

Supplemental Material for GaussiGAN: Controllable Image Synthesis with 3D Gaussians from Unposed Silhouettes

Youssef A. Mejjati¹Isa Milefchik²Aaron Gokaslan³Oliver Wang⁴Kwang In Kim⁵James Tompkin²¹ University of Bath² Brown University³ Cornell University⁴ Adobe⁵ UNIST

This supplemental document contains additional experiments on varying K and the dataset size, comparisons against a geometry proxy-based method for scene generation [5], and comparisons to an unsupervised 2D parts-based inference method [6] that could not fit into the main paper (Sec. A). We also provide additional and fuller results reporting (Sec. B), more detailed discussion of how to estimate Gaussian covariance matrices (Sec. C), additional details on mask texturing (Sec. D), network architecture details (Sec. E), and finally a derivation of the analytic Gaussian projection derivative (Sec. F).

Please also see our supplemental video, which includes examples of interactive editing and results showing rotations of the recovered 3D Gaussians and their use in generating masks and textures, in comparison to HoloGAN, PlatonicGAN, and Liao et al.

A Additional comparisons

A.1 Varying K and dataset size.

We provide K at training time, which is simple to estimate by hand for many objects, e.g., one each for the body and head, one for each limb. As K varies, our density losses over random rotations encourage detail where it is required (Fig. 1, left). Too few K diminish pose or shape; too many K leads to redundant Gaussians. As we set a minimum size, these appear as ‘little dots’ (Fig. 1, right) and can be ignored without affecting downstream tasks. For more control, a user could pre-define the canonical \mathcal{G}^c from which a set of per-image deformations is learned. We also shows how the Gaussians are still usefully recovered as input data decreases 64× in number (Fig. 1, right), though with less mask detail.

A.2 Liao et al.’s [5] method on our data

Liao et al. [5]’s method uses cube and sphere mesh proxies to represent multiple simple scene objects, and this allows control in image generation over camera rotation and object rotation. Our data has one complex object with deforming parts. Figure 3 shows that these proxies are promising but still unfortunately too simple for our data. Even though their generated images are of reasonable quality (though low in resolution and with minor artifacts), the pose of the object changes when

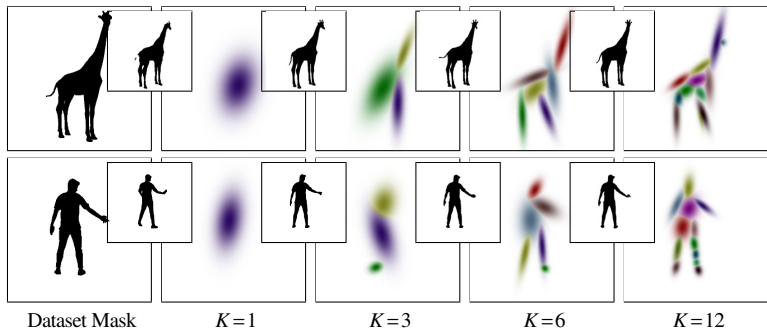


Figure 1: *Left*: Varying K produces levels of abstraction over the object’s shape and pose and so over generation control. At low K , only the major features are represented such as legs and neck. At higher K , details like individual legs ($K=12$, top) and leg parts (calf, thigh) appear with the detail required to model the pose variation, e.g., in *Manuel* (bottom), the right leg moves more in the animation and gains a knee at $K=12$.

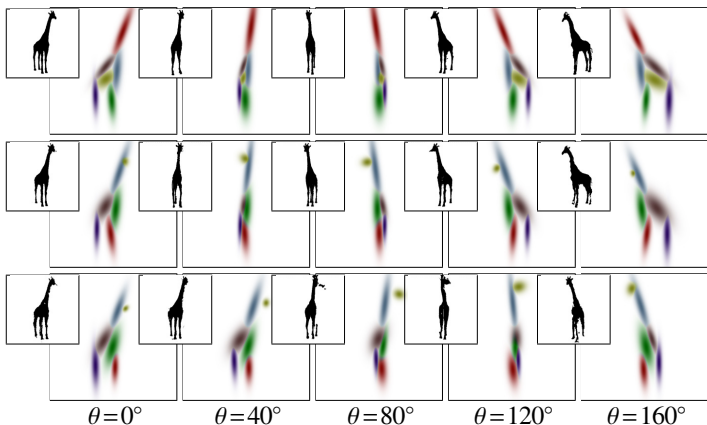


Figure 2: Randomly sampling sparser datasets still recovers the coarse 3D structure of the input object. Rows 0, 1, 2, 3 use $\frac{1}{8}$, $\frac{1}{16}$, $\frac{1}{32}$, $\frac{1}{64}$ of images in the training set; approximately 280, 140, 70, 35 images respectively. Colorings are different across rows.

rotating, e.g. the *Giraffe* neck bends while rotating. One dataset fails to rotate at all—*Pegasus*. Further, while their RGB appearance is often as vivid as the input, the texture is not consistent when rotating the camera, e.g., lighting variation, as the proxy geometry is not sufficiently descriptive to allow separation of shape and texture variation. Liao et al. is designed primarily for multiple objects; future work could build upon both methods to handle multiple complex objects.

A.3 Lorenz et al. [6] 2D part discovery on our data

For parts-based discovery, we compare our 3D parts to the 2D part maps from Lorenz et al. [6]. For fair comparison, we train the method only on mask images. The discovered parts are relevant, though some areas miss representation (Fig. A.3, *Maple*), and with less conformance to the underlying 3D space (e.g., failing to rotate correctly with the object, *Carla*).

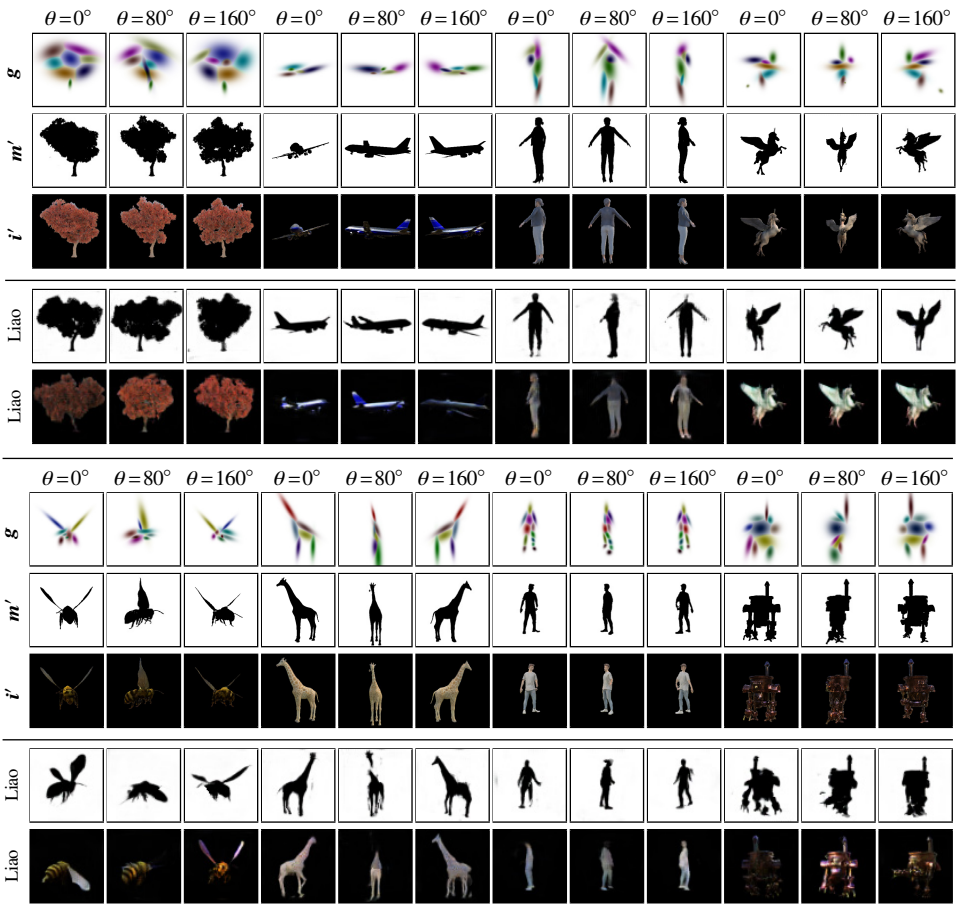


Figure 3: Please zoom in to see detail. Rows in each block: Reconstructed Gaussians, masks, and RGB images, across three output angles and with any texture-specific latent variables fixed, with comparisons to Liao *et al.* [5] and on just masks and just RGB foregrounds. Top block of five rows: Datasets of objects of fixed pose showing increasing shape complexity: Maple, Airplane, Carla, Pegasus. Bottom block of five rows: Datasets of animated objects with varying pose showing increasing shape complexity: Bee, Giraffe, Manuel, Old Robot.

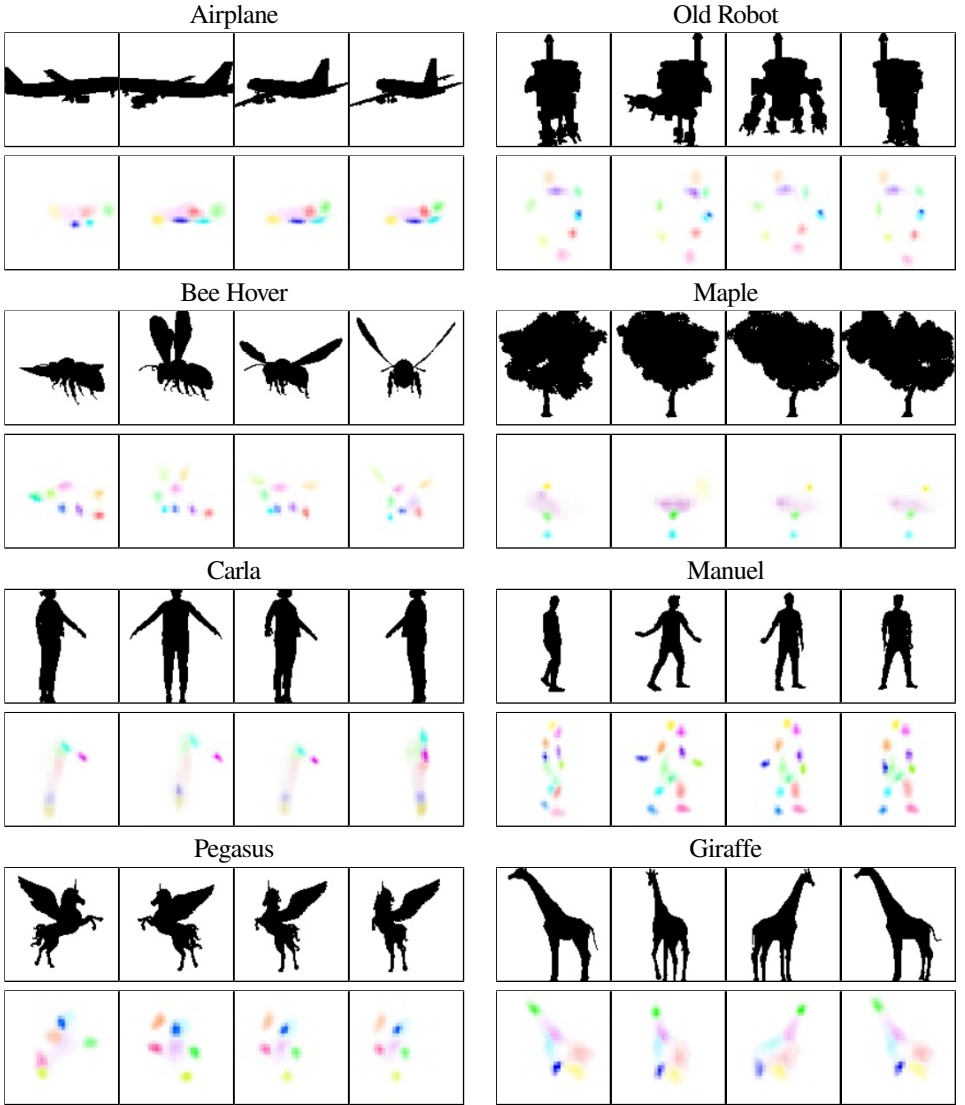


Figure 4: Results for Lorenz et al. [1] trained on masks only from our various datasets. Each has been trained for 50,000 iterations. The top row in each subfigure is the input into the network and the bottom row is the learned 2D part map. The part maps generally correspond to different areas of the object, but can struggle to represent object rotation (e.g., legs do not rotate in Manuel). For the ‘dense’ class of *Maple*, only the the relatively static trunk is consistently labeled.

B Additional results

We show additional qualitative results across all datasets in Figures 7–14.

Interactive editing. In our supplemental video, we show an application of user control over the recovered Gaussian proxies. We built a demo that exploits our Gaussian proxy’s form to allow simple drag-and-drop object part translation, anisotropic scale, and rotation, plus camera control and lighting variation via \mathbf{z}_t . This allows in-distribution editing of the poses, e.g., *Giraffe* neck bending and leg adjustments. It also allows some out-of-distribution adjustments, such as placing the giraffe in a combination of poses that were not in any one input example, or more ‘creative’ edits such as enlarging or elongating certain Gaussians.

For our interactive editing scenario, which is 2D, each individual generated 2D image is coherent. However, while our approach recovers a coarse 3D ‘rig’ or ‘artist’s mannequin’ for an object, some texturing can be inconsistent in 3D (such as in rotation animations) as we only affect a 2D generator. While this gives high resolution, some fine detail can appear to shimmer as the camera rotates (e.g., on the *Maple* scene). This is because the chosen K coarse Gaussians do not provide sufficiently localized conditioning for the small leaf features as the camera varies by small angles.

Adaptation to background changes. Figure 5 shows that when the background becomes darker, the generated foreground also becomes darker, and vice versa. In addition, we show in Figure 6 that our generated texture is still reasonable when given backgrounds that are out of distribution. This effect happens automatically through conditioning foreground generation on the background image.

Quantitative evaluations. In Tables 1 and 2, we report quantitative metrics for each dataset independently; these were averaged in the main paper due to space limitations.

Table 1: *Top*: $KID \times 100 \pm STD \times 100$ and FID values (lower is better). *Bottom*: Mean IoU $\times 100$ (higher is better) and DSSIM $\times 100$ (lower is better) for reconstructed masks over the test set.

On RGB	Giraffe		Manuel		Maple		Carla	
	KID ▼	FID ▼	KID ▼	FID ▼	KID ▼	FID ▼	KID ▼	FID ▼
Ours	2.72 \pm 0.41	53.96	4.24 \pm 0.35	62.36	11.39 \pm 0.8	108.07	3.64 \pm 0.29	62.73
PlatonicGAN [10]	42.98 \pm 0.68	327.99	63.49 \pm 0.77	435.38	53.95 \pm 1.29	364.83	43.48 \pm 0.64	338.58
HoloGAN [10]	39.28 \pm 0.72	320.65	31.21 \pm 0.81	292.44	46.49 \pm 1.14	324.37	25.05 \pm 0.68	240.78
Liao et al. [10]	33.43 \pm 0.77	281.26	42.61 \pm 0.86	317.13	36.56 \pm 1.08	248.64	39.13 \pm 0.81	310.21
On masks	IoU ▲ DSSIM ▼		IoU ▲ DSSIM ▼		IoU ▲ DSSIM ▼		IoU ▲ DSSIM ▼	
Ours	83.96	6.22	81.46	5.66	86.51	21.87	89.39	6.22
PlatonicGAN [10]	67.70	23.85	67.54	16.57	92.77	33.09	83.66	13.20
HoloGAN [10]	30.68	39.38	42.62	18.40	62.74	64.24	38.60	38.83
Liao et al. [10]	31.27	60.82	41.13	31.14	68.94	88.61	36.50	58.93

Table 2: *Top*: $KID \times 100 \pm STD \times 100$ and FID values (lower is better). *Bottom*: Mean IoU $\times 100$ (higher is better) and DSSIM $\times 100$ (lower is better) for reconstructed masks over the test set.

On RGB	Bee		Pegasus		Old Robot		Airplane	
	KID ▼	FID ▼	KID ▼	FID ▼	KID ▼	FID ▼	KID ▼	FID ▼
Ours	7.62 \pm 0.44	124.18	11.03 \pm 0.61	156.33	15.93 \pm 0.76	186.81	16.71 \pm 1.17	188.07
PlatonicGAN [10]	38.06 \pm 0.73	324.56	59.86 \pm 1.29	456.46	56.49 \pm 0.94	439.7	39.31 \pm 0.79	314.61
HoloGAN [10]	18.32 \pm 0.44	252.95	27.94 \pm 0.94	291.92	41.29 \pm 1.34	372.52	32.18 \pm 0.91	291.29
Liao et al. [10]	24.77 \pm 0.59	265.52	32.61 \pm 0.76	325.35	37.83 \pm 1.04	333.02	26.64 \pm 0.82	261.98
On masks	IoU ▲ DSSIM ▼		IoU ▲ DSSIM ▼		IoU ▲ DSSIM ▼		IoU ▲ DSSIM ▼	
Ours	76.11	9.22	85.17	6.01	87.58	9.86	65.57	9.72
PlatonicGAN [10]	65.82	23.76	82.35	17.04	81.88	25.52	76.64	17.58
HoloGAN [10]	34.58	29.81	44.83	31.25	52.65	40.70	48.11	24.17
Liao et al. [10]	17.98	44.29	39.74	45.70	33.81	76.42	31.42	39.09

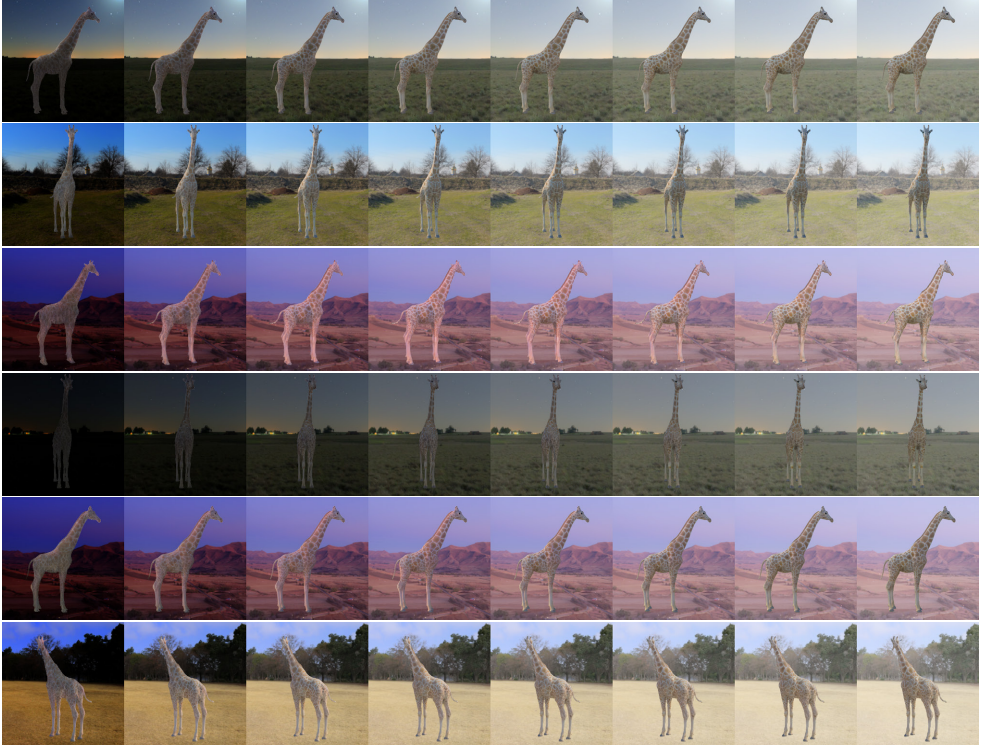


Figure 5: Lighting variation in the foreground as the background varies in intensity.

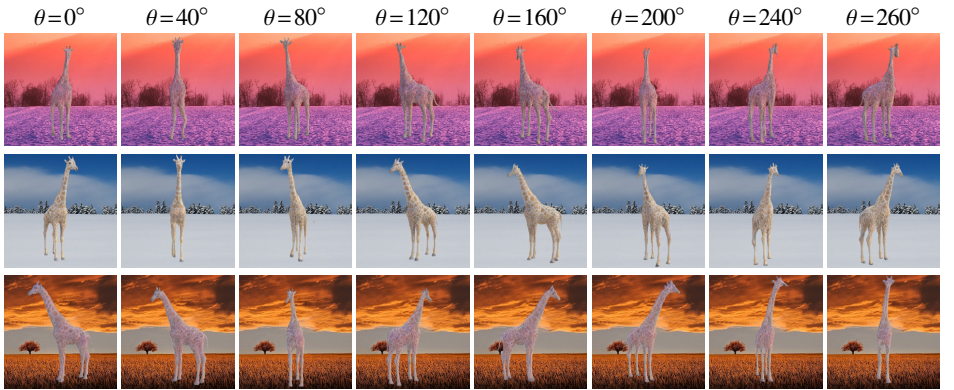


Figure 6: At test time, when giving significantly out of distribution backgrounds (e.g., fields of pink flowers under sunset, snowy landscape), our generator still produces reasonable results that match the lighting. Each row contains an example rotated from a different input instance at a different initial orientation.



Figure 7: Additional results for Giraffe, with two randomly sampled latent vectors for texture.

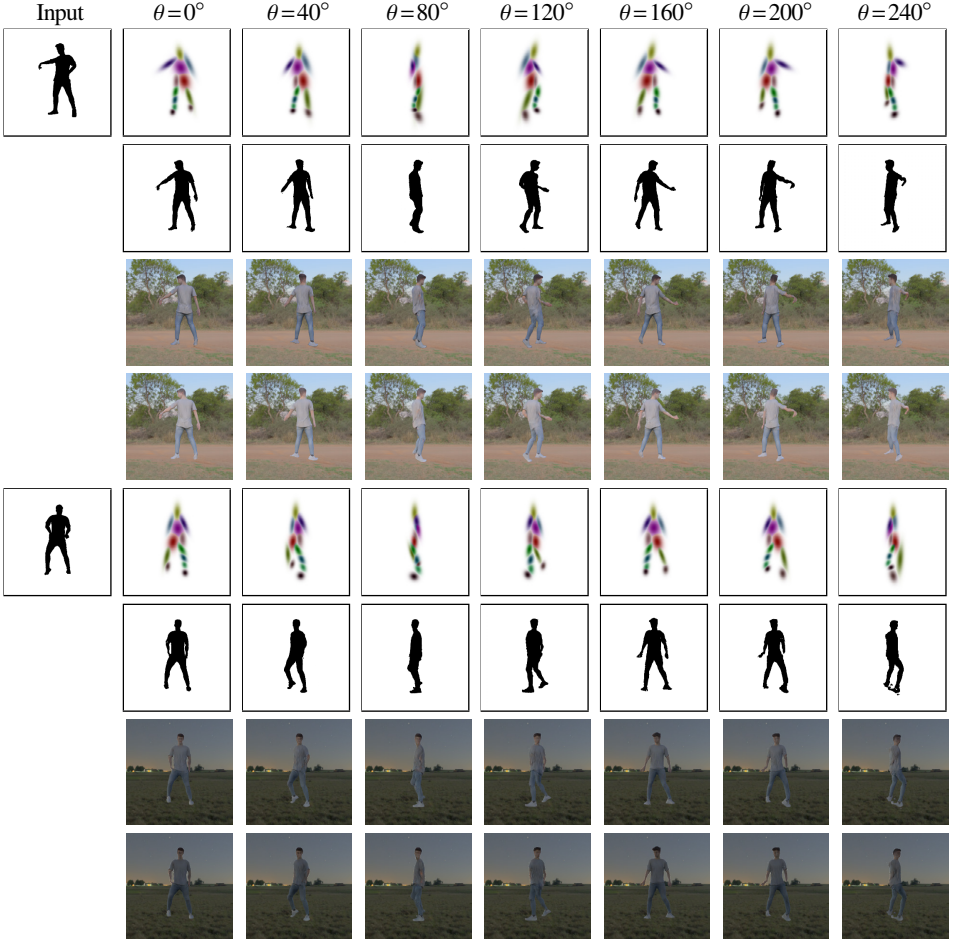


Figure 8: Additional results for Manuel, with two randomly sampled latent vectors for texture.

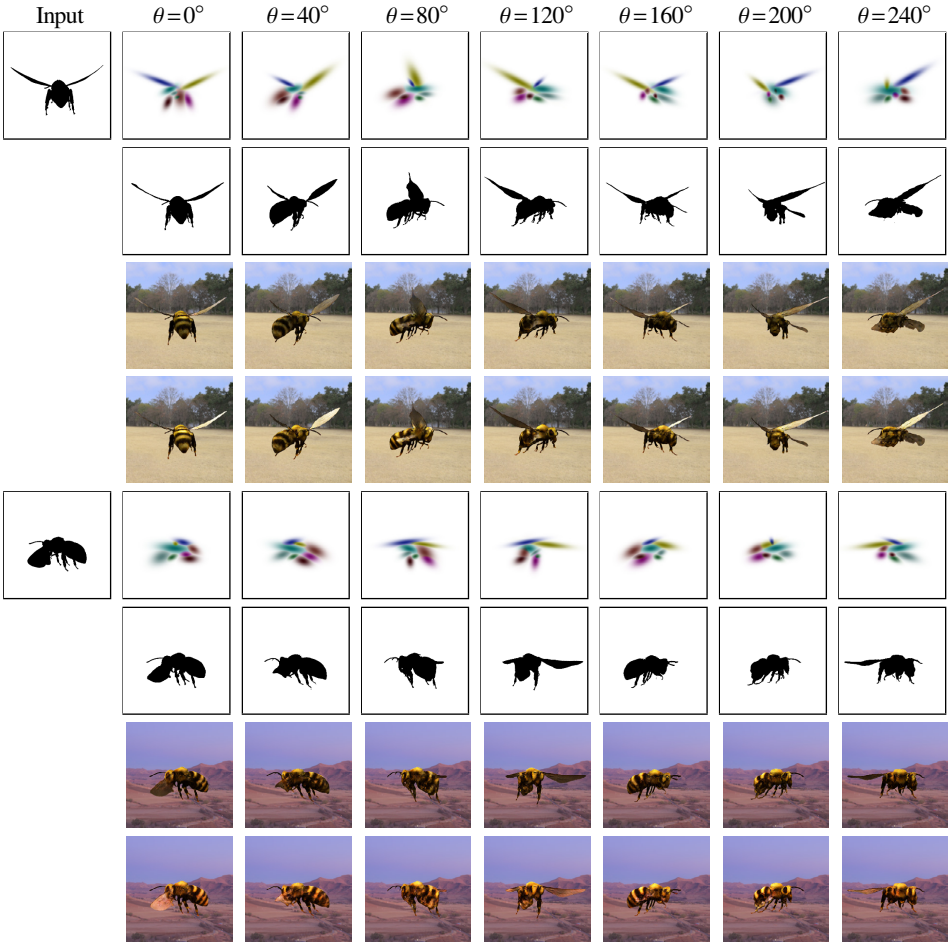


Figure 9: Additional results for beehover, with two randomly sampled latent vectors for texture.



Figure 10: Additional results for OldRobot, with two randomly sampled latent vectors for texture.

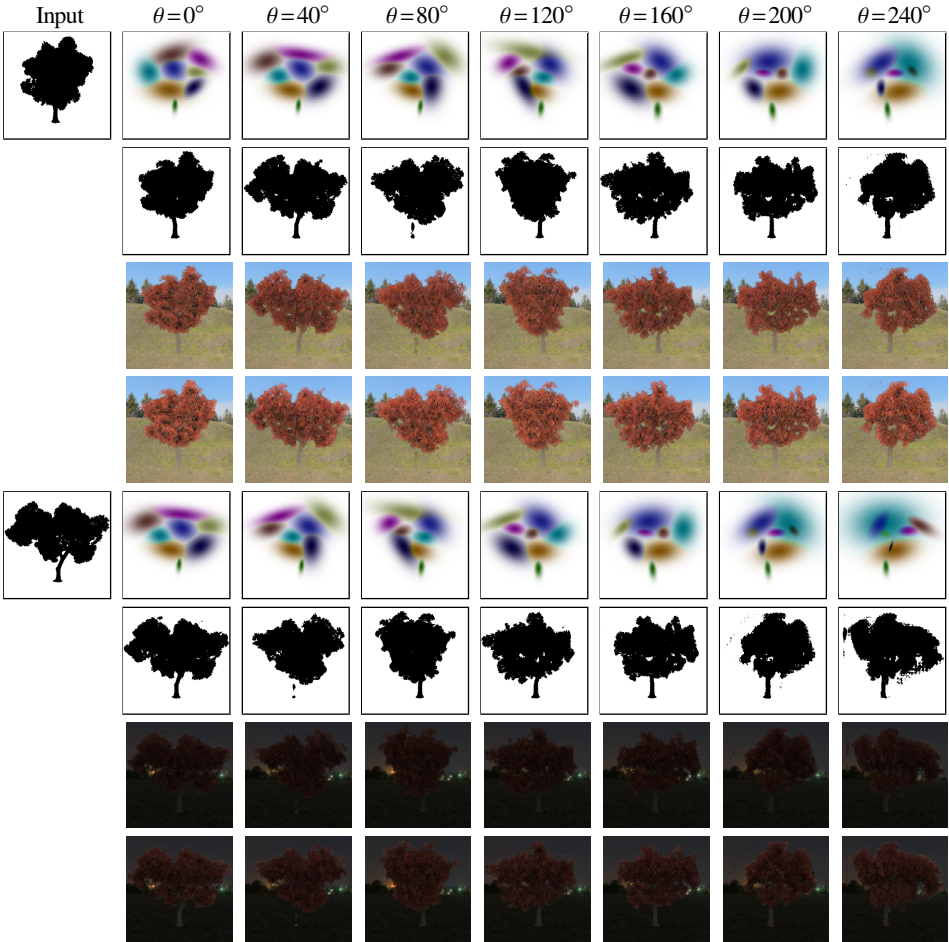


Figure 11: Additional results for Maple, with two randomly sampled latent vectors for texture.



Figure 12: Additional results for Carla, with two randomly sampled latent vectors for texture.



Figure 13: Additional results for Airplane, with two randomly sampled latent vectors for texture.



Figure 14: Additional results for Pegasus, with two randomly sampled latent vectors for texture.

C Methods to estimate Gaussian covariance Σ

When estimating the covariance matrices Σ of our Gaussian proxies, we consider three approaches: the eigendecomposition approach, the Cholesky decomposition approach, and the conditional covariance method [11].

Eigendecomposition approach. Naïvely predicting the values in the Gaussian covariance matrices Σ_k as free parameters does not satisfy the positive definiteness requirements for a covariance matrix. Instead, we leverage the eigendecomposition of $\Sigma = \mathbf{V}\mathbf{U}\mathbf{V}^T$, where \mathbf{U} is the diagonal matrix of eigenvalues with strictly positive values on the diagonal, and \mathbf{V} is an orthogonal matrix formed by the eigenvectors of Σ . We use a fully connected network to predict the diagonal values in \mathbf{U} . To ensure that they are positive, we use a sigmoid activation at the final layer, and also add a small $\epsilon=0.01$ for strict positiveness. Similarly, we predict the columns of \mathbf{V} using a fully connected network. In this case, we want \mathbf{V} to be orthonormal. As such, we adopt the following process: First, we predict two vectors \mathbf{v}_1 and \mathbf{v}'_2 and obtain \mathbf{v}_2 as the cross product of \mathbf{v}_1 and \mathbf{v}'_2 . Then, the third vector \mathbf{v}_3 is obtained as the cross product of \mathbf{v}_1 and \mathbf{v}_2 . Finally, the i -th column of \mathbf{V} is obtained by normalizing \mathbf{v}_i .

In addition, learning covariances with 32-bit float types caused issues; 64-bit double produced more stable training.

Cholesky decomposition approach. The Cholesky decomposition approach enforces positive semi-definiteness via predicting $\Sigma^{\frac{1}{2}}$ such that $\Sigma = (\Sigma^{\frac{1}{2}})^T \Sigma^{\frac{1}{2}}$. However, in our Gaussian proxies, the individual elements σ of Σ have intended meaning as the 3D scale of object parts, but $\Sigma^{\frac{1}{2}}$ does not provide an intuitive control over those σ values.

Conditional covariance approach. In this approach [11], we describe the covariance matrix as:

$$\Sigma_i = \begin{bmatrix} \sigma_1^2 & c_{12}\sigma_1\sigma_2 & c_{13}\sigma_1\sigma_3 \\ c_{12}\sigma_1\sigma_2 & \sigma_2^2 & c_{23}\sigma_2\sigma_3 \\ c_{13}\sigma_1\sigma_3 & c_{23}\sigma_2\sigma_3 & \sigma_3^2 \end{bmatrix}, \quad (1)$$

where σ_i are the individual standard deviations and c_{ij} are the correlations between variables indexed by i and j . Given these correlations, our network only needs to predict six variables: $\sigma_1, \sigma_2, \sigma_3, c_{12}, c_{13}, c_{23}$. We predict all variables directly, apart from c_{23} , which is predicted as a combination of c_{12}, c_{13} :

$$c_{23} = c_{12}c_{13} + c_{23|1} \sqrt{(1-c_{12})^2(1-c_{13})^2}, \quad (2)$$

where $c_{23|1}$ is the *partial correlation* [11]. Our network predicts $c_{23|1}$ instead of c_{23} and uses it to compute c_{23} . This approach ensures that the resulting covariance matrix is positive definite, and similarly to the eigenvalue decomposition, results on a stable training while allowing us to directly impose bounds on individual σ_i .

To evaluate these three methods, we form a test scenario where we try to optimize the Gaussian parameters for each method with the goal to fit 10 randomly selected Giraffe silhouettes; that is, we minimize the density loss \mathcal{L}_g .

Figure C shows the averaged loss values for the three methods. We see that the conditional correlation based method is the fastest to converge, followed by our eigenvalue decomposition method. Although after few iterations there is no difference between both methods. We see that the Cholesky based method fails to minimize the reconstruction objective in the same way as other methods. As such, we chose the eigenvalue based method due to its better objective minimum and its intuitive interpretation: eigenvectors represent the direction of the Gaussians while eigenvalues represent the amplitude along each direction, and this maps well to the scale and rotation of each ‘part’ in the reconstruction.

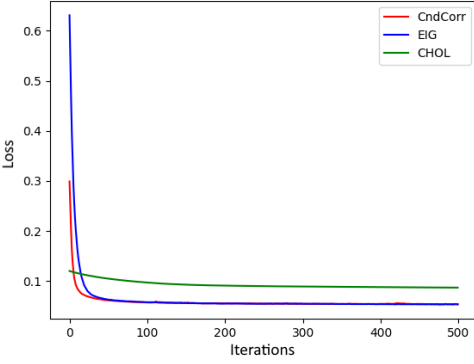


Figure 15: Average \mathcal{L}_g function of iterations computed for three covariance estimation methods. While Cndcorr: conditional correlation, and EIG: eigenvalue decomposition methods reach the same minima, CHOL: Cholesky based methods struggle due to the lack of precision propagated in the 2D projection phase. Our eigenvalue decomposition method is only slightly slower than the conditional covariance method, but it reaches the same minima and offers more intuitive control of the resulting Gaussians.

D Details of mask texturing

This section is significantly extended from the main paper, explaining the losses for mask texturing and providing details on the rationale for these losses.

Given a database of RGB images $\mathbf{i} \in \mathcal{I}$ and corresponding binary masks $\mathbf{m} \in \mathcal{M}$, we wish to learn a generative model of texture inside the mask area conditioned on the background. For this, we compute the masked image $\mathbf{i}_b = \mathbf{i} \odot (\mathbf{1} - \mathbf{m})$ containing only background pixels, and the foreground image $\mathbf{i}_f = \mathbf{i} \odot \mathbf{m}$, where \odot is the element-wise product. We use an appearance encoder E_i to extract a latent representation $\mathbf{z}_i \in \mathbb{R}^8$ for the foreground texture: $\mathbf{z}_i = E_i(\mathbf{i}_f)$. The goal of \mathbf{z}_i is to supervise the texture synthesis via a separate ‘latent reconstruction’ loss (that we will introduce later), and to let us sample \mathbf{z}_i at test time to provide control over foreground generation.

Next, we feed the background image \mathbf{i}_b through a U-Net like architecture with residual blocks separating the encoding and decoding part. We name this network G_i . Through tiling, we concatenate \mathbf{z}_i layer-wise in the encoding phase of the U-Net. This helps to ensure a strong conditioning on the appearance. As an additional textural hint, we concatenate the Gaussian maps \mathbf{g} obtained from \mathbf{m} in the decoding stage of the U-net. This conditioning is also applied layer-wise in the same way G_m is conditioned.

The final composited image \mathbf{i}' is thus obtained by compositing the output of G_i with the original background: $\mathbf{i}' = G_i(\mathbf{i}_b, \mathbf{z}_i, \mathbf{g}) \odot \mathbf{m} + \mathbf{i}_b$.

D.1 Losses

We encourage our network to learn texture using multiple losses, with overall energy to minimize given by:

$$\begin{aligned} \mathcal{L}'(G_i, E_i, D_i, \mathbf{m}) = & \beta_1 \mathcal{L}_{Rec}^i + \beta_2 \mathcal{L}_p^i + \beta_3 \mathcal{L}_{KL}^i + \beta_4 \mathcal{L}_{Adv}^i \\ & + \beta_5 \mathcal{L}_{FM}^i + \beta_6 \mathcal{L}_{zRec}^i. \end{aligned} \quad (3)$$

For hyper-parameters, we fix $\beta_1 = 100$ $\beta_2 = 0.5$ $\beta_3 = 0.01$ $\beta_4 = 1$ $\beta_5 = 10$ $\beta_6 = 0.1$.

Reconstruction loss. We encourage the synthesized image \mathbf{i}' to be an identity of the input image \mathbf{i} . We use the L_1 loss: $\mathcal{L}_{Rec}^i(\mathbf{i}, \mathbf{i}') = \|\mathbf{i} - \mathbf{i}'\|_1$.

Perceptual loss. We encourage the final image to have fine-grained details by using a perceptual loss. Following Johnson *et al.* [24] we use a VGG16 network, and extract features from its second convolutional block (‘conv2_2’), and encourage the real and generated features from \mathbf{i} and \mathbf{i}' to be similar; that is $\mathcal{L}_p^i(\mathbf{i}, \mathbf{i}') = \|\phi_2(\mathbf{i}) - \phi_2(\mathbf{i}')\|_1$, where ϕ_2 correspond to the conv2_2 feature activations.

KL loss. To allow texture sampling of the foreground at test time, we need \mathbf{z}_i to have a constrained and known structure. Inspired from VAEs, we hence obtain \mathbf{z}_i by predicting its mean and variance vectors, then we sample it using the re-parametrization trick [9]. Using the aforementioned statistics, we enforce the latent vector to be sampled from a standard normal distribution, using the KL divergence loss.

Latent reconstruction loss. Even though the KL loss insures a constrained latent representation, which allows sampling from the same space encoded by E_i . It does not insure that the generator G_i decodes different \mathbf{z}_i into diverse images. For example, G_i can output the same image independently of \mathbf{z}_i , and in that scenario all the losses would still be minimized. To avoid such a scenario, we add a novel encoder E'_i , that reconstructs the latent \mathbf{z}_i from i' , and enforce a reconstruction loss via: $\mathcal{L}_{zRec}^i = \|\mathbf{z}_i - \mathbf{z}'_i\|_1$. Note that when back-propagating gradients through that loss, we update all the parameters involved in the generation process, apart from the parameters of E_i . Doing so avoids the scenario where E_i and G_i hide the information of the latent code without producing diverse images [9].

Adversarial loss. We follow the same GAN loss as for the mask generation part, but the input to the new discriminator $D_{i,m}$ is now the tuple (i, m) , fed through concatenation. Using the tuple ensures that the generated texture is sampled from the distribution of real textures, and also that it is correlated in the same way to the mask as the real texture is. As such the adversarial loss is given by:

$$\begin{aligned} \mathcal{L}_{Adv}(G_i, D_{i,m}) = & \mathbb{E}_{(i', m)} [\min(0, -D_i(i', m) - 1)] \\ & + \mathbb{E}_{(i, m)} [\min(0, D_i(i, m) - 1)]. \end{aligned} \quad (4)$$

Feature match loss. Separate from a perceptual loss, we also add a deep feature matching loss in a similar manner to the mask generation part. This helps improve sharpness by enforcing that real and generated images elicit similar deep feature responses:

$$\mathcal{L}_{FM}^i(D_i) = \mathbb{E}_{i, i', m} \left[\sum_{l=1}^L \left\| D_i^{(l)}(i', m) - \bar{D}_i^{(l)}(i, m) \right\|_2^2 \right], \quad (5)$$

where L is the number of feature layers within the network.

E Architecture details

Our network has multiple neural network based components. All the discriminators are multiscale with a depth of 3 and share the same architecture. The only variable that can change is the number of channels in the input/output layers. Please see Appendices E to E for details on the architectures of our individual components.

Mask generation. Appendices E to E show the architecture of the different components for the mask generation part.

Texture generation. For texture we use the same discriminator architectures used for the mask. The foreground encoder E_i shares the same architecture as Table E, but predicts mean and log-covariances that are used for sampling the texture latent using the (re)-parametrization trick, and that are also used for the KL divergence loss. Similarly, the encoder E'_i that reconstructs \mathbf{z}_i from i' also shares the same architecture as described in Table E. The texture generation network is based on a U-Net like architecture described in Table E.

Table 3: Architecture for the canonical prediction network \mathcal{G}^c . FC refers to a fully connected layer. CONST refers to the input learnable constant. K is the number of Gaussians. Blue rows correspond to the prediction of the mean vector, orange rows correspond to the prediction of the covariance matrix, and non-colored rows are shared between both. Final feature activations are re-scaled into the intervals stated in Section 3 of the main paper.

Layer	#neurons	Act.
CONST.	256	-
FC.	256	LReLU
FC.	256	LReLU
FC.	256	LReLU
FC.	256	LReLU
FC (μ_k^c).	$K*3$	Tanh
FC (v_1).	$K*3$	Sigmoid
FC (v_2').	$K*3$	Sigmoid
FC (U).	$K*3$	Sigmoid

Table 4: Architecture for the encoder E_m . ‘Conv.’ is convolutional layer; ‘Res.’ is residual block; ‘InstNorm’ is instance normalization; ‘Act.’ is activation function. ‘LReLU’ denotes Leaky ReLU with a factor of 0.2.

Layer	#Filters	Size	Stride	InstNorm	Act.
Conv.	64	7×7	1	✓	LReLU
Conv.	64	3×3	2	✓	LReLU
Conv.	128	3×3	1	✓	LReLU
Conv.	128	3×3	2	✓	LReLU
Conv.	128	3×3	1	✓	LReLU
Conv.	128	3×3	2	✓	LReLU
Conv.	512	3×3	2	✓	LReLU
MaxPool.	-	-	-	-	-
FC.	8	-	-	-	NA

Table 5: Architecture for per-instance transforms prediction. FC refers to a fully connected layer. K is the number of Gaussians. Colored rows correspond to heads for specific transforms, while uncolored rows represent the shared part of the network. Final feature activations are re-scaled to be in the intervals discussed in Section 3.2 of the main paper.

Layer	#neurons	Act.
FC.	256	LReLU
FC.	256	LReLU
FC.	256	LReLU
FC.	256	LReLU
FC.	256	LReLU
FC.	256	LReLU
FC. (t)	$K*3$	Tanh
FC.	256	LReLU
FC.	256	LReLU
FC. (s)	$K*3$	Sigmoid
FC.	256	LReLU
FC.	256	LReLU
FC. (θ)	$K*3$	Tanh
FC.	256	LReLU
FC.	256	LReLU
FC. (T_0)	1	Tanh

Table 6: Architecture for the generator G_m . ‘T-conv.’ is a transposed convolutional layer; ‘InstNorm’ is instance normalization; ‘Act.’ is activation function. ‘LReLU’ denotes Leaky ReLU with a factor of 0.2.

Layer	#Filters	Size	Stride	InstNorm	Act.
T-conv.	256	3×3	1	✓	LReLU
T-conv.	256	3×3	2	✓	LReLU
T-conv.	128	3×3	1	✓	LReLU
T-conv.	128	3×3	2	✓	LReLU
T-conv.	64	3×3	1	✓	LReLU
T-conv.	64	3×3	2	✓	LReLU
Conv.	1	3×3	1	✓	Tanh

Table 7: Architecture of the discriminators D_m and D_i . ‘LReLU’ denotes Leaky ReLU with a factor of 0.2.

Layer	#Filters	Size	Stride	InstNorm	Act.
Conv.	64	4×4	2	-	LReLU
Conv.	128	4×4	2	✓	LReLU
Conv.	256	4×4	2	✓	LReLU
Conv.	512	4×4	1	✓	LReLU
Conv.	1	4×4	1	-	Ident

Table 8: Architecture for the texture generator G_i . ‘Conv.’ is convolutional layer; ‘Res.’ is residual block; ‘InstNorm’ is instance normalization; ‘Act.’ is activation function. ‘LReLU’ denotes Leaky ReLU with a factor of 0.2.

Layer	#Filters	Size	Stride	InstNorm	Act.
Conv.	64	7×7	1	✓	LReLU
Conv.	128	3×3	2	✓	LReLU
Conv.	256	3×3	2	✓	LReLU
Conv.	512	3×3	2	✓	LReLU
Res.	256	3×3	1	✓	LReLU
Res.	256	3×3	1	✓	LReLU
Res.	256	3×3	1	✓	LReLU
Res.	256	3×3	1	✓	LReLU
Res.	256	3×3	1	✓	LReLU
Res.	256	3×3	1	✓	LReLU
Res.	256	3×3	1	✓	LReLU
Res.	256	3×3	1	✓	LReLU
Res.	256	3×3	1	✓	LReLU
Deconv.	256	3×3	2	✓	LReLU
Deconv.	128	3×3	2	✓	LReLU
Deconv.	64	3×3	2	✓	LReLU
Conv.	3	7×7	1	✓	LReLU

F Analytic projection of anisotropic 3D Gaussians to 2D Gaussians

Here, for reference and completeness, we reproduce the projection function π from the supplemental material of Sridhar et al. [8] within our setting. In the main paper, we declare a general perspective pinhole camera with intrinsic matrix \mathbf{K} , rotation \mathbf{R} , and translation \mathbf{t} such that camera matrix \mathbf{P} is represented as $\mathbf{K}[\mathbf{R}, \mathbf{t}]$. We also declare K of unnormalized anisotropic 3D Gaussians $\{\mathcal{G}_k\}_{k=1}^K$. Each Gaussian \mathcal{G}_k has mean vector $\mu_k \in \mathbb{R}^3$ and covariance matrix $\Sigma_k \in \mathbb{R}^{3 \times 3}$.

The extrinsic camera parameters are the orientation \mathbf{R}_ϕ of the camera at position \mathbf{o} . They transform each Gaussian (μ, Σ) to the camera coordinate system by

$$\Sigma_{\mathbf{o}} = \mathbf{R}_\phi \Sigma_k \mathbf{R}_\phi^\top, \quad (6)$$

$$\mu_{\mathbf{o}} = \mathbf{R}_\phi (\mu_k - \mathbf{o}), \quad (7)$$

where the camera is looking down the positive z axis. In our case, the camera's position when $\mathbf{R}_\phi = \mathbf{I}$ is at $z=2$, with the object scaled in size to approximately fill the vertical view of the frame under \mathbf{K} with angle of view equal to 90° .

Sridhar et al. form a mathematical expression for the cone formed by rays drawn from \mathbf{o} that are tangent to the anisotropic Gaussian. All points on this cone satisfy

$$\mathbf{x}^\top \mathbf{M} \mathbf{x} = 0, \quad (8)$$

where the cone matrix \mathbf{M} is

$$\mathbf{M} = \Sigma_{\mathbf{o}}^{-1} (\mu_{\mathbf{o}} - \mathbf{o}) \mu_{\mathbf{o}}^\top \Sigma_{\mathbf{o}}^{-1} - (\mu_{\mathbf{o}}^\top \Sigma_{\mathbf{o}}^{-1} \mu_{\mathbf{o}} - 1) \Sigma_{\mathbf{o}}^{-1}. \quad (9)$$

Points that form a projected ellipsoid on the canonical¹ image plane at $z=1$ must also satisfy Eq. 8. Sridhar et al. derive an expression for this intersection, based on the matrix form of the second-degree polynomial representation of a conic section

$$px^2 + qxy + ry^2 + sx + ty + u = 0; \quad (10)$$

where $\mathbf{x} = [x; y; 1]^\top$. This is equivalent to Eq. 8 with \mathbf{M} as

$$\mathbf{M} = \begin{bmatrix} p & q/2 & s/2 \\ q/2 & r & t/2 \\ s/2 & t/2 & u \end{bmatrix}. \quad (11)$$

Let \mathbf{M}_{ij} denote the 2×2 submatrix excluding the i th row and j th column. The canonical parameters of the projected ellipse are given by

$$\tilde{\mu}^\pi = \frac{1}{4pr - q^2} \begin{bmatrix} qt - 2rs \\ sq - 2pt \end{bmatrix} = \frac{1}{|\mathbf{M}_{33}|} \begin{bmatrix} |\mathbf{M}_{31}| \\ -|\mathbf{M}_{23}| \end{bmatrix}, \quad (12)$$

$$\tilde{\Sigma}^\pi = -\frac{|\mathbf{M}|}{|\mathbf{M}_{33}|} \mathbf{M}_{33}^{-1} \quad (13)$$

For a general camera with intrinsic matrix \mathbf{K} , the projected ellipse (μ^π, Σ^π) from the canonical image plane is transformed to a general image plane. The transformed ellipse parameters are

$$\mu^\pi = \mathbf{K}_{33} \tilde{\mu}^\pi + [k_{13}, k_{23}]^\top \quad (14)$$

$$\Sigma^\pi = \mathbf{K}_{33} \tilde{\Sigma}^\pi \mathbf{K}_{33}^\top, \quad (15)$$

where k_{ij} here is the entry (i, j) within the \mathbf{K} matrix. These equations form our 3D space and projection model.

¹Unrelated to the canonical object model.

F.1 Camera discussion

A perspective camera was important to induce a consistent 3D space. As focal length dominates \mathbf{K} , and as it induces ‘zooming,’ one might think that perspective effects could be handled by rescaling and centering all images and masks. This may work for simpler ‘sphere-like’ objects or orthographic data. However, our object shape complexity, such as the long and angled neck of the giraffe, induces perspective variation under rotation, and a simpler camera model failed to learn a smooth 3D camera and object space.

References

- [1] Kunihiro Baba, Ritei Shibata, and Masaaki Sibuya. Partial correlation and conditional correlation as measures of conditional independence. *Australian & New Zealand Journal of Statistics*, 46(4):657–664, 2004.
- [2] Philipp Henzler, Niloy J Mitra, and Tobias Ritschel. Escaping Plato’s cave: 3D shape from adversarial rendering. In *ICCV*, pages 9984–9993, 2019.
- [3] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016.
- [4] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *ICLR*, 2014.
- [5] Yiyi Liao, Katja Schwarz, Lars Mescheder, and Andreas Geiger. Towards unsupervised learning of generative models for 3D controllable image synthesis. In *CVPR*, 2020.
- [6] Dominik Lorenz, Leonard Bereska, Timo Milbich, and Bjorn Ommer. Unsupervised part-based disentangling of object shape and appearance. In *CVPR*, pages 10955–10964, 2019.
- [7] Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. HoloGAN: Unsupervised learning of 3D representations from natural images. In *ICCV*, pages 7588–7597, 2019.
- [8] Srinath Sridhar, Helge Rhodin, Hans-Peter Seidel, Antti Oulasvirta, and Christian Theobalt. Real-time hand tracking using a sum of anisotropic gaussians model. In *3DV*, pages 319–326, 2014.
- [9] Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A. Efros, Oliver Wang, and Eli Shechtman. Toward multimodal image-to-image translation. In *NeurIPS*, 2017.