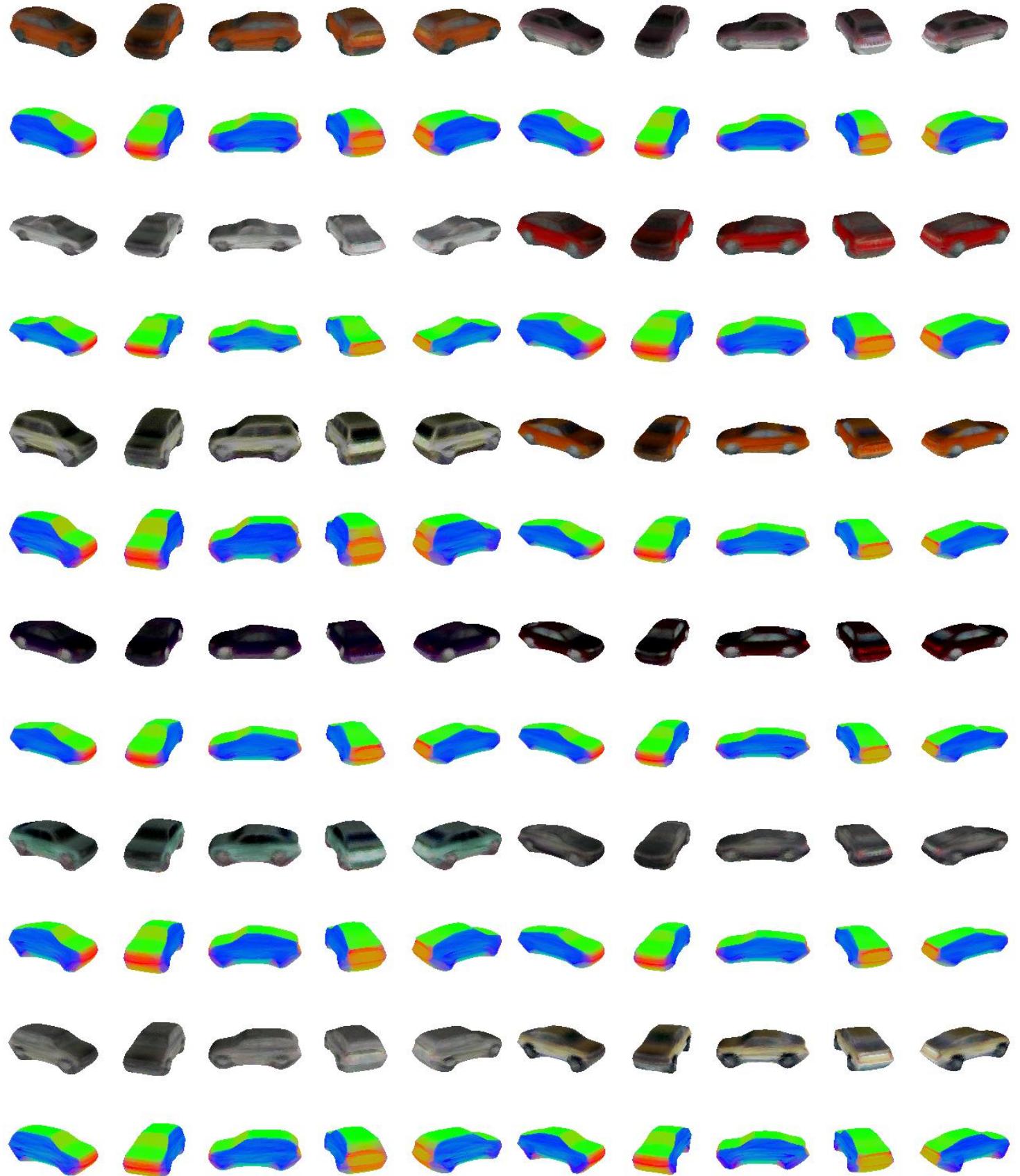


Figure 9. Examples of cars generated by our model, in setting (NO-MASK) with parametrization (DENSE).

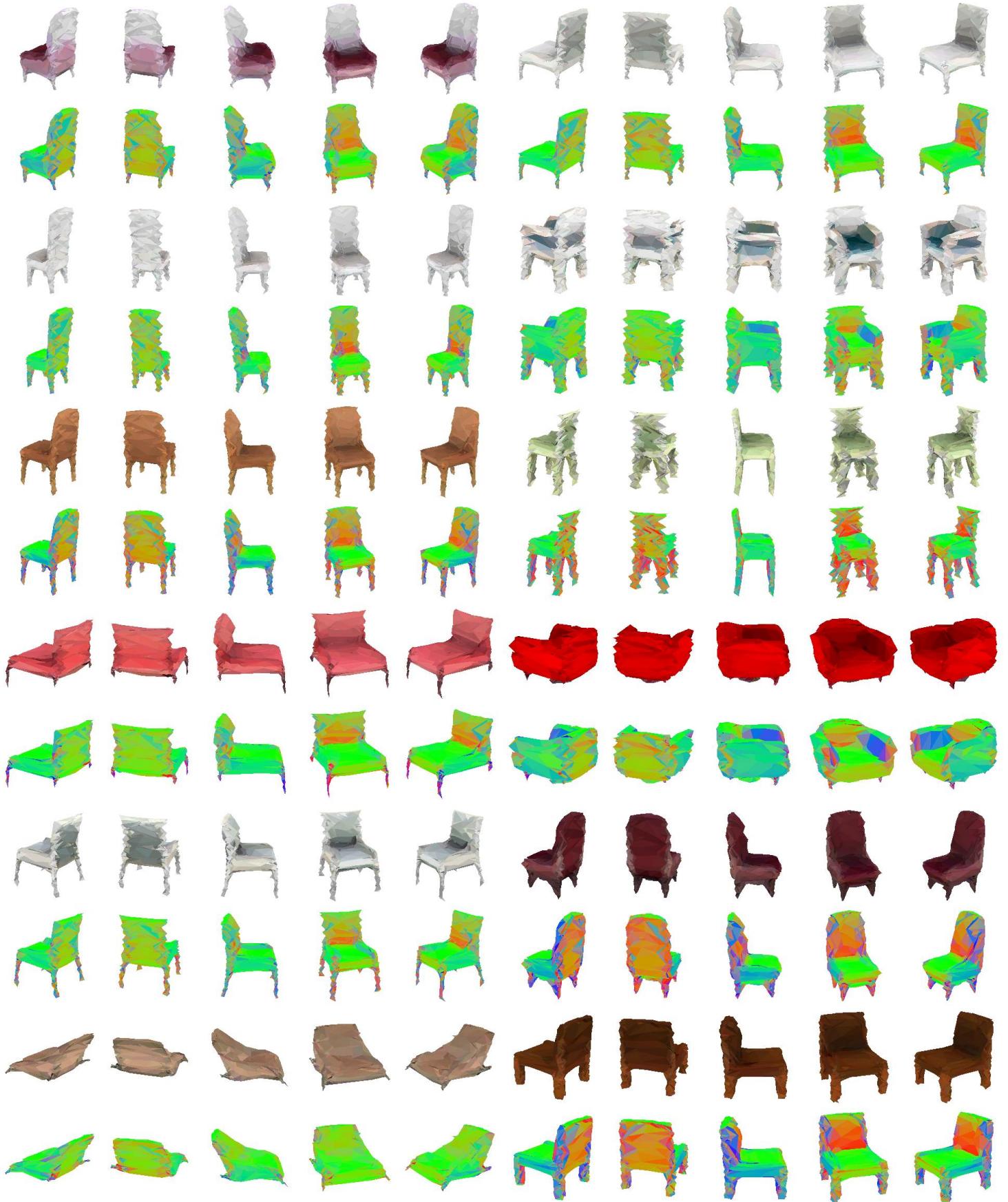


Figure 10. Examples of chairs generated by our model, in setting (MASK) with parametrization (DENSE).

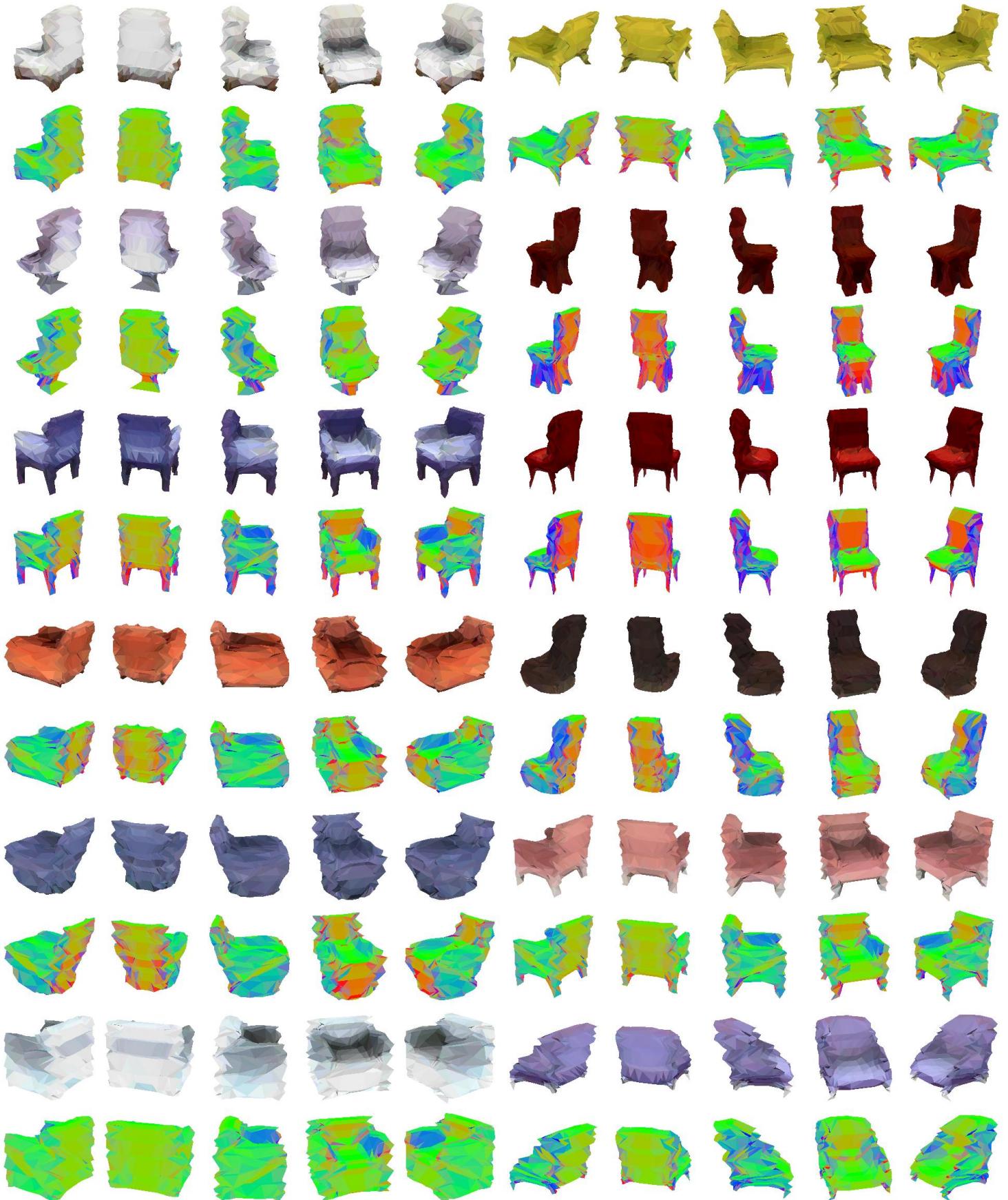


Figure 11. Examples of chairs generated by our model, in setting (MASK) with parametrization (PUSHING).

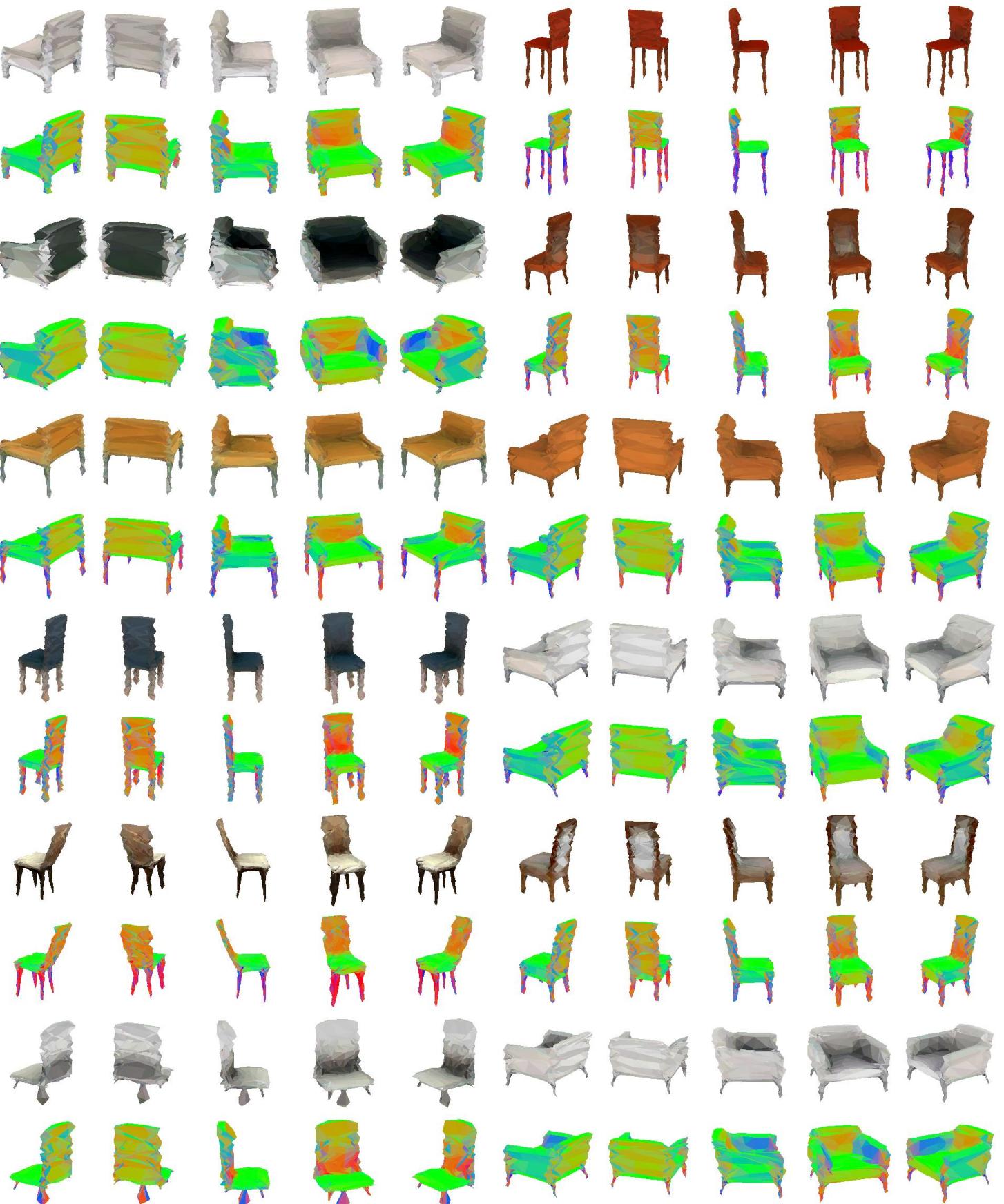


Figure 12. Examples of chairs generated by our model, in setting (NO-MASK) with parametrization (DENSE).

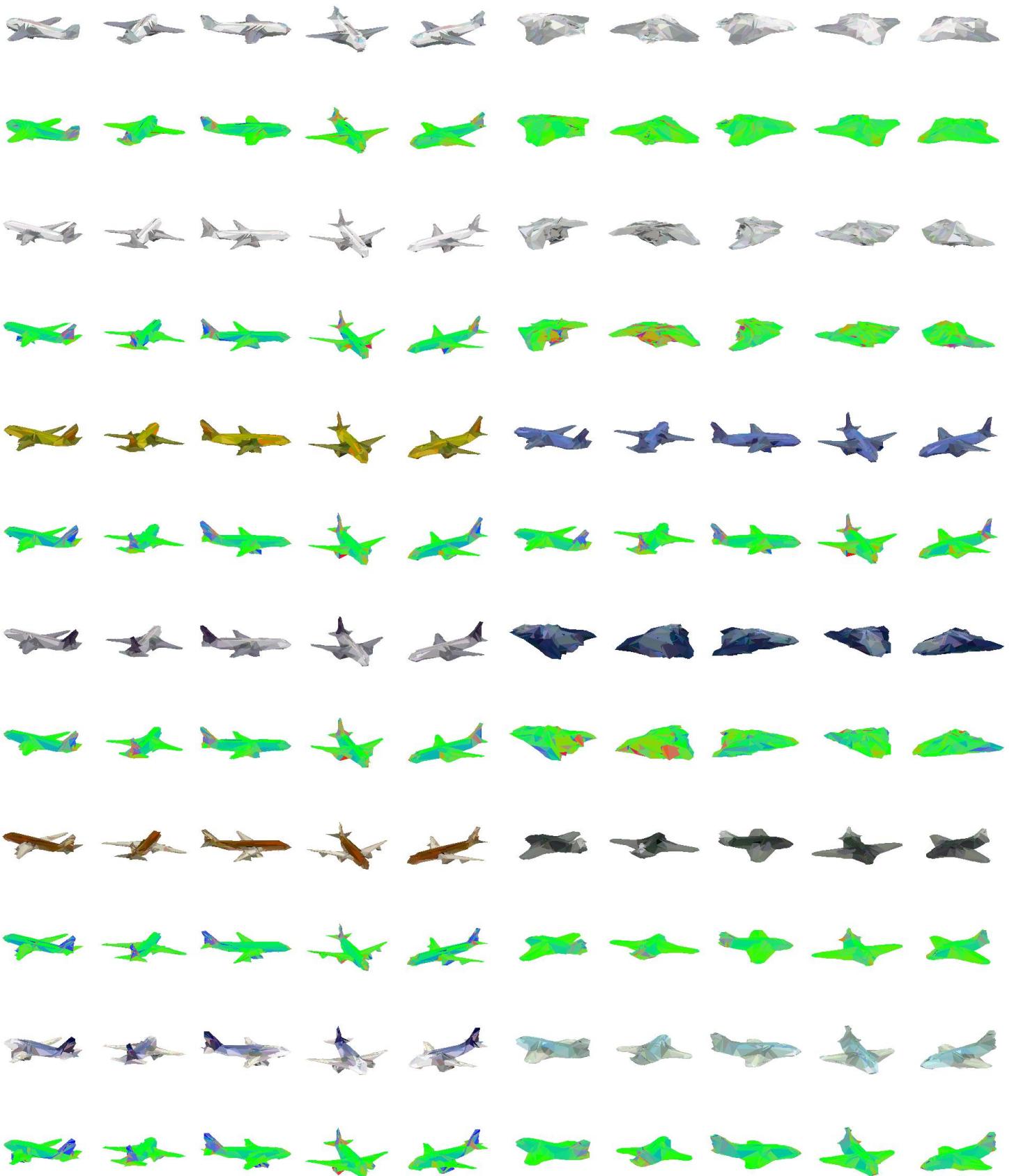


Figure 13. Examples of airplanes generated by our model, in setting (MASK) with parametrization (DENSE).

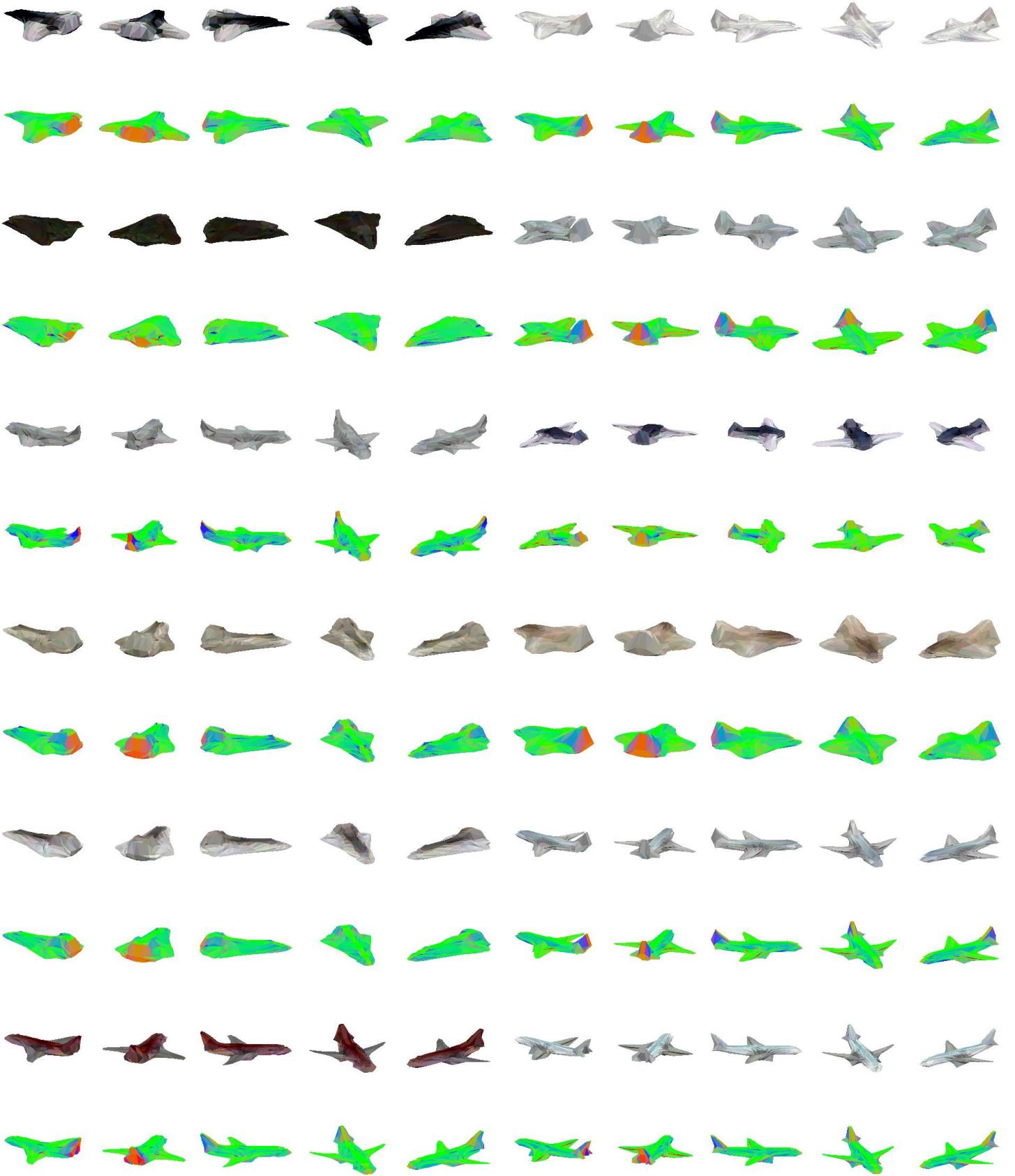


Figure 14. Examples of airplanes generated by our model, in setting (MASK) with parametrization (PUSHING).

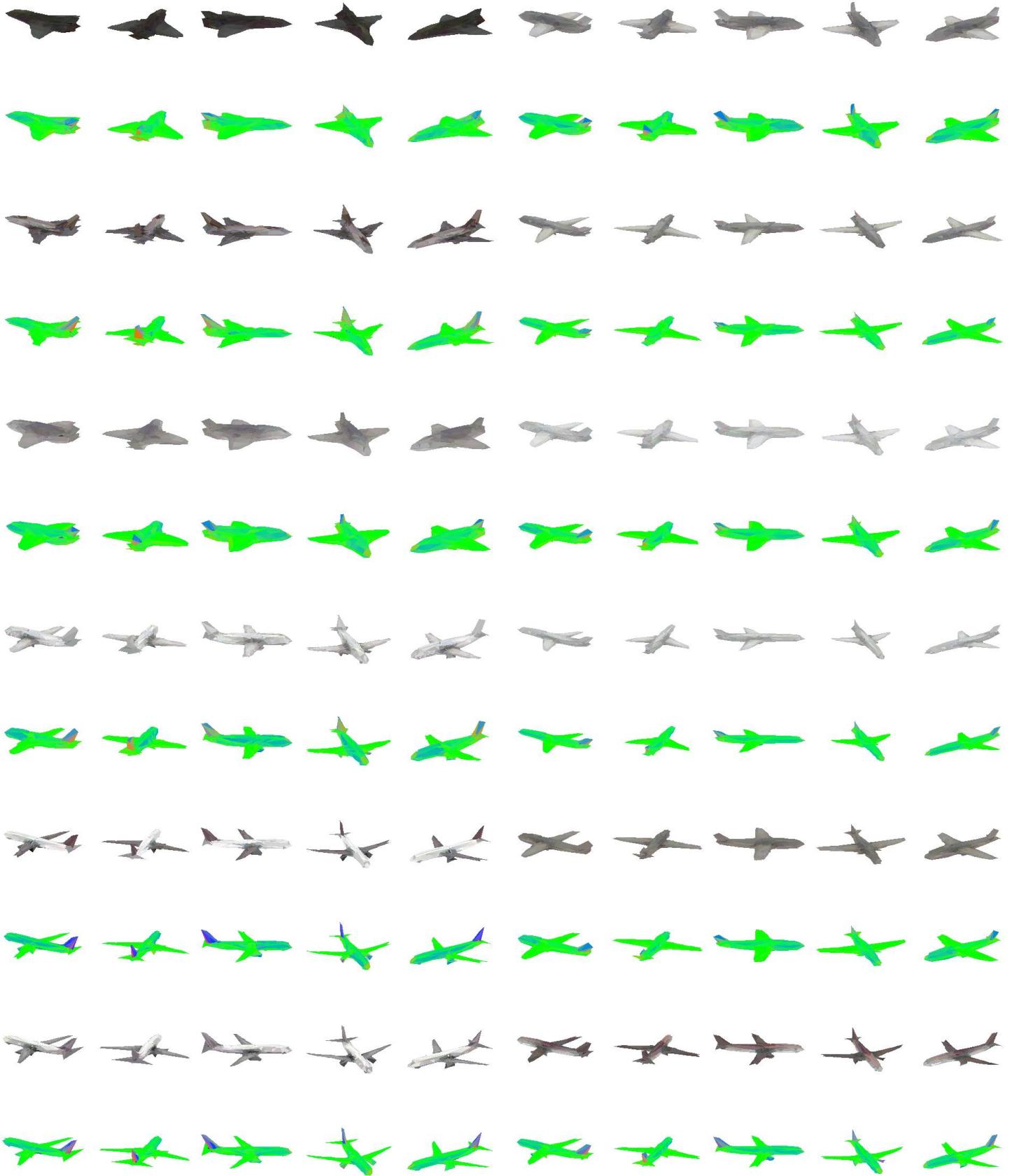


Figure 15. Examples of airplanes generated by our model, in setting (NO-MASK) with parametrization (DENSE).

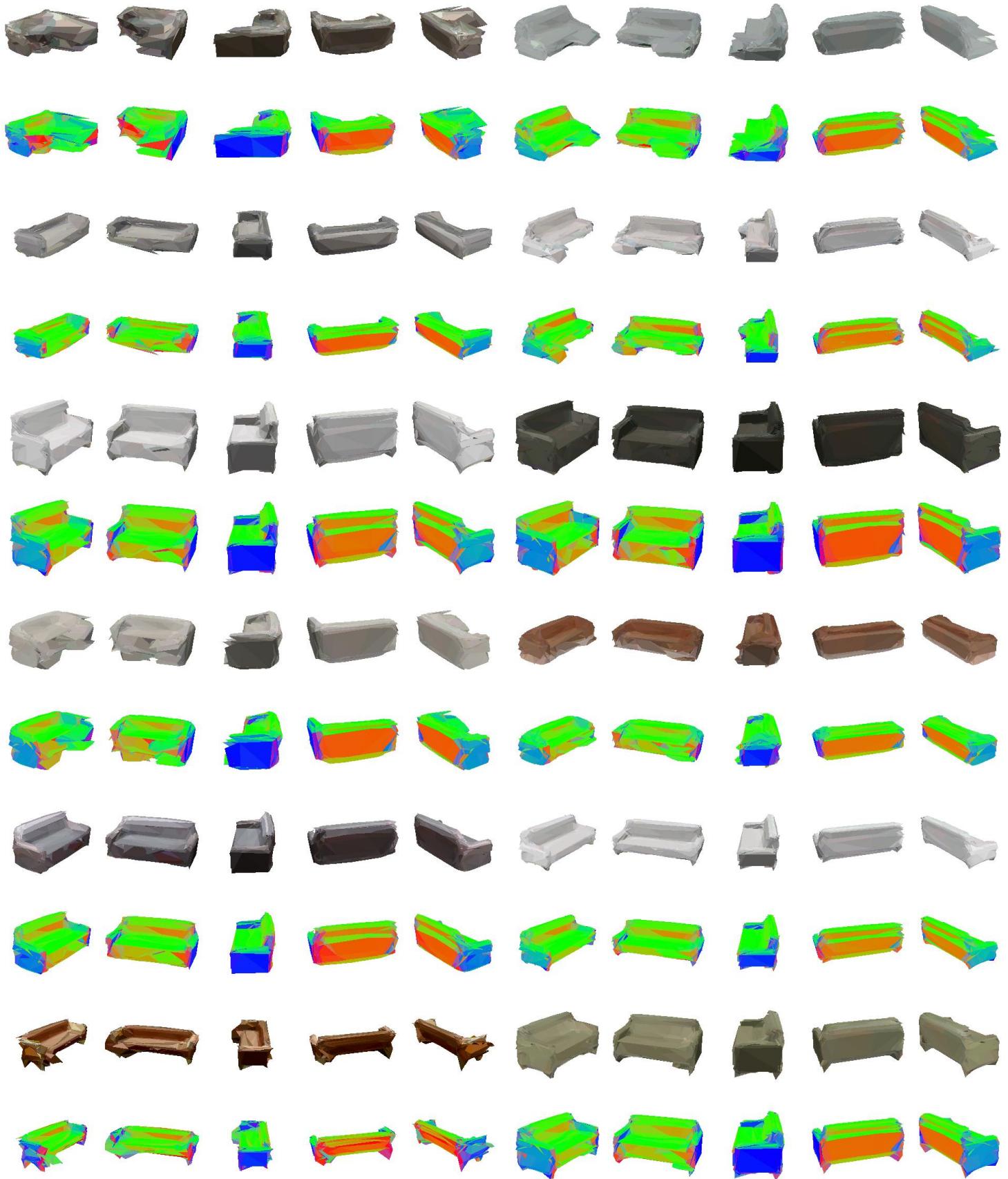


Figure 16. Examples of sofas generated by our model, in setting (MASK) with parametrization (DENSE).

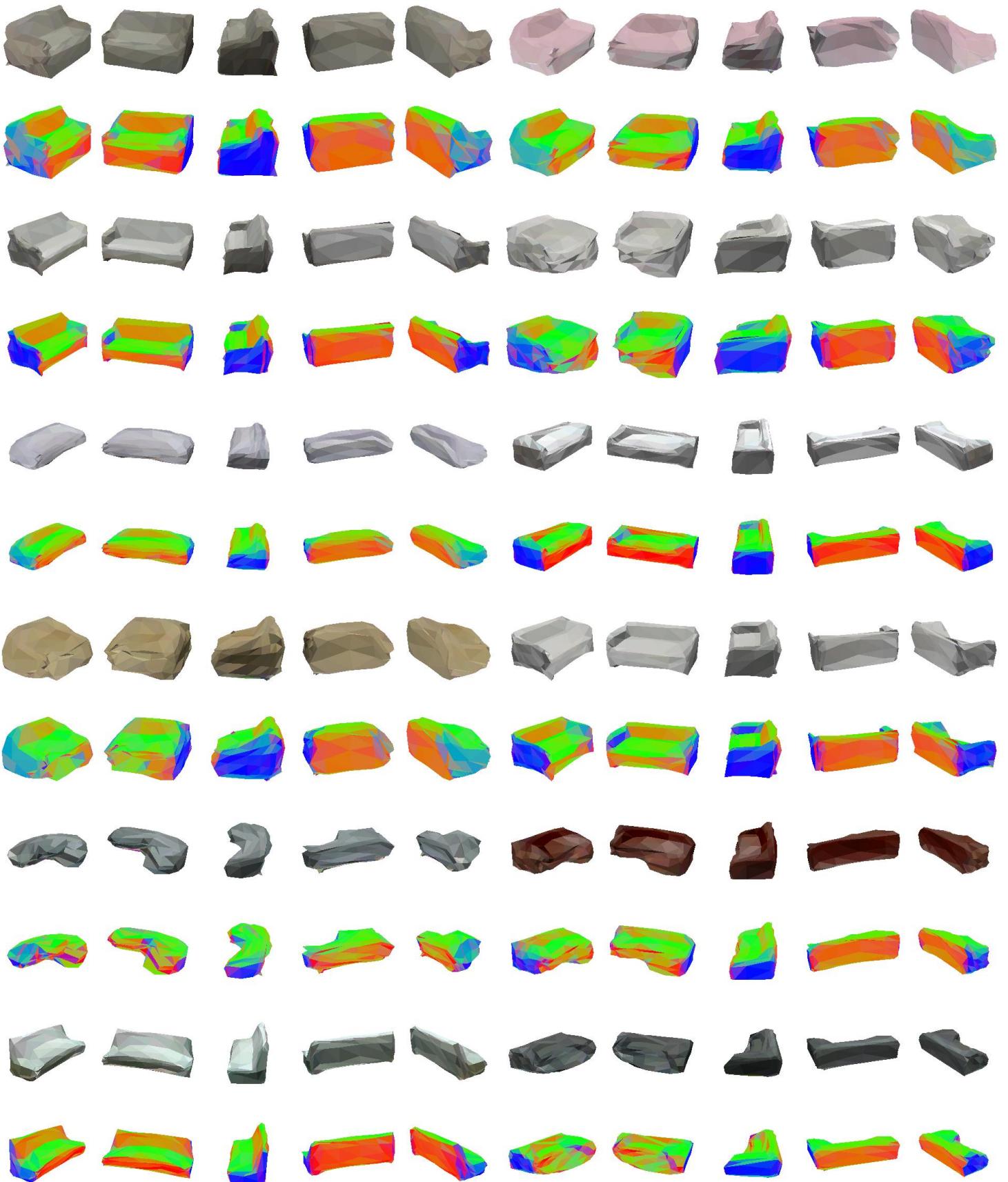


Figure 17. Examples of sofas generated by our model, in setting (MASK) with parametrization (PUSHING).

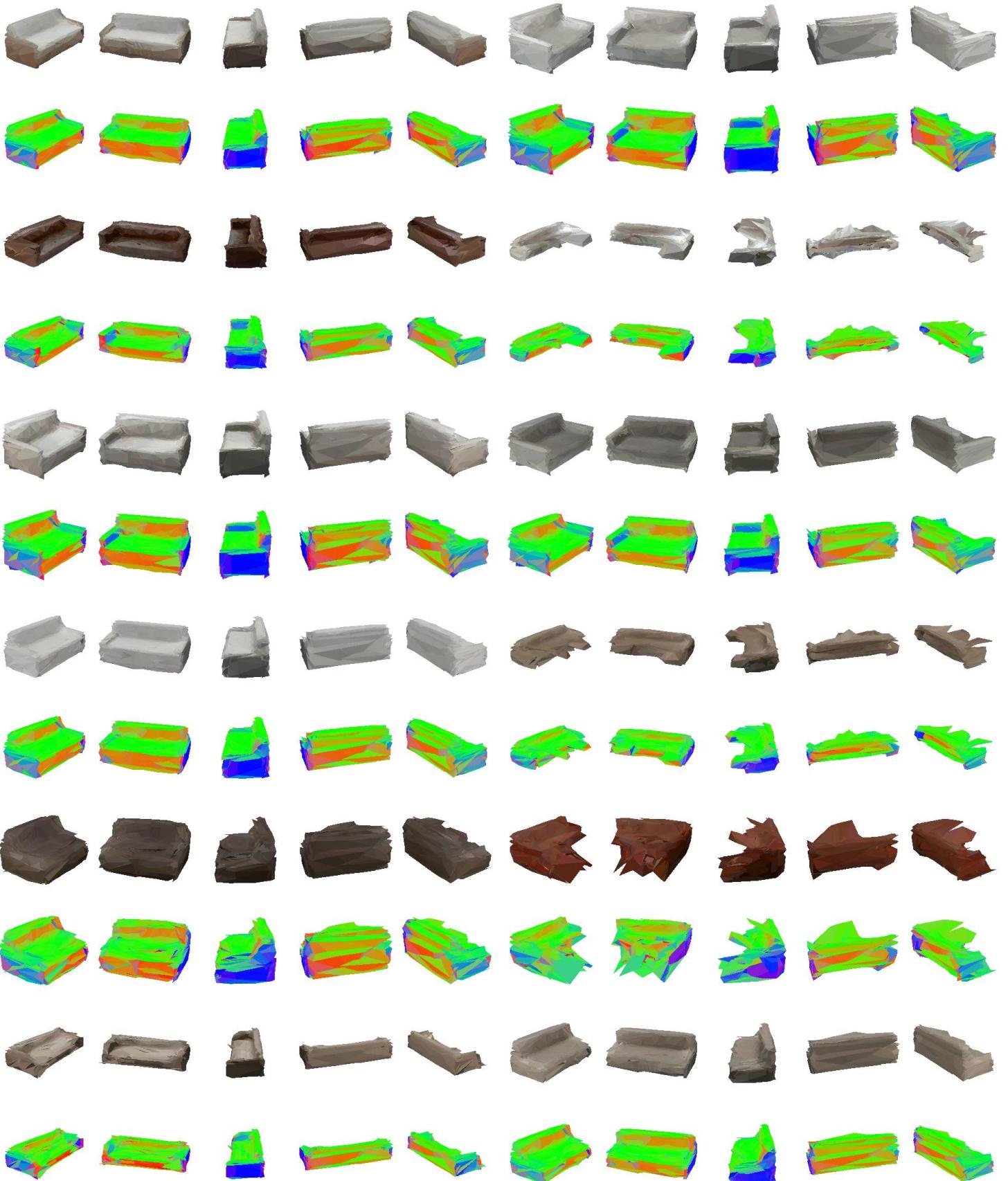


Figure 18. Examples of sofas generated by our model, in setting (NO-MASK) with parametrization (DENSE).

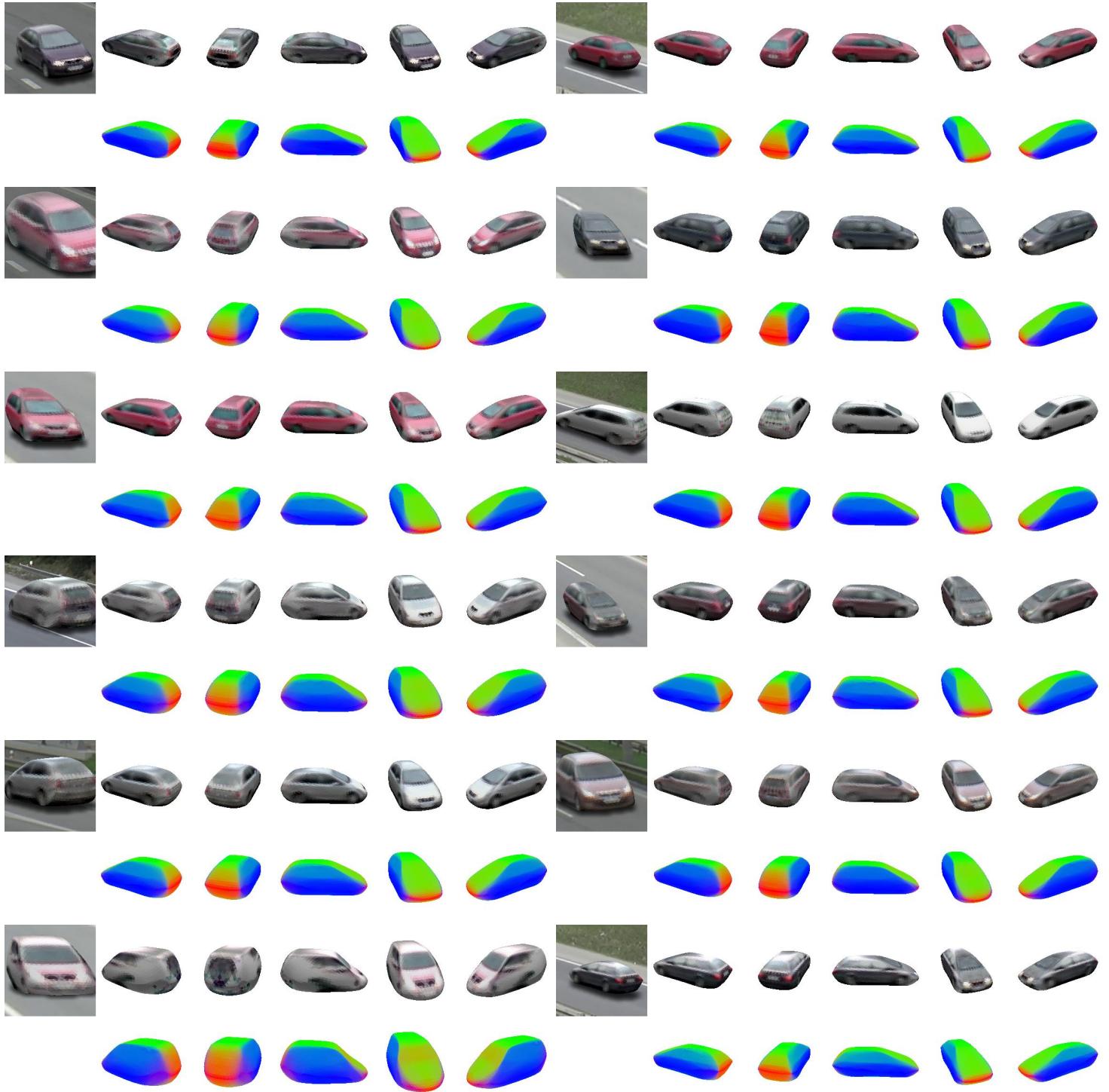


Figure 19. Examples of cars generated by our model, in setting (MASK).

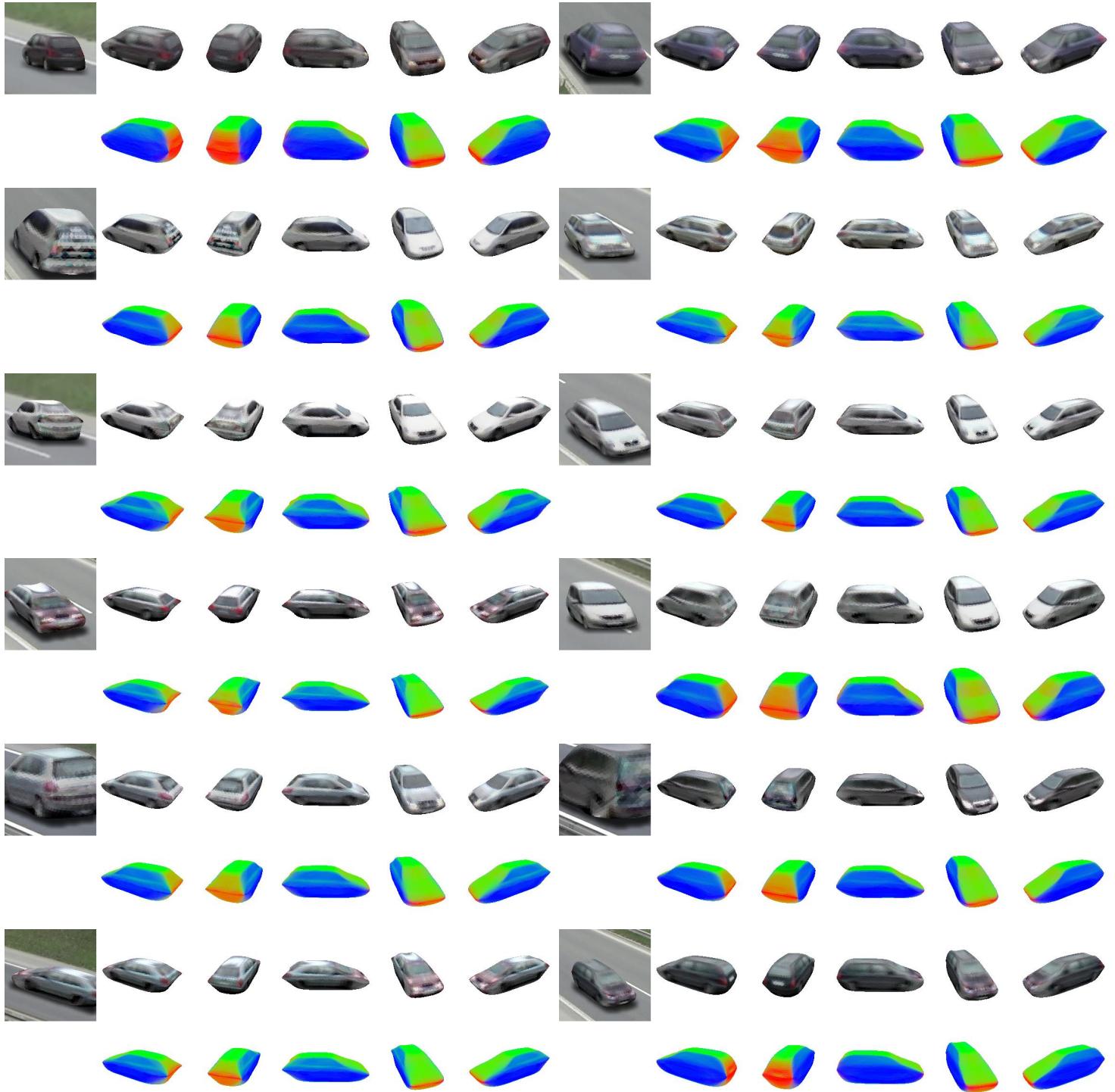


Figure 20. Examples of cars generated by our model, in setting (NO-MASK).

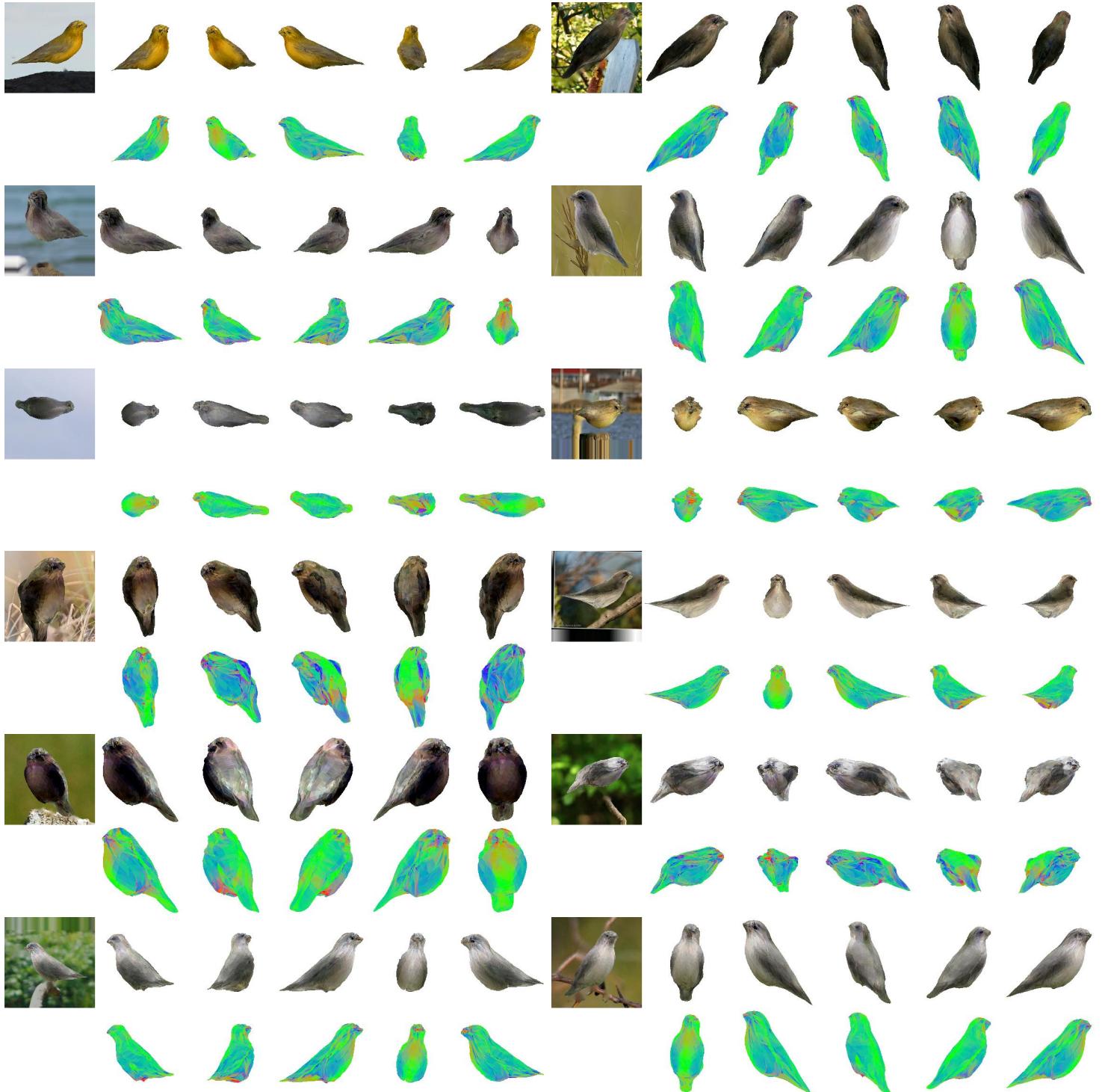


Figure 21. Examples of birds generated by our model, in setting (MASK).

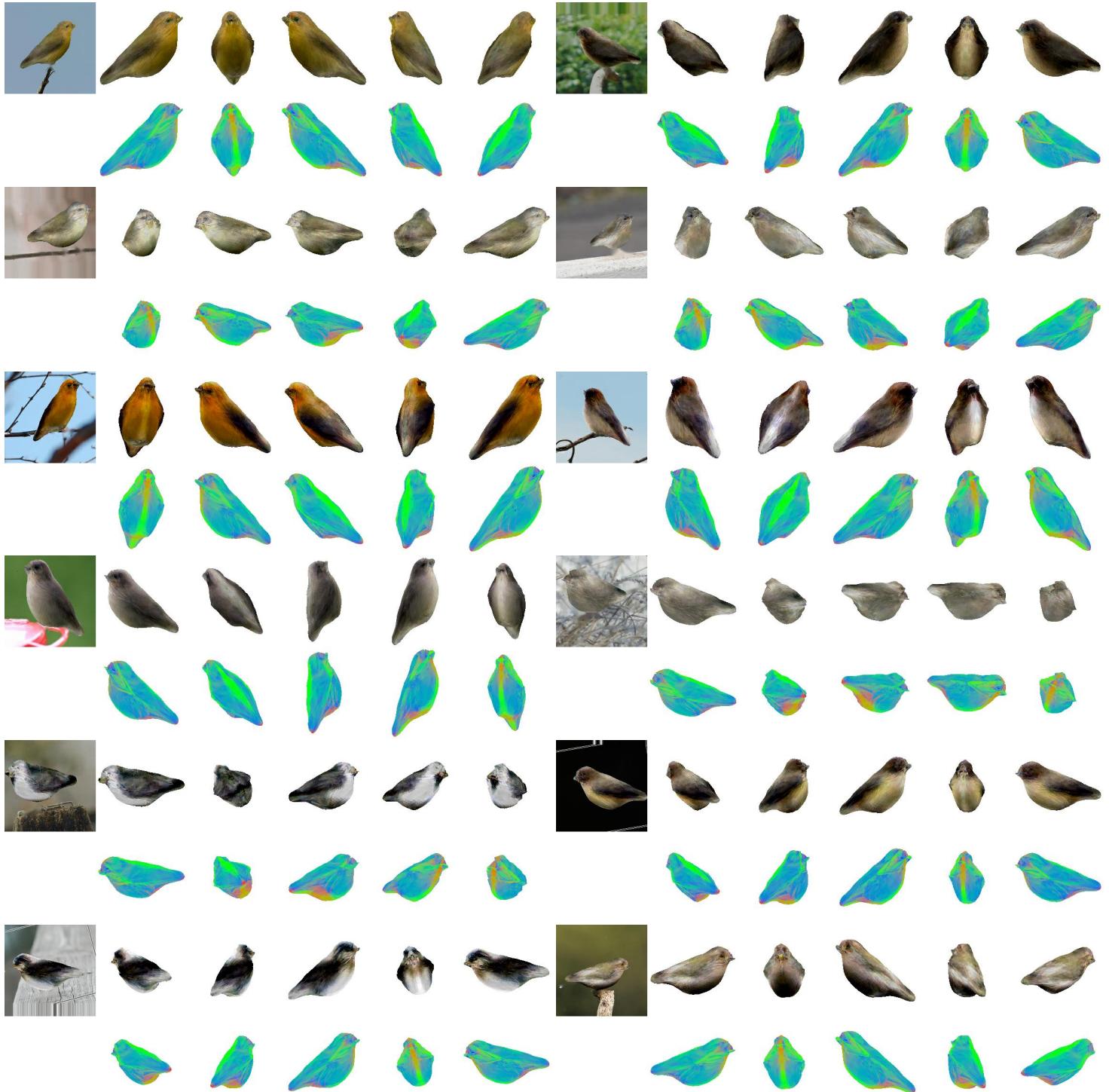


Figure 22. Examples of birds generated by our model, in setting (NO-MASK).

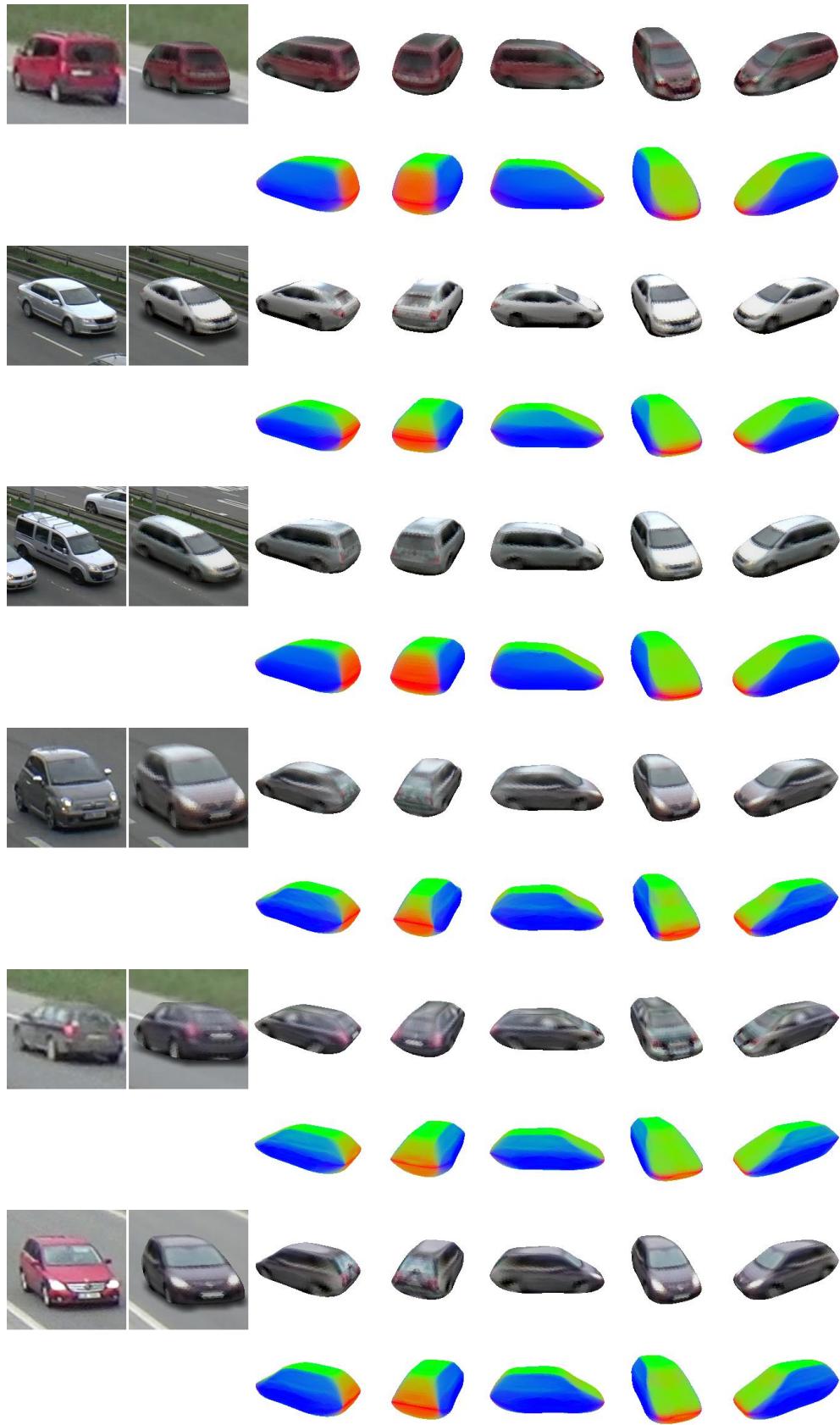


Figure 23. Additional examples of car reconstructions, in setting (MASK) (top three) and (NO-MASK) (bottom three). The left-hand image is the input to our model; the next is our reconstruction, rendered over the ground-truth background; the remaining five columns are different views of the reconstructed instance, with normal-maps below

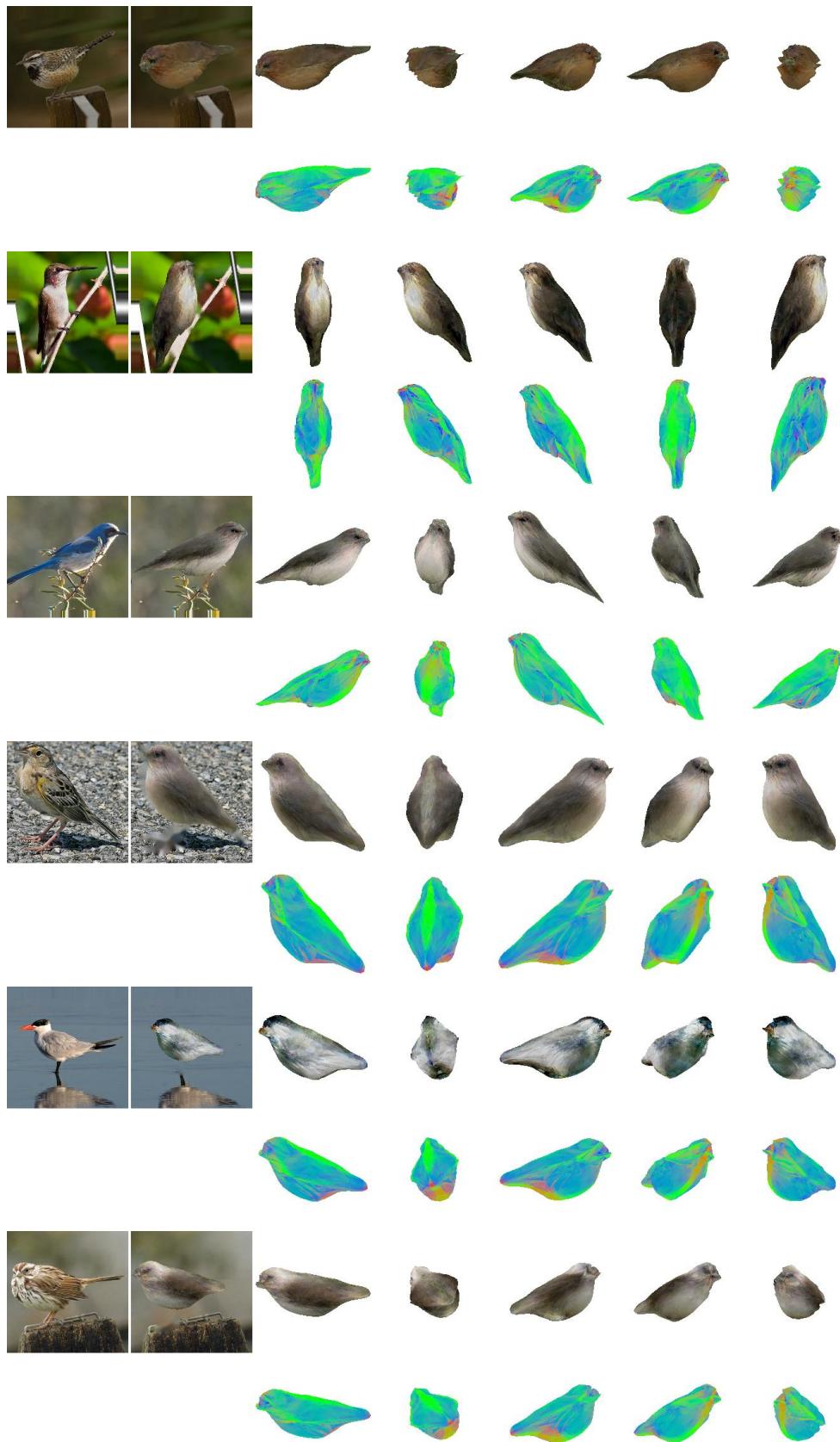


Figure 24. Additional examples of bird reconstructions, in setting (MASK) (top three) and (NO-MASK) (bottom three). The left-hand image is the input to our model; the next is our reconstruction, rendered over a pseudo-background (see text); the remaining five columns are different views of the reconstructed instance, with normal-maps below