BROWN

Interactive Image Synthesis Using a Latent 3D Gaussian Model

by

Isa Milefchik

A Thesis submitted in partial fulfillment of the requirements for Honors
in the Department of Computer Science at Brown University

Providence, Rhode Island
April 2021

Acknowledgements

The work presented in this thesis is the result of a collaborative effort. Specifically, sections 2 through 4.1 are the product of work done by Youssef Mejjati, Aaron Gokaslan, Oliver Wang, Kwang In Kim, James Tompkin, and myself, while sections 4.2, 4.3, and 5 are solely my own contribution. The method described in chapter 3 was chiefly engineered by Youssef Mejjati, whom I am most grateful to have been able to work with and learn from. This thesis would, of course, not have been possible without the tremendous mentorship and guidance of professor James Tompkin.

Contents

Chapter 1

Introduction

To achieve a method that robustly learns a reconstruction of three-dimensional objects from photos taken "in the wild," with the goal of controllable image generation, the issue of pose variability must be given special consideration. In general, we often lack supervision for all variables that exist in such data (i.e., lighting, texture, pose, etc.). To devise a method that will allow for the parametrization and manipulation of such variables, it is helpful to include a parts-based decomposition of the target object's three-dimensional structure that can be shared between different instances of the object. This intermediate structure not only assists in learning the variations in shape of these objects, but also allows for user control over pose during inference.

For our primary method and its variations, we consider a setting with only mask supervision. This is a related problem to shape from silhouette, which traditionally involves rigid objects, known camera parameters, and a voxelization strategy. Our method allows for the variation of object pose, requires no camera parameters, and produces an analytic intermediate representation rather than a discrete, voxel-based one. Recent work approaching the problem of 3D reconstruction of objects with variable pose rely on voxel prediction [6], which can conflate three-dimensional spaces across instances in the unposed camera setting, and a deep voxel-based representation [30] which provides too much freedom to recover coherent 3D spaces (Fig. 4.2).

Our choice of a 3D anisotropic Gaussian mixture as the latent representation of the object's structure provides a coarse geometric description with the ability to capture the object's range of articulated poses. These are low dimensional to infer, have an analytically-differentiable projection model under perspective cameras, are composable for parts, can represent position, scale, and rotation to model part pose transforms, and are simple to self-supervise. Even then, directly inferring the Gaussian parameters across instances is difficult. Instead, we employ a canonical 3D Gaussian set plus per-instance and per-Gaussian transformation parameters that describe camera and object pose for each instance. Through training via 2D silhouette reconstruction, our representation and losses associate object parts with Gaussians, despite not having any part-level supervision.

For evaluation, we control input variation using synthetic (rendered) data that contains varying camera pose, object pose, and illumination, and show that a low-dimensional per-instance structure

can improve 3D coherence. Using the learned Gaussians within 2D RGB generation, we show disentangling of pose, view-dependent texture, and shading variation caused by lighting differences. This lets us insert objects at arbitrary viewing angles into backgrounds with matched appearance. We also explore variations on our method which seek to further strengthen 3D coherence and better unify the Gaussian representation and generated output. Through a graphical interface that allows for direct manipulation of the individual Gaussians, we demonstrate our method's capabilities in providing expressive, intuitive control over object image generation.

Chapter 2

Related Works

Image and object generation and insertion. GANs have shown tremendous progress in learning-based whole image generation [5, 48, 50], including disentangling latent features [32, 21, 13]. Beyond whole images, research has investigated how to learn to generate and add 2D objects to a given background [44], including 2D object shape generation [19] also via bounding boxes [46], completing bounding boxes with texture [7], learning to warp 2D foregrounds [24] and insert 2D objects [31]. Instead, we represent object shape and pose by learning an explicit 3D representation that allows controllable image generation.

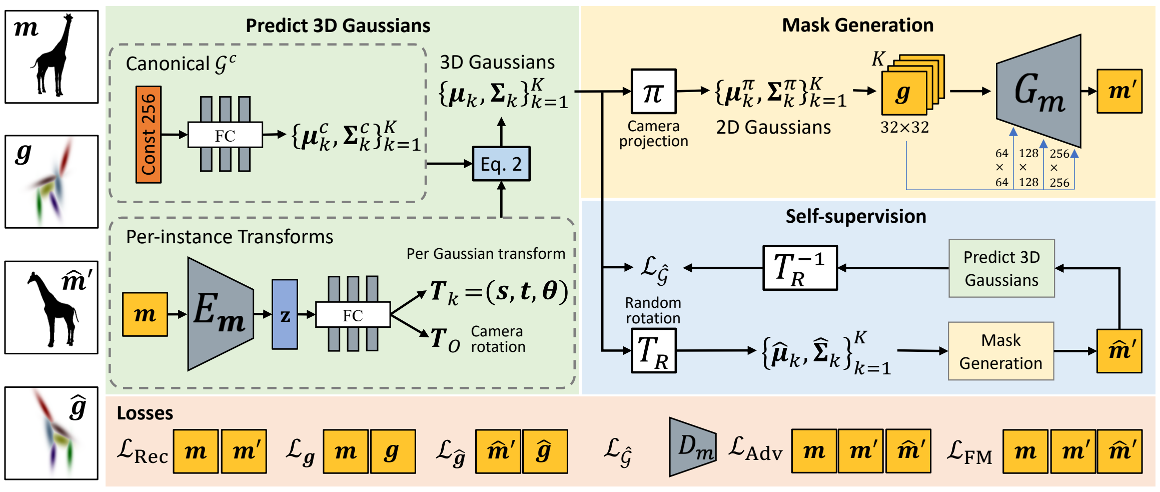
Unsupervised keypoint and part detection. Gaussians are related to keypoints and parts. Learning these is possible with supervision [25, 33, 29] and without. Here, Thewlis *et al.* [42, 41] use equivariance under 2D image transformations like warping to predict object keypoints; however, this requires the transformations to be known. To address this, Jakab *et al.* [8] learn keypoints in a self supervised way by reconstructing an object's appearance and geometry from different viewpoints. To make these intuitive, Jakab *et al.* later use a skeleton prior (*e.g.*, face, eyes, nose) to guide a discriminator [9]. This has been extended to video prediction with realistic motion [14]. Some methods use Gaussians within their pipelines. Lorenz et al. [26] predict unconstrained 2D activation maps per part for unsupervised part discovery, then estimate 2D Gaussian parameters from these to mark keypoints. Instead, we directly learn a set of 3D Gaussians to describe the shape and pose of an instance.

3D object representations. Learned representations exist for taking 3D input data like point clouds [1], volumes [36], or meshes [45, 11, 3] and generating 3D output data. These include techniques to fit sets of Gaussians to 3D shapes using 3D supervision [4], and by combining 3D supervision with multi-view silhouette losses [47]. Some works use pre-defined detailed canonical 3D meshes for 2D images [52], e.g., to learn surface parameterizations [18]. Other works learn representations from 2D input data via 3D representations, but require camera information to be given at training time [27]. For instance, DeepVoxels [38] projects RGB values on known camera rays to learn an deep voxel space that reproduces 2D inputs when projected and decoded. Other works require object-specific pose information, such as human skeletal data [17]. Without camera poses, Lei et al. build surface parametrizations for rigid 3D objects [20].

For image generation, few works take only 2D input and *no* camera or object pose information for supervision—this is hard as there is no explicit constraint on the 3D space. Liao et al. use cube and sphere mesh proxies to represent multiple simple scene objects [22]. Schwarz et al. generate radiance fields for synthetic 3D objects [36]. HoloGAN uses deep voxels within an implicit rotation space [30], and PlatonicGAN uses discrimination on random rotations to learn a generative voxel space [6]. Different geometry and appearance proxies have different trade-offs, e.g., voxels can capture shape detail but are a high dimensional space to predict; our 3D Gaussian proxy is coarse but low dimensional and can capture transformable parts.

Chapter 3

Method



Predict 3D Gaussians
Canonical $\mathcal{G}^c$
Const 256
FC
$\{\mu_k^c, \Sigma_k^c\}_{k=1}^K$
3D Gaussians
$\{\mu_k, \Sigma_k\}_{k=1}^K$
Eq. 2
Per-instance Transforms
Per Gaussian transform
E_m
z
$T_k = (s, t, \theta)$
T_O Camera rotation
Mask Generation
π
Camera projection
$\{\mu_k^\pi, \Sigma_k^\pi\}_{k=1}^K$
2D Gaussians
K
g
32×32
G_m
m'
64 × 64
128 × 128
256 × 256
Self-supervision
$\mathcal{L}_{\hat{\mathcal{G}}}$
T_R^{-1}
Predict 3D Gaussians
Random rotation
T_R
$\{\hat{\mu}_k, \hat{\Sigma}_k\}_{k=1}^K$
Mask Generation
$\widehat{m}'$
Losses
$\mathcal{L}_{\text{Rec}}$ m m'
$\mathcal{L}_g$ m g
$\mathcal{L}_{\hat{g}}$ $\widehat{m}'$ $\hat{g}$
$\mathcal{L}_{\hat{\mathcal{G}}}$
D_m $\mathcal{L}_{\text{Adv}}$ m m' $\widehat{m}'$
$\mathcal{L}_{\text{FM}}$ m m' $\widehat{m}'$

Figure 3.1: Learning a K-part 3D Gaussian representation with only mask supervision $\boldsymbol{m}$. *Green:* For each instance, we predict 3D anisotropic Gaussians by combining a canonical representation with scale, rotation, and translation transforms. *Yellow:* We project these down to 2D Gaussians in an analytically-differentiable way, then sample these into K maps. $\boldsymbol{g}$ conditions network $G_{\boldsymbol{m}}$ to generate a detailed mask $\boldsymbol{m}'$ as a reconstruction of $\boldsymbol{m}$. *Blue:* To learn a meaningful and smooth 3D space, we self supervise reconstruction by forcing a random rotation of our estimated 3D Gaussians to also produce a plausible mask $\widehat{\boldsymbol{m}}'$ *and* for its 3D Gaussian prediction to be consistent after the inverse rotation. *Orange:* We penalize reconstruction losses on masks and promote realism via adversarial discrimination.

We wish to reconstruct parts-based models for objects as a set of Gaussian proxies. To accomplish this, we will use supervision only via performing the task of mask reconstruction. We train a network to predict a set of 3D anisotropic Gaussians as coarse proxies for the objects' shape and pose, where each Gaussian emerges to loosely represent one part of the object; the mean defines the position and its covariance defines the rotation and scale of the part. Prediction is trained by projecting Gaussians into a perspective camera and transforming them into a detailed mask via a GAN. In this process, we recover a canonical Gaussian representation for the object, from which specific pose and shape transforms are estimated per image instance.

Input masks and anisotropic 3D Gaussians. We start with a dataset of 256×256 binary segmentation masks $\boldsymbol{m} \in \mathcal{M}$ of an object under varying unknown camera parameters and object poses. We also require a given number K of unnormalized anisotropic 3D Gaussians $\{\mathcal{G}_k\}_{k=1}^{K}$ (Fig. 3.1). Each Gaussian $\mathcal{G}_k$ has mean vector $\boldsymbol{\mu}_k \in \mathbb{R}^3$ and covariance matrix $\boldsymbol{\Sigma}_k \in \mathbb{R}^{3\times3}$ with its density declared as:

$$\mathcal{G}_k(\mathbf{x}) = \exp\left(-(\mathbf{x}-\boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)\right). \tag{3.1}$$

Camera. 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 $\mathbf{K}[\mathbf{R}, \mathbf{t}]$. To project a 3D anisotropic Gaussian into our camera's image plane to produce a 2D anisotropic Gaussian, we use analytically-differentiable projection function π [39]. This is valid for perspective cameras, unlike orthographic [6] or para-perspective [47] projection models that are less applicable to real-world cameras. In our experiments, $\mathbf{K}$ is fixed across instances and approximately matches that in the data.

Canonical Gaussians. Given a 256-dimensional constant [12] as input, we use a fully connected network $E_{\mathcal{G}^c}$ to predict the canonical 3D Gaussians $\mathcal{G}_k^c$ each parameterized by a mean and covariance $(\boldsymbol{\mu}^c, \boldsymbol{\Sigma}^c)$ (Fig. 3.1, green, top).

Per-instance Gaussian transforms. Given an input mask $\boldsymbol{m}$, we extract a latent vector representing pose $\mathbf{z} \in \mathbb{R}^8$ via a convolutional encoder network $E_{\boldsymbol{m}}$. Then, from $\mathbf{z}$, we use a fully connected network to predict two transformations: 1) A camera transformation $\mathbf{T}_O$ that moves the camera with respect to the canonical model; in our experiments, we mainly consider a yaw rotation $\mathbf{R}_\phi$. 2) K Gaussian local transformations $\mathbf{T}_k$ consisting of scale, translation, and rotation $(\mathbf{s}_k, \mathbf{t}_k, \boldsymbol{\theta}_k)$ with each in $\mathbb{R}^3$ (Fig. 3.1, green, bottom).

Given the canonical parameters $(\boldsymbol{\mu}_k^c, \boldsymbol{\Sigma}_k^c)$, we obtain the per-instance Gaussians $\mathcal{G}_k$ with parameters $(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ via:

$$\begin{aligned} \boldsymbol{\mu}_k &= \mathbf{R}_\phi(\boldsymbol{\mu}_k^c + \mathbf{t}_k) \\ \boldsymbol{\Sigma}_k &= (\mathbf{R}_\phi \mathbf{R}_{\theta_k} \mathbf{U}_k \mathbf{s}_k \mathbf{S}_k)(\mathbf{R}_\phi \mathbf{R}_{\theta_k} \mathbf{U}_k \mathbf{s}_k \mathbf{S}_k)^\top, \end{aligned} \tag{3.2}$$

where $\mathbf{R}_{\theta_k}$ is the rotation matrix form of $\boldsymbol{\theta}_k$, and $\mathbf{S}_k$ and $\mathbf{U}_k$ are obtained via eigenvalue decomposition of $\boldsymbol{\Sigma}_k^c$: $\boldsymbol{\Sigma}_k^c = (\mathbf{U}_k\mathbf{S}_k)(\mathbf{U}_k\mathbf{S}_k)^\top$. $\mathbf{S}_k$ is a diagonal matrix. The square of its (j, j)-th entry represents the j-th eigenvalue of $\boldsymbol{\Sigma}_k$. This allows us to control the scale and rotation of each individual Gaussian via the matrices $\mathbf{U}_k$ and $\mathbf{S}_k$.

Estimating Σ. Naïvely predicting the values in the Gaussian covariance matrices $\boldsymbol{\Sigma}_k$ as free parameters does not satisfy the positive definiteness requirements for a covariance matrix. Instead, we leverage the eigendecomposition of $\boldsymbol{\Sigma} = \mathbf{V}\mathbf{U}\mathbf{V}^\top$, 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 $\boldsymbol{\Sigma}$. 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.

Conditional mask synthesis. Even a large number of Gaussian proxies will not reconstruct fine mask detail. As such, we use a conditional mask generator $G_{\boldsymbol{m}}$ to add back the detail using up-sampling transposed convolutions (Fig. 3.1, yellow). Given the 3D Gaussians for an instance, we project them to 2D Gaussians on the image plane of our camera: $\pi(\mathcal{G}_k) = (\boldsymbol{\mu}_k^\pi, \boldsymbol{\Sigma}_k^\pi)$. Then, using the 2D version of Eq. 3.1, we sample the density of each projected Gaussian on a raster grid to create K Gaussian maps $\{\boldsymbol{g}_k\}_{k=1}^K$. These are input to $G_{\boldsymbol{m}}$ to condition the synthesis of predicted mask $\boldsymbol{m}'$, which is the learned reconstruction of $\boldsymbol{m}$. We enforce a stronger effect in $G_{\boldsymbol{m}}$ by using layer-wise conditioning via Gaussian maps at $32^2, 64^2, 128^2,$ and 256^2 resolutions.

3.1 Losses

We encourage our network to reconstruct an object using multiple losses, with overall energy to minimize given by:

$$\mathcal{L}(E_{\mathcal{G}^c}, E_{\boldsymbol{m}}, G_{\boldsymbol{m}}, D_{\boldsymbol{m}}) = \lambda_1 \mathcal{L}_{\text{Rec}} + \lambda_2 \mathcal{L}_{\boldsymbol{g}} + \lambda_3 \mathcal{L}_{\widehat{\mathcal{G}}} + \lambda_4 \mathcal{L}_{\widehat{\boldsymbol{g}}} + \lambda_5 \mathcal{L}_{\text{Adv}} + \lambda_6 \mathcal{L}_{\text{FM}} \tag{3.3}$$

Reconstruction loss. We encourage synthesized mask $\boldsymbol{m}'$ to reconstruct input instance mask $\boldsymbol{m}$ with an L_1 loss: $\mathcal{L}_{\text{Rec}}(\boldsymbol{m}, \boldsymbol{m}') = \|\boldsymbol{m} - \boldsymbol{m}'\|_1$.

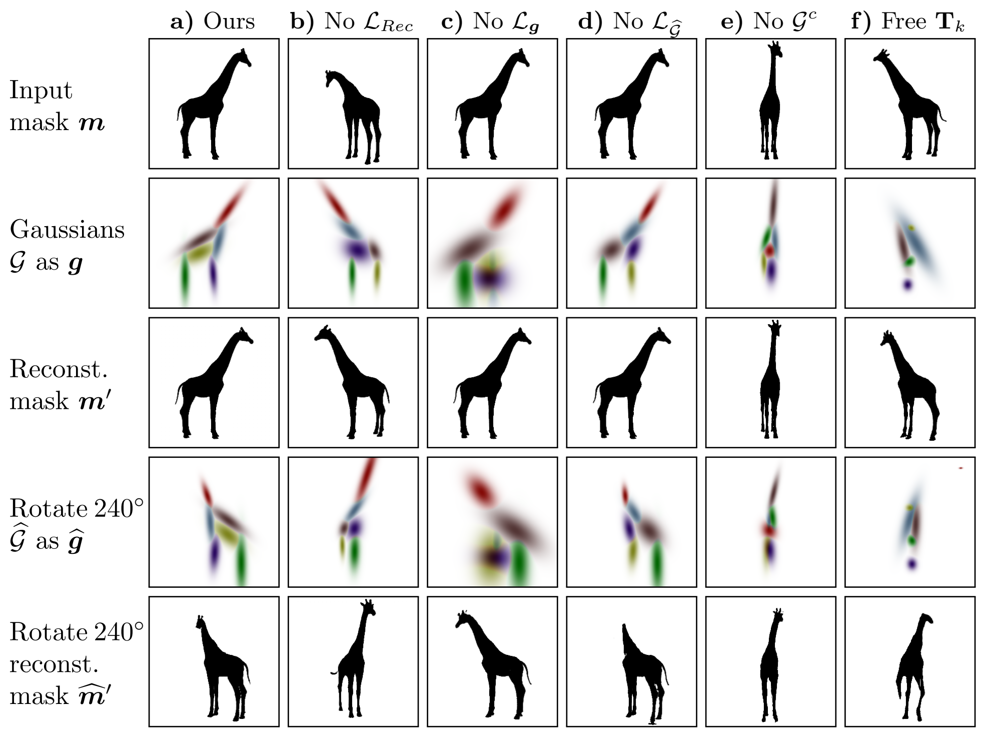
Density loss. Even though they cannot represent fine detail in $\boldsymbol{m}$, we still wish for all projected Gaussians to 1) cover regions of the mask without overlap, and 2) cover as much of the mask as possible. We encourage this via:

$$\mathcal{L}_{\boldsymbol{g}}(\boldsymbol{m}, \boldsymbol{g}) = \left\|\boldsymbol{m} - \Sigma_{k=1}^K \boldsymbol{g}_k\right\|_1. \tag{3.4}$$

The sum over sampled 2D Gaussians is equivalent to a grayscale version of the colored parts visualization in Figure 3.2. Here, both inputs are in the range $[0, 1]$, and we take $\boldsymbol{g}$ at our mask resolution of 256×256.

Self-supervised transform mask loss. We wish for the 3D space expressed through our recovered object Gaussians and camera transform parameters in $\mathbf{T}_O$ to be consistent across varying camera views even though we only have mask supervision. Thus, we randomly sample a 3D transformation $\mathbf{T}_R$, again mainly as a yaw rotation, and apply it via Eq. 3.2 to produce rotated 3D Gaussians $\widehat{\mathcal{G}} = (\widehat{\boldsymbol{\mu}}, \widehat{\boldsymbol{\Sigma}})$. As before, these are then projected via π to 2D parameters $(\widehat{\boldsymbol{\mu}}^\pi, \widehat{\boldsymbol{\Sigma}}^\pi)$, then sampled into 2D maps $\widehat{\boldsymbol{g}}$, and finally via $G_{\boldsymbol{m}}$ to generate a mask $\widehat{\boldsymbol{m}}'$ (Fig. 3.1, blue).

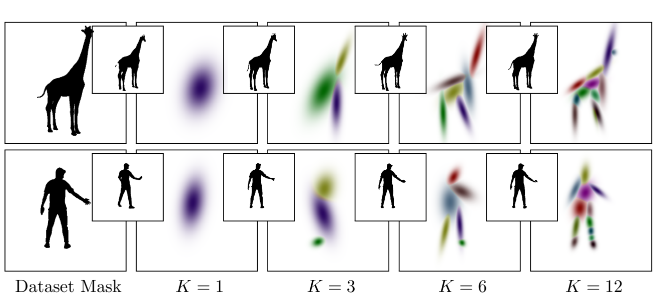
As $\widehat{\boldsymbol{m}}'$ does not correspond to a known input instance, we cannot directly enforce $\mathcal{L}_{\text{Rec}}$. Instead, we encourages the projected novel view Gaussians $\widehat{\boldsymbol{g}}$ to be consistent with the synthesized novel view $\widehat{\boldsymbol{m}}'$ via a second density loss: $\mathcal{L}_{\widehat{\boldsymbol{g}}}(\widehat{\boldsymbol{m}}', \widehat{\boldsymbol{g}}) = \|\widehat{\boldsymbol{m}}' - \sum_{k=1}^K \widehat{\boldsymbol{g}}_k\|_1$. Without this loss, $\boldsymbol{g}$ can describe well the input mask $\boldsymbol{m}$, but the rotated $\widehat{\boldsymbol{g}}$ may not describe well the generated mask $\widehat{\boldsymbol{m}}'$.


a) Ours
b) No $\mathcal{L}_{Rec}$
c) No $\mathcal{L}_{\boldsymbol{g}}$
d) No $\mathcal{L}_{\widehat{\mathcal{G}}}$
e) No $\mathcal{G}^c$
f) Free $\mathbf{T}_k$
Input mask $\boldsymbol{m}$
Gaussians $\mathcal{G}$ as $\boldsymbol{g}$
Reconst. mask $\boldsymbol{m}'$
Rotate 240° $\widehat{\mathcal{G}}$ as $\widehat{\boldsymbol{g}}$
Rotate 240° reconst. mask $\widehat{\boldsymbol{m}}'$

	a) Ours	b) No $\mathcal{L}_{Rec}$	c) No $\mathcal{L}_{\boldsymbol{g}}$	d) No $\mathcal{L}_{\widehat{\mathcal{G}}}$	e) No $\mathcal{G}^c$	f) Free $\mathbf{T}_k$
IoU ▲	83.96	65.09	82.98	84.75	86.62	73.16
DSSIM ▼	6.22	14.20	6.83	6.16	5.32	10.94

Figure 3.2: Ablations for *Giraffe*. Note: Input masks vary per column as certain effects are only visible at particular angles; Gaussian colors vary across columns. **(a)** Our full loss model. **(b)** Without a reconstruction loss on $\boldsymbol{m}'$, the Gaussians only approximately correspond to the input mask. **(c)** Without a density loss on $\boldsymbol{g}$, the Gaussians do not well represent the input mask, yet $G_{\boldsymbol{m}}$ still produces the correct mask from these less 'coherent' Gaussians. **(d)** Not 'closing the loop' in the self-supervised loss hurts self occlusion cases or when the 2D Gaussian layouts are not sufficient to recover 3D information. **(e)** Not using a canonical representation fails to rotate Gaussians recovered for thin front/back views. **(f)** Not reasonably bounding the per-instance transforms allows nonsense canonicals.

Table: Over the test set, mean IoU×100 and DSSIM×100 of reconstructed masks vs. ground truth masks at specific camera angles. Our qualitative results show these metrics do not tell the whole story.



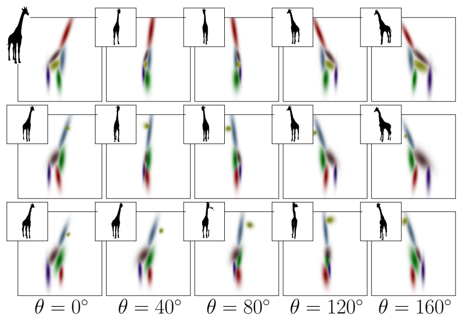


Figure 3.3: *Left:* Varying K produces levels of abstraction over the object's shape and pose and so over generation control. At low K, only the major features are represented such as legs and neck. At higher K, details like individual legs ($K = 12$, top) and leg parts (calf, thigh) appear with the 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 $K = 12$. *Right:* Randomly sampling sparser datasets still recovers the coarse 3D structure of the input object. Rows 0, 1, 2, use $\frac{1}{16}$, $\frac{1}{32}$, $\frac{1}{64}$, of images in the training set; approximately 140, 70, and 35 images respectively. Colorings are different across rows.

Self-supervised transform inverse 3D Gaussian loss. We can also pass $\boldsymbol{m}'$ back through our 3D Gaussian prediction stages (Fig. 3.1, green) to recover an estimate of the proxies under random transform $\mathbf{T}_R$. Then, we can invert this transform and penalize a loss against our initial estimate of the 3D Gaussians. With slight notation abuse: $\mathcal{L}_{\widehat{\mathcal{G}}}(\mathcal{G}, \widehat{\mathcal{G}}') = \|\mathcal{G} - \mathbf{T}_R^{-1}(\eta(\widehat{\boldsymbol{m}}'))\|_1$, where η predicts 3D Gaussians for a mask.

Adversarial loss. Training using only reconstruction losses tends to produce blurry images, so we adopt an adversarial training strategy. $G_{\boldsymbol{m}}$ attempts to generate realistic masks to fool a discriminator $D_{\boldsymbol{m}}$, while $D_{\boldsymbol{m}}$ attempts to classify generated masks separately from real training masks. Within this, we also discriminate against our self-supervised transform masks $\widehat{\boldsymbol{m}}'$: these should also fool $D_{\boldsymbol{m}}$. We use a hinge-GAN loss $\mathcal{L}_{\text{Adv}}$ for better training stability [23, 43, 28]:

$$\begin{aligned}\mathcal{L}_{\text{Adv}}(G_{\boldsymbol{m}}, D_{\boldsymbol{m}}) = \mathbb{E}_{\widehat{\boldsymbol{m}}'}[\min(0, -G_{\boldsymbol{m}}(\widehat{\boldsymbol{m}}') - 1)] + \\ 2\mathbb{E}_{\boldsymbol{m}}[\min(0, G_{\boldsymbol{m}}(\boldsymbol{m}) - 1)] + \mathbb{E}_{\boldsymbol{m}'}[\min(0, -D_{\boldsymbol{m}}(\boldsymbol{m}') - 1)].\end{aligned} \tag{3.5}$$

To reconstruct the 3D shape within a consistent world space, along with $\boldsymbol{m}$ and $\boldsymbol{m}'$, we find that it is sufficient to give the discriminator a mask $\widehat{\boldsymbol{m}}'$ generated from only one random rotation per instance (as similarly found by Henzler et al. [6]), rather than multiple random rotations.

Feature match loss. We improve sharpness by enforcing that real and generated images elicit similar deep feature responses in each layer l of the discriminator $D_M^{(l)}$ [35, 49]:

$$\mathcal{L}_{\text{FM}}(D_{\boldsymbol{m}}) = \mathbb{E}_{\boldsymbol{m},\boldsymbol{m}',\widehat{\boldsymbol{m}}'}\left[\Sigma_{l=1}^{L}\left\|D_{\boldsymbol{m}}^{(l)}(\widehat{\boldsymbol{m}}') - \bar{D}_{\boldsymbol{m}}^{(l)}(\boldsymbol{m})\right\|_2^2 + \left\|D_{\boldsymbol{m}}^{(l)}(\boldsymbol{m}') - \bar{D}_{\boldsymbol{m}}^{(l)}(\boldsymbol{m})\right\|_2^2\right], \tag{3.6}$$

where $\bar{D}_{\boldsymbol{m}}^{(l)}$ is the moving average of feature activations in layer l, and L is the number of layers.

Constraining pose and shape. We bound $\boldsymbol{\mu}_k$ to $[-1, 1]$ and the diagonal values of $\boldsymbol{\Sigma}_k^c$ to $[0.01, 0.51]$. We prevent any Gaussian from being too small/too large; this encourages learning to use all Gaussians. To remove implausible canonical $\mathcal{G}^c$, we constrain $\mathbf{T}_k = (\mathbf{s}_k, \mathbf{t}_k, \boldsymbol{\theta}_k)$ to produce per-instance $\mathcal{G}$ that remain somewhat close to $\mathcal{G}^c$ while still giving freedom to accommodate shape and pose changes (Figure 3.2). Discriminating masks generated from $\mathcal{G}^c$ is also possible, with self-supervision via random transforms, and may help relax per-instance transform constraints.

3.2 Mask texturing

3D Gaussian proxies could apply to various scenarios, e.g., as a conditioning rig to an image generation task [40]. We demonstrate object posing and inserting into an existing image (Fig. 4.1). For this, we condition a separate second GAN on the mask and Gaussians, and on a background image to let us approximately match the scene lighting.

Given a database of RGB images $\boldsymbol{i} \in \mathcal{I}$ and corresponding binary masks $\boldsymbol{m} \in \mathcal{M}$, we wish to learn a generative model of texture inside the mask conditioned on the background. First, we compute the background image $\boldsymbol{i}_b = \boldsymbol{i} \odot (\mathbf{1} - \boldsymbol{m})$ and the foreground image $\boldsymbol{i}_f = \boldsymbol{i} \odot \boldsymbol{m}$, where $\odot$ is the element-wise product. Next, we use an appearance encoder $E_{\boldsymbol{i}}$ to extract a latent representation $\mathbf{z}_{\boldsymbol{i}} \in \mathbb{R}^8$ for the foreground texture: $\mathbf{z}_{\boldsymbol{i}} = E_{\boldsymbol{i}}(\boldsymbol{i}_f)$; this lets us sample foregrounds at test time. We tile $\mathbf{z}_{\boldsymbol{i}}$ and concatenate it with the background image $\boldsymbol{i}_b$, and pass it into a U-Net-like network $G_{\boldsymbol{i}}$ to generate texture. In $G_{\boldsymbol{i}}$'s encoding phase, we layer-wise condition via $\mathbf{z}_{\boldsymbol{i}}$. In $G_{\boldsymbol{i}}$'s decoding phase, we concatenate the Gaussian maps $\boldsymbol{g}$ obtained from $\boldsymbol{m}$ and apply layer-wise conditioning as per $G_{\boldsymbol{m}}$. The final image $\boldsymbol{i}'$ is created from the output of $G_{\boldsymbol{i}}$ with the original background: $\boldsymbol{i}' = G_{\boldsymbol{i}}(\boldsymbol{i}_b, \mathbf{z}_{\boldsymbol{i}}, \boldsymbol{g}) \odot \boldsymbol{m} + \boldsymbol{i}_b$.

Losses. We train our network by minimizing an energy:

$$\mathcal{L}^{\boldsymbol{i}}(G_{\boldsymbol{i}}, E_{\boldsymbol{i}}, D_{\boldsymbol{i},\boldsymbol{m}}) = \beta_1 \mathcal{L}^{\boldsymbol{i}}_{\text{Rec}} + \beta_2 \mathcal{L}^{\boldsymbol{i}}_{p} + \beta_3 \mathcal{L}^{\boldsymbol{i}}_{\text{KL}} + \beta_4 \mathcal{L}^{\boldsymbol{i}}_{\text{Adv}} + \beta_5 \mathcal{L}^{\boldsymbol{i}}_{\text{FM}} + \beta_6 \mathcal{L}^{\boldsymbol{i}}_{\mathbf{z}\text{Rec}}. \tag{3.7}$$

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

Perceptual loss. We encourage fine-grain detail by using a VGG16 perceptual loss [10] from the second convolutional block (ϕ_2 :='conv2$_2$'): $\mathcal{L}^{\boldsymbol{i}}_{p}(\boldsymbol{i}, \boldsymbol{i}') = \|\phi_2(\boldsymbol{i}) - \phi_2(\boldsymbol{i}')\|_1$

KL loss. To structure $\mathbf{z}_{\boldsymbol{i}}$ for test-time sampling, we predict mean and variance vectors for $\mathbf{z}_{\boldsymbol{i}}$, sample one using the re-parametrization trick [15], then enforce that it comes from a Normal distribution using the KL divergence loss.

Latent reconstruction loss. The KL loss does not ensure that $G_{\boldsymbol{i}}$ decodes $\mathbf{z}_{\boldsymbol{i}}$ into diverse images. To prevent $\mathbf{z}_{\boldsymbol{i}}$ from being ignored, we add a novel encoder $E'_{\boldsymbol{i}}$ to reconstruct $\mathbf{z}_{\boldsymbol{i}}$ from $\boldsymbol{i}'$, and enforce a reconstruction loss via: $\mathcal{L}^{\boldsymbol{i}}_{\mathbf{z}\text{Rec}} = \|\mathbf{z}_{\boldsymbol{i}} - \mathbf{z}_{\boldsymbol{i}'}\|_1$. When back-propagating gradients from $\mathcal{L}^{\boldsymbol{i}}_{\mathbf{z}\text{Rec}}$, we update all generation parameters *apart* from those in $E_{\boldsymbol{i}}$. This avoids $E_{\boldsymbol{i}}$ and $G_{\boldsymbol{i}}$ hiding the latent code information without producing diverse images [51].

Adversarial losses. Finally, we also train $D_{\boldsymbol{i},\boldsymbol{m}}$ to discriminate $(\boldsymbol{i}, \boldsymbol{m})$ from $(\boldsymbol{i}', \boldsymbol{m})$ such that the RGB image and mask are correlated, and we use $D_{\boldsymbol{i},\boldsymbol{m}}$ to penalize a feature matching loss between $(\boldsymbol{i}, \boldsymbol{m})$ and $(\boldsymbol{i}', \boldsymbol{m})$.

3.3 Discussion

Importance of losses and components (Figure 3.2). Removing the reconstruction losses on $\boldsymbol{m}'$ allows a mask to only approximately correspond to the Gaussians as long as it satisfies the discriminator and $\mathcal{L}_{\boldsymbol{g}}$. Removing the density loss on $\boldsymbol{g}$ causes less 'coherent' Gaussians: they are not forced to represent the generated mask, yet $G_{\boldsymbol{m}}$ can still produces a high detail mask from these Gaussians. Finally, the transform inverse loss 'closes the loop' for the self supervision and helps maintain 3D space consistency and mask quality, especially under cases when penalizing the 2D maps $\widehat{\boldsymbol{g}}'$ alone cannot accurately predict 3D, such as when objects have strong rotation-dependent self occlusion.

Canonical $\mathcal{G}^c$ encourages a meaningful 3D space as each instance should be consistent with other instances. Directly estimating per-instance Gaussians fails for thin front/back views as the self-supervised rotation must only be consistent with $\mathcal{L}_{\boldsymbol{g}}$ and discrimination $\mathcal{L}_{\text{Adv,FM}}$ (Fig. 3.2e), instead learning a non-linear space that only rotates between front/back views. Further, estimating μ, Σ values without the const+FC layers [12] led to worse performance.

Quantitatively, we compute IoU and DSSIM between ground truth and generated masks across a range of angles (Fig. 3.2, bottom). While removing $\mathcal{L}_{\widehat{\mathcal{G}}}$ or $\mathcal{G}^c$ improve the metrics slightly, qualitatively our final model is more coherent: parts can flicker in and out without $\mathcal{L}_{\widehat{\mathcal{G}}}$, and the 3D space is less coherent without $\mathcal{G}^c$.

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. 3.3, 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. 3.3, 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-instance deformations is learned. We also shows how the Gaussians are still usefully recovered as input data decreases $64\times$ in number (Fig. 3.3, right), though with less mask detail.

Chapter 4

Experiments

4.1 Learning Gaussian proxies for shape and pose from masks

Datasets. We render RGB images and masks using path tracing with ten real-world 360° HDR lighting maps of outdoor natural environments for realistic lighting and self-shadowing. For each image, we randomly rotate the camera around the up vector at a fixed distance from the object, to match settings in the literature [30]. We use four datasets without pose variation and of increasing shape complexity (*Maple*, *Airplane*, *Carla*, and *Pegasus*), and four animated datasets with pose variation (*Bee*, *Giraffe*, *Manuel*, *Old Robot*). These include hovering and flapping wings, walking, neck bending, and dancing (each with 110-400 frames; see video). We randomly sample animation frames: poses are not matched across views or in any temporal or rotation order, and we discard object and camera poses during training. We use 1,000/2,000 images for static/animated datasets, with a random 90/10% training/test split.

Training and hyperparameters. We train mask and texture generators for 200 epochs on 2 RTX 2080 TI GPUs. We use the ADAM optimizer with a learning rate of $1e-4$, and $\beta = 0.5$. For static datasets, we predict the yaw rotation $\boldsymbol{R}_\phi$ per instance to affect the canonical Gaussians. For the mask hyper-parameters, we set $\lambda_1 = 100$, $\lambda_2 = 100$, $\lambda_3 = 100$, $\lambda_4 = 100$, $\lambda_5 = 1$, $\lambda_6 = 10$. We chose λ_1, λ_2, λ_3. λ_4 over the interval $[0, 10, 50, 100]$. For *Giraffe* with slower animation, $\lambda_3 = 10$ and $\lambda_4 = 50$ led to a slightly better Gaussians. For texture hyper-parameters, we fix $\beta_1 = 100$ $\beta_2 = 0.5$ $\beta_3 = 0.01$ $\beta_4 = 1$ $\beta_5 = 10$ $\beta_6 = 0.1$.

Baselines. To show the value of model components, we compare to HoloGAN [30], Platonic-GAN [6], and Liao et al. [22], and provide methods with *just masks* and *just RGB foregrounds*. Each uses 3D proxies to generate images. HoloGAN and PlatonicGAN use voxels: the HoloGAN bottleneck has 64-dim. deep appearance vectors in 16^3 voxels that are projected to 2D and decoded, while PlatonicGAN directly predicts a 64^3 RGBA voxel space *per instance* to handle variation. As

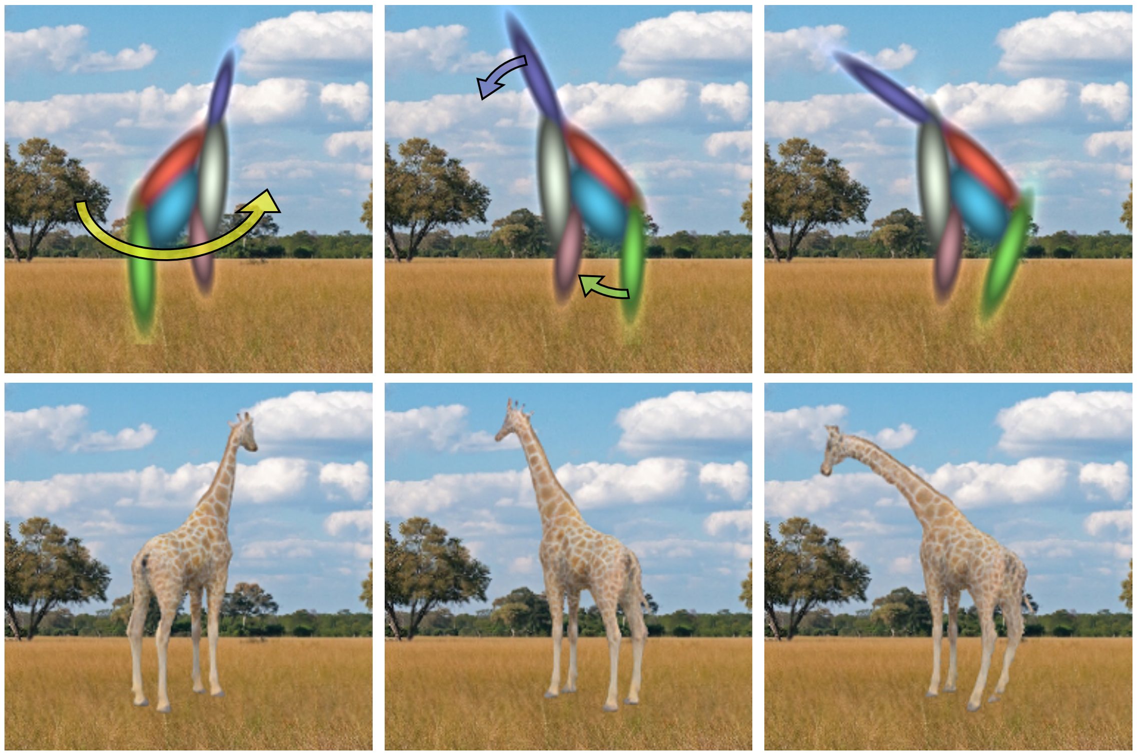


Figure 4.1: Explicitly recovering camera, shape, and pose allows interactive 3D Gaussian manipulation to generate novel instances.

a constraint, HoloGAN employs a weaker (but flexible) *latent* reconstruction loss vs. our projection and pixel-wise reconstruction loss. PlatonicGAN also uses a pixel-wise reconstruction loss and, via projection, this can be related to unposed voxel carving when given masks as inputs. Liao et al. estimate multiple 3D primitives (cubes or spheres) as proxies for objects with simple geometry.

Results—mask only. This setting compares the ability to reconstruct a 3D camera and object space. For the static datasets (Fig. 4.2), HoloGAN's deep voxels reconstruct the input masks, but its latent rotation space can be incoherent with masks at incorrect angles. PlatonicGAN's voxel spaces are naturally 3D and with shape detail, but suffer some incorrect rotations and include spurious or missing geometry. Our approach infers plausible coarse 3D structure that controls high-quality 2D mask generation.

For the animated datasets, methods must accommodate instance pose variation, and all baselines perform worse. HoloGAN has both part errors (incorrect leg placement) and a low-coherence 3D space (rotation is not smooth). PlatonicGAN's shapes are incorrectly reconstructed, with missing or misplaced legs and spurious content: even though the method estimates per-instance shape, without a canonical model these are incorrectly corresponded in the 3D estimation task, combining parts of objects from across poses. Our method by construction has a canonical 3D $\mathcal{G}^c$ and transformable parts, producing a coherent 3D camera and coarse posable object space.

On masks	IoU×100 ▲	DSSIM×100 ▼
Ours	**81.97**	**9.35**
PlatonicGAN [6]	77.29	21.29
On RGB	**KID×100 ▼**	**FID×100 ▼**
Ours (via masks)	**9.16 ± 0.60**	**117.81**
PlatonicGAN [6]	49.7 ± 0.89	375.26
HoloGAN [30]	32.72 ± 0.87	298.35
Liao et al. [22]	34.2 ± 0.84	292.89

Table 4.1: Metrics are computed per dataset and then mean averaged.

Results—foreground only. Here, the reconstruction task is more complex, with additional texture and lighting variation. Even for static scenes, HoloGAN struggles to generate high-quality appearance, and the resulting 3D spaces for dynamic scenes mix all input variations or fail to correctly rotate the image (Fig. 4.2, *Maple*). PlatonicGAN successfully generates detail, but again these have object geometry errors and the predicted voxel coloring only approximates the intended output (Fig. 4.2, *all*). Liao et al. generated instances are of broadly good quality, though the pose is entangled with the camera rotation and texture is low resolution and less consistent. As might be expected, our approach demonstrates that using additional mask information to separate shape and appearance allows conditioning higher-fidelity 2D texture generation with disentangled 3D camera, pose, and lighting consistency.

Quantitative results. PlatonicGAN and our method infer explicit 3D spaces. As such, we compute IoU and DSSIM on masks at a known camera angle and compare to test-set ground truth masks: if a method forms a coherent 3D camera and object space, then masks will match (Tab. 4.1). For methods that infer implicit 3D spaces (without meaningful angles), we compute KID and FID on generated RGB foregrounds (KID/FID are pre-trained for RGB via ImageNet).

Real-world data. We show the benefits of a mid-level 3D structure via the managed control of variation available in synthetic data. Many other variations exist in real-world datasets. To show this gap, we demonstrate our method on highly-varied MS COCO data (Fig. 4.3). Here, our 3D space and Gaussians are plausible, even though there is significant quality variation in the hand-drawn input masks especially for front/back views; better masks would improve this [16].

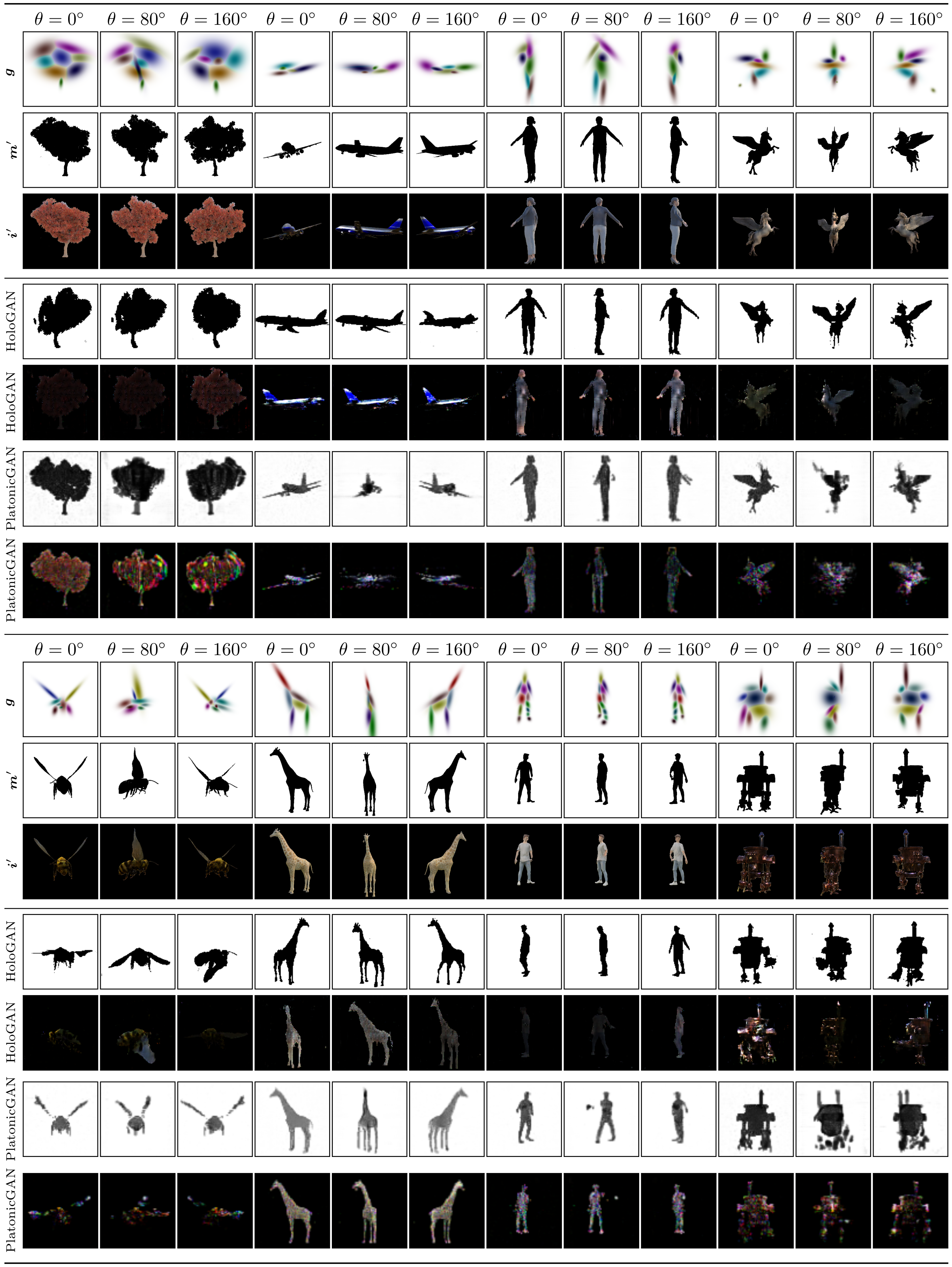


Figure 4.2: *Rows in each block:* Reconstructed Gaussians, masks, and RGB images, across three output angles and with any texture-specific latent variables fixed, with comparisons to HoloGAN [30] and PlatonicGAN [6] run on *just masks* and *just RGB foregrounds*. Note that HoloGAN only infers a latent 'angle', making mapping to an explicit 3D space not possible. *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.*

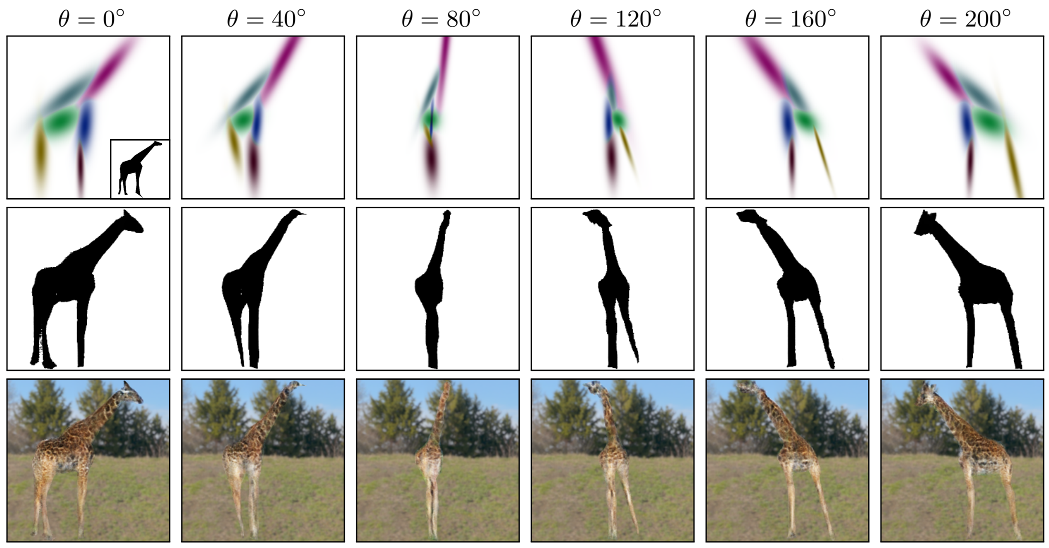


Figure 4.3: Our method can produce plausible 3D Gaussians from low-quality and highly-varied masks from MS COCO.

4.2 Learning Gaussian proxies for shape and pose from RGBA foreground

Inferring 3D Gaussians from mask images presents several issues. One, the pose of a silhouette can often appear ambiguous, as a 180° rotation of an object can produce a near-identical silhouette in some cases. Illusions such as The Spinning Dancer demonstrate the difficulty that even humans have perceiving a consistent rotation direction from the silhouette of a spinning object. Our model can often be confused by pose ambiguity of the input mask and generate correspondingly ambiguous results (Fig. 4.5). Furthermore, we discovered that while the texture generator converges faster when conditioned by the inferred Gaussian maps, these maps have little to no effect on the generated texture once finished training.

4.2.1 Method

Motivated both by the limitations of mask-only input and by the disconnect discovered between inferred Gaussians and texture generation, we designed an alternative method for generating textured object images by conditioning on RGBA foreground images (Fig. 4.4). The architecture remains largely the same as in section 3, except for the following changes:

Input and output. The network takes as input RGBA images $\boldsymbol{i}$ of object foregrounds. In the red, green, and blue channels, pixels belonging to the background are set to zero, and the alpha channel is set to the object mask. The network's generator $G_{\boldsymbol{i}}$ accordingly produces RGBA images $\boldsymbol{i'}$ of the same composition as the input.

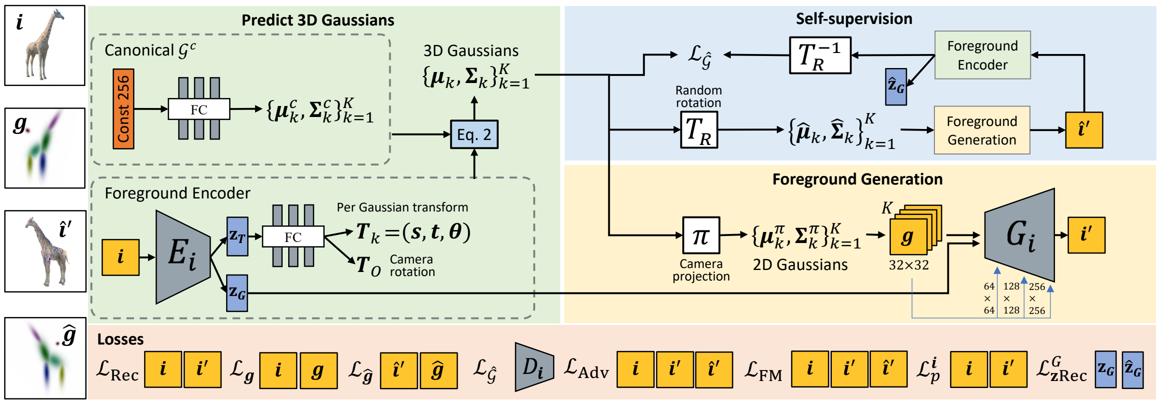

i
g
$\hat{i}'$
$\hat{g}$
Predict 3D Gaussians
Canonical $\mathcal{G}^c$
Const 256
FC
$\{\mu^c_k, \Sigma^c_k\}^K_{k=1}$
3D Gaussians
$\{\mu_k, \Sigma_k\}^K_{k=1}$
Eq. 2
Foreground Encoder
i
E_i
z_T
z_G
FC
Per Gaussian transform
$T_k = (s, t, \theta)$
T_O Camera rotation
Self-supervision
$\mathcal{L}_{\hat{\mathcal{G}}}$
T_R^{-1}
Foreground Encoder
$\hat{z}_G$
Random rotation
T_R
$\{\hat{\mu}_k, \hat{\Sigma}_k\}^K_{k=1}$
Foreground Generation
$\hat{i}'$
Foreground Generation
π
Camera projection
$\{\mu^\pi_k, \Sigma^\pi_k\}^K_{k=1}$
2D Gaussians
K
g
32×32
G_i
i'
64 × 64
128 × 128
256 × 256
Losses
$\mathcal{L}_{\text{Rec}}$ i i'
$\mathcal{L}_g$ i g
$\mathcal{L}_{\hat{g}}$ i' $\hat{g}$
$\mathcal{L}_{\hat{\mathcal{G}}}$
D_i
$\mathcal{L}_{\text{Adv}}$ i i' $\hat{i}'$
$\mathcal{L}_{\text{FM}}$ i i' $\hat{i}'$
$\mathcal{L}^i_p$ i i'
$\mathcal{L}^G_{\text{zRec}}$ z_G $\hat{z}_G$

Figure 4.4: A modified method from section 3 using RGBA foreground $\boldsymbol{i}$ in place of $\boldsymbol{m}$. The encoding network $E_{\boldsymbol{i}}$ produces two separate latent vectors, one to condition the fully-connected network predicting per Gaussian transforms, and one to condition the foreground generator $G_{\boldsymbol{i}}$. Here, the network jointly learns shape and texture, where generation of the alpha channel is an equivalent task to predicting $\boldsymbol{m}$. A perceptual loss $\mathcal{L}^{\boldsymbol{i}}_p$ similar to that found in section 3.2 is introduced, as well as a reconstruction loss $\mathcal{L}_{\boldsymbol{z}}$ between $\mathbf{z}_G$ and $\hat{\mathbf{z}}_G$.

Foreground encoder. The encoding network $E_{\boldsymbol{i}}$ produces two separate latent vectors, $\mathbf{z}_{\mathbf{T}}$ and $\mathbf{z}_G$, through the use of two divergent fully-connected layers appended to the end of the convolutional network. The former latent vector is used to infer the Gaussian transformation parameters $\mathbf{T}_k$ and $\mathbf{T}_o$, while the latter is tiled and concatenated with the projected Gaussian maps $\boldsymbol{g}$ and fed into $G_{\boldsymbol{i}}$.

Additional losses. In the same manner as the generator described in section 3.2, we encourage detailed texture generation with a perceptual loss $\mathcal{L}^{\boldsymbol{i}}_p$ using the RGB channels of $\boldsymbol{i}$ and $\boldsymbol{i}'$. Additionally, to restrict the network from encoding object shape into $\mathbf{z}_G$, and thus bypassing the Gaussian prediction network, we apply a $\mathbf{z}_G$ reconstruction loss $\mathcal{L}^G_{\mathbf{z}\text{Rec}}(\mathbf{z}_G, \hat{\mathbf{z}}_G) = \|\mathbf{z}_G - \hat{\mathbf{z}}_{\mathbf{G}}\|_1$. All together, this gives us the loss combination:

$$\mathcal{L}(E_{\mathcal{G}^c}, E_{\boldsymbol{i}}, G_{\boldsymbol{i}}, D_{\boldsymbol{i}}) = \lambda_1\mathcal{L}_{\text{Rec}} + \lambda_2\mathcal{L}_{\boldsymbol{g}} + \lambda_3\mathcal{L}_{\hat{\mathcal{G}}} + \lambda_4\mathcal{L}_{\hat{\boldsymbol{g}}} + \lambda_5\mathcal{L}_{\text{Adv}} + \lambda_6\mathcal{L}_{\text{FM}} + \lambda_7\mathcal{L}^{\boldsymbol{i}}_p + \lambda_8\mathcal{L}^G_{\mathbf{z}\text{Rec}}$$

4.2.2 Results.

Training on the giraffe dataset, we see that the network is able to reproduce the input image well, but often struggles to generate convincing results when the camera is rotated (Fig. 4.6). The generated shape can become distorted, with the head of the giraffe being a particularly challenging component to render. The pose of the inferred Gaussians sometimes does not correspond with the generated image's pose. Texture generation is also inconsistent, with lighting changing noticeably during rotation (see third example in Fig. 4.6).

One of the Gaussians in the canonical representation learned by the network is placed away from the giraffe's body and orbits as a small sphere. When this Gaussian aligns with the neck under certain rotations, it can influence generation. Otherwise, it is neglected by the network. Though

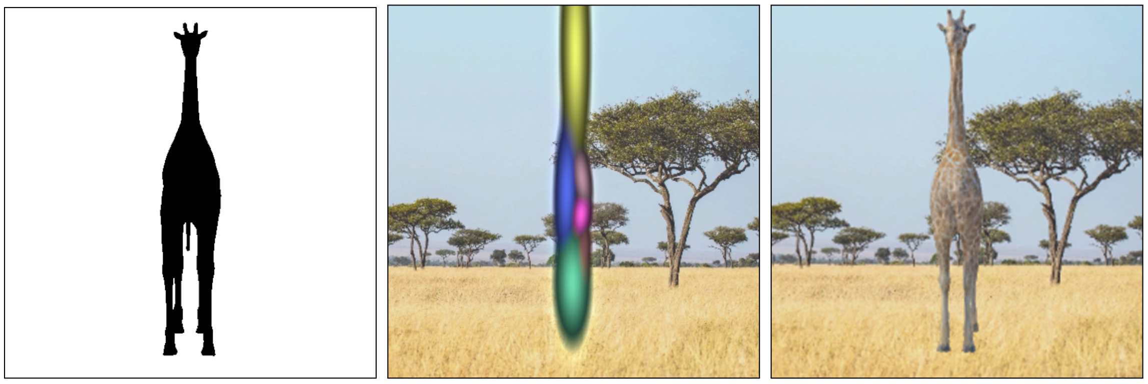


Figure 4.5: It is sometimes difficult to disambiguate a pose as back-to-front or front-to-back from only a silhouette. In this example, the model takes as input a back-to-front mask (*left*), produces Gaussians describing a front-to-back giraffe (*center*), and textures the generated mask as an indeterminate mix between the two poses (*right*).

it is unsatisfying that the network chooses to discard one of the available Gaussians, it should be noted that such neglected Gaussians can be pruned from the network after training.

4.2.3 Discussion

This experiment demonstrates the strengths of inferring mask and texture separately. It is evidently easier for a single network to learn one of these problems than to learn them together. In addition to this, the advantages we hoped to demonstrate with this method failed to materialize. There is no indication that the cues for pose provided by input texture rather than silhouette alone were used by the network. The same instances of front-to-back and back-to-front ambiguity we observed in our original method can be found here as well.

It is possible that the network is simply underpowered for the task at hand. Creating separate encoder networks for $\mathbf{z}_G$ and $\mathbf{z}_T$ may lead to better results, as $\mathbf{z}_G$ is intended to encode texture while $\mathbf{z}_T$ must encode something related to the shape of the object.

4.3 Ray marching Gaussian densities

In our primary method, the 3D Gaussian set is analytically projected into 2D Gaussian parameters, then rendered into individual Gaussian maps that are fed into the mask generator. This method is unable to produce occlusion, an important visual cue for depth ordering that could assist the network in cases of pose ambiguity.

One way we might create occlusion is through a process similar to the painter's algorithm: We sort the Gaussians by the distance between their $\boldsymbol{\mu}$ vector and the camera center, and render the 2D Gaussian projections from nearest to farthest, multiplying by one minus the sum of the previously rendered Gaussians as we go. This procedure can be thought of as rendering 2D Gaussian planes parallel to the image plane. However, while sorting is implemented in the TensorFlow framework

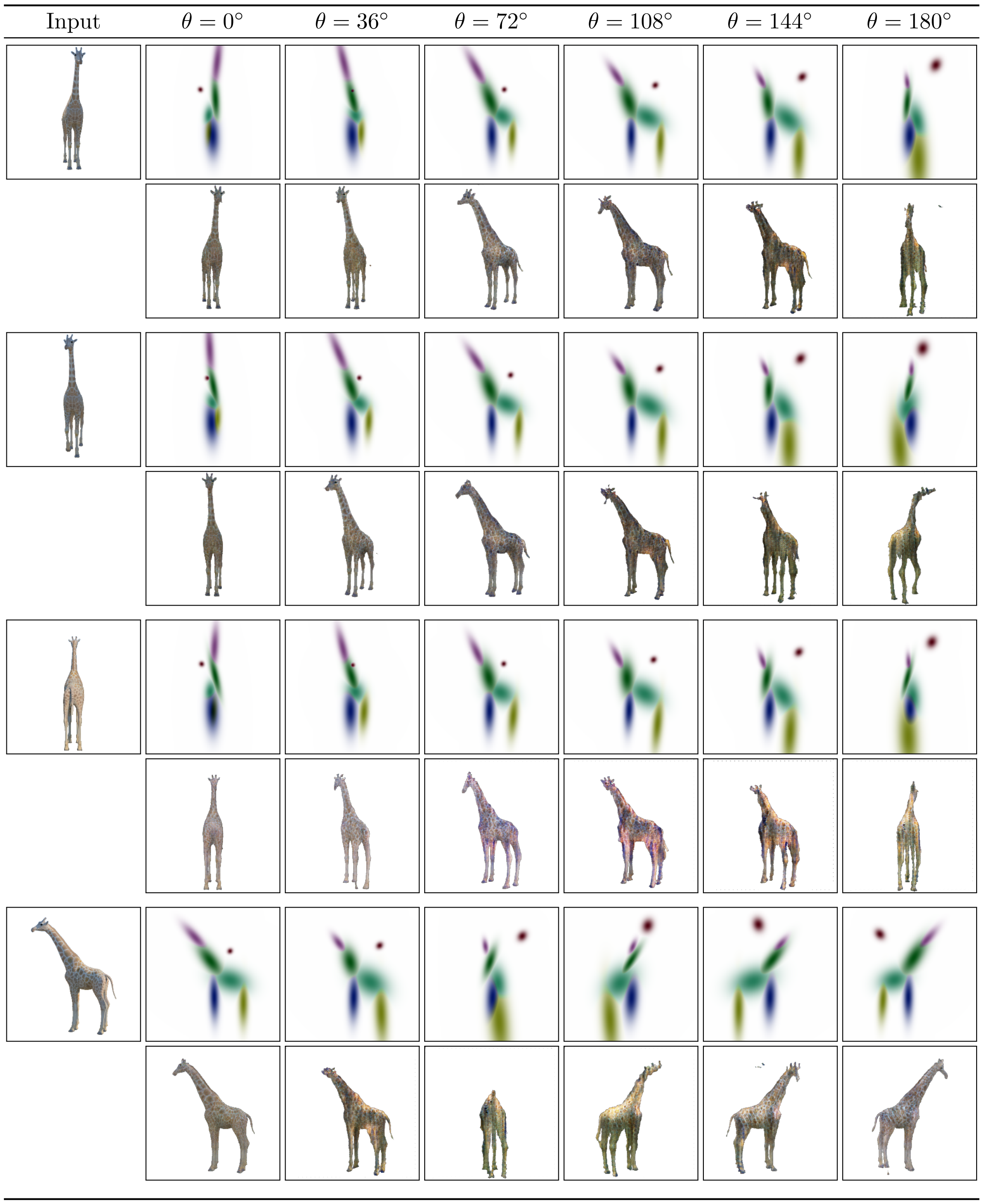


Figure 4.6: Generating RGBA foreground images from RGBA input. Though the network is able to recreate the input image with reasonable accuracy, rotating the Gaussians produces inferior generations.

and allows for gradients to pass through, it does not produce informative gradients itself that could guide learning of Gaussian visibility. Even when using "soft" sorting functions such as [2] that are fully differentiable, the issue remains that this planar approximation for Gaussian visibility does not model object-part visibility well; rotating the camera can cause Gaussians to dramatically "flicker" in and out of view as sort order changes, while the true object parts they represent are at least partially visible throughout the rotation.

Rhodin *et al.* [34] propose an analytically differentiable model with transmittance for rendering isotropic 3D Gaussians, but the isotropic constraint is crucial to producing a closed-form solution for computing transmittance at any point along a ray cast through the Gaussian densities.

4.3.1 Method

To implement a model incorporating transmittance for the rendering of *anisotropic* 3D Gaussians, we utilize volumetric ray marching. Formally, we define the density at position $\mathbf{x}$ to be the sum of Gaussians in $\mathcal{G}$:

$$\text{Density}(\mathbf{x}) = \sum_{\mathcal{G}_k \in \mathcal{G}} \mathcal{G}_k(\mathbf{x}) \tag{4.1}$$

where $\mathcal{G}_k(\mathbf{x})$ remains the same as defined in eq. 3.1. The transmittance from a camera center $\mathbf{o}$, in the direction $\mathbf{n}$, and at a distance s is then defined in accordance to the Beer-Lambert law of light absorption:

$$T(\mathbf{o}, \mathbf{n}, s) = \exp\left(-\int_0^s \text{Density}(\mathbf{o} + t\mathbf{n})\ \mathrm{d}t\right) \tag{4.2}$$

The total radiance $L_{\mathcal{G}_k}$ of a Gaussian $\mathcal{G}_k$ along a ray, with ambient radiance L_e, is defined as

$$L_{\mathcal{G}_k}(\mathbf{o}, \mathbf{n}) = \int_0^\infty T(\mathbf{o}, \mathbf{n}, s) \cdot \mathcal{G}_k(\mathbf{o} + s\mathbf{n}) \cdot L_e\ \mathrm{d}s \tag{4.3}$$

To approximate T, we take n steps of size α along the ray parametrized by $\mathbf{o}$ and $\mathbf{n}$:

$$\begin{aligned} \widehat{T}(\mathbf{o}, \mathbf{n}, \alpha, n) &= \exp\left(-\sum_{j=0}^{n} \text{Density}(\mathbf{o} + (j \cdot \alpha)\mathbf{n}) \cdot \alpha\right) && (4.4)\\ &= \prod_{j=0}^{n} \exp\left(-\text{Density}(\mathbf{o} + (j \cdot \alpha)\mathbf{n}) \cdot \alpha\right) && (4.5)\end{aligned}$$

Approximating $L_{\mathcal{G}_k}$ follows the same logic:

$$\begin{aligned} \widehat{L}_{\mathcal{G}_k}(\mathbf{o}, \mathbf{n}, \alpha, n) &= \sum_{i=0}^{n} \widehat{T}(\mathbf{o}, \mathbf{n}, \alpha, i) \cdot \mathcal{G}_k(\mathbf{o} + (i \cdot \alpha)\mathbf{n}) \cdot L_e \cdot \alpha && (4.6)\\ &= \sum_{i=0}^{n} \left[\prod_{j=0}^{i} \big(\exp\left(-\text{Density}(\mathbf{o} + (j \cdot \alpha)\mathbf{n}) \cdot \alpha\right)\big) \cdot \mathcal{G}_k(\mathbf{o} + (i \cdot \alpha)\mathbf{n}) \cdot L_e \cdot \alpha\right] && (4.7)\end{aligned}$$

An efficient implementation of this summation stores transmittance in a variable for accumulation by successive multiplications at each iteration.

Attempting to render the 3D Gaussians in this manner at the full resolution of 256×256 proved too computationally intensive, with multiple hours spent per epoch. To alleviate the computational burden of ray marching, we rendered each Gaussian map at a resolution of 32×32 and obtained maps of sizes 64×64, 128×128, and 256×256 through bicubic interpolation. Because of the generally smooth appearance of Gaussian densities, interpolating from a low-resolution is reasonable (Fig. 4.7). For the values of α, n, and L_e, we used 0.15, 20, and 1.0, respectively.

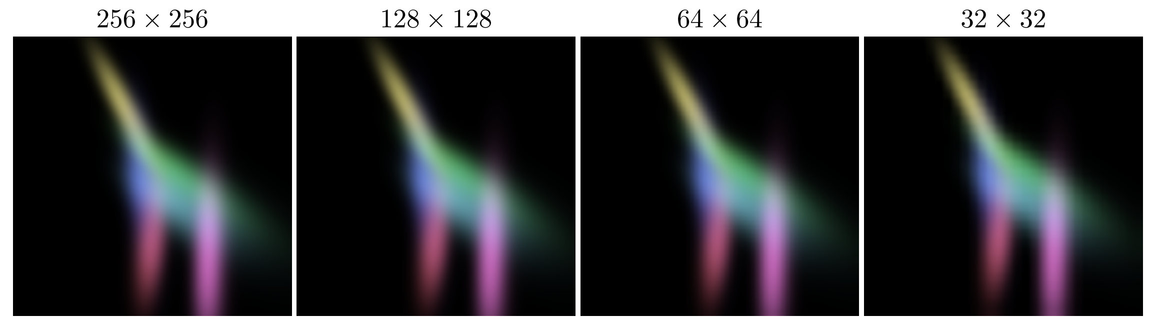


Figure 4.7: Ray marching Gaussian densities at successively lower resolutions and using bicubic interpolation to rescale them to the size of 256×256 (sizes at tops of images indicate the resolution in which they were rendered using ray marching). We see that the smooth appearance of the Gaussians makes them suitable for low-dimensional rendering and rescaling. *Note:* The parameters for these Gaussians were not learned but hard-coded for purposes of demonstration.

4.3.2 Results

In general, we found that the network struggled to learn a useful representation for mask generation (Fig. 4.8). Gaussians were grouped to the center of the camera's field of view and lacked an informative shape to guide the generator. Besides a clear distinction between the neck and body of the giraffe, object parts were not well-differentiated by the Gaussian prediction unit. The generator was required to extrapolate large features of the output shape from these ambiguous Gaussian maps. The mask does not fail to rotate with the camera, but inconsistent shape prevents any interpretation of the output as a coherent 3D space.

4.3.3 Discussion

Ray marching Gaussian densities is a much more complex method of producing 2D maps than the projection of 3D to 2D Gaussian parameters. This additional complexity may make the learning of the 3D representation considerably harder. Testing different parameters and new losses may make this method more successful. However, even after greatly reducing the resolution in which the 3D Gaussians are ray marched, training takes roughly three times as long as our original method. Deriving a closed-form solution to the transmittance of anisotropic Gaussian densities similar to [34] would greatly improve the tractability of this method.

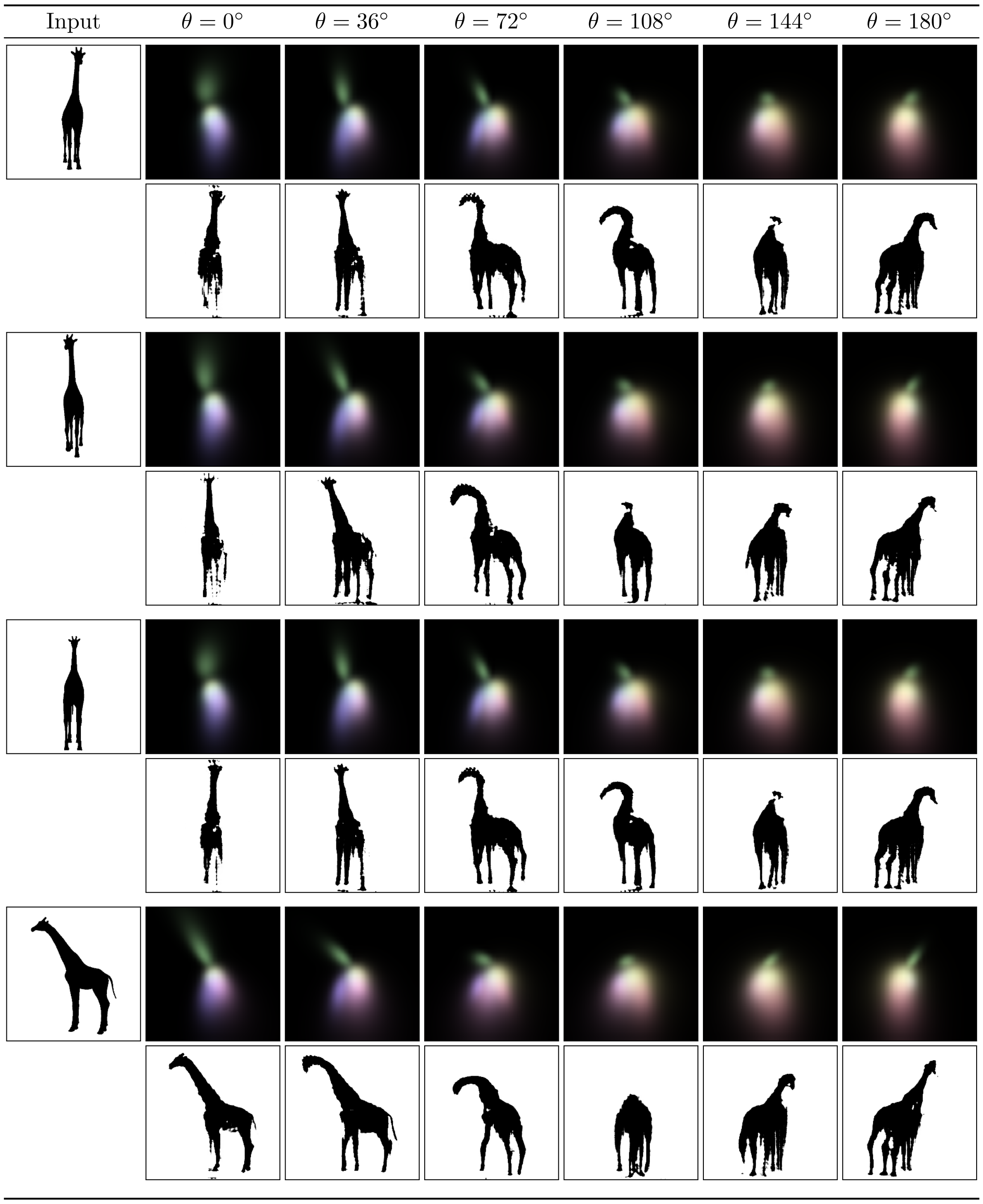
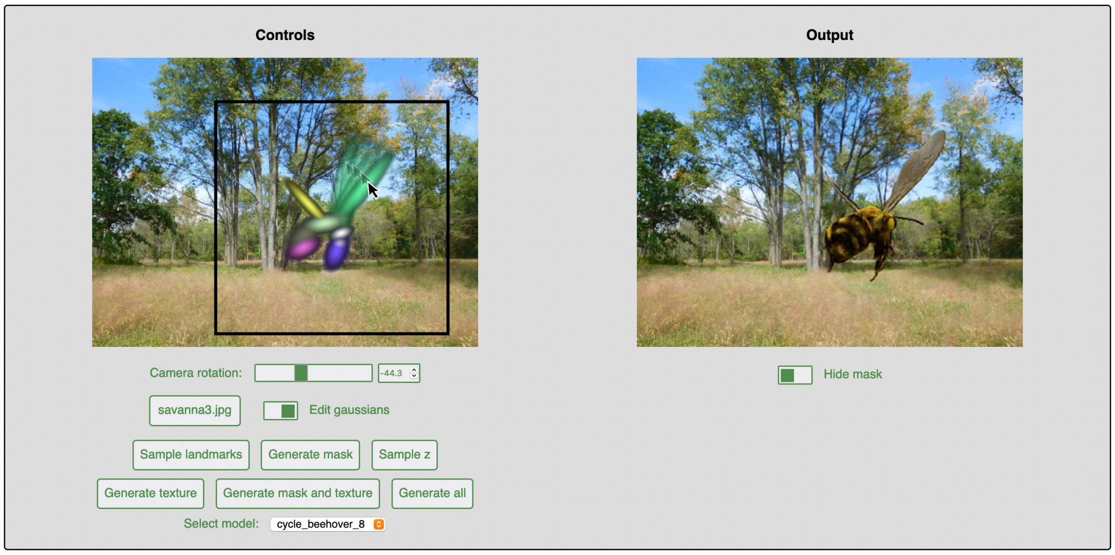


Figure 4.8: Mask generation conditioned upon ray marched Gaussian densities for four different input masks. Images of the Gaussian sets in this figure have not been inverted so as to remain consistent with the additive mixing of colored light.

Chapter 5

Graphical User Interface



Camera rotation: -44.3

savanna3.jpg

Select model: cycle_beehover_8

Figure 5.1: A Flask web app interface for interaction with the model.

A key advantage of our method is its ability to produce a coarse 3D representation which can be easily manipulated to express the shape and pose of the generated object. To fully demonstrate this feature of our network, we present a graphical user interface operated through the user's web browser that allows for intuitive interaction with the Gaussian parameters and generative networks (Fig. 5.1).

5.1 Controls

The left canvas of the interface has two modes. The first allows the user to draw and translate a bounding box designating the region where the object is generated. The second mode allows the user to translate, rotate, and scale the individual Gaussians, with each of these sub-modes toggled by specific key presses. Rotation of the Gaussians is achieved via the Arcball input technique [37]. Rotation of the camera is controlled by a slider below the canvas. Background images can be uploaded by the user, and the different trained models available can be selected through a drop-down menu.

The right canvas of the interface displays the composited output of the network. The user can toggle whether a semi-transparent overlay of the generated mask is visible. This overlay translates and scales with the bounding box, guiding for the target position of texture generation.

An array of buttons found on the left side of the interface define various actions relating to the models. The “Sample landmarks” button selects an input mask from a directory at random, and infers a 3D Gaussian set corresponding to that mask. The “Generate mask” button generates a mask for the given Gaussian set. The “Sample z” button randomly samples a new $\mathbf{z_i}$ vector conditioning the texture generator. The “Generate texture” button generates texture for the current mask and background defined by the bounding box. The “Generate mask and texture” button generates the mask image and texture image at once. The “Generate all” button samples the landmarks, generates the mask, and generates the texture image ($\mathbf{z_i}$ remains constant unless “Sample z” is pressed or a new model is selected).

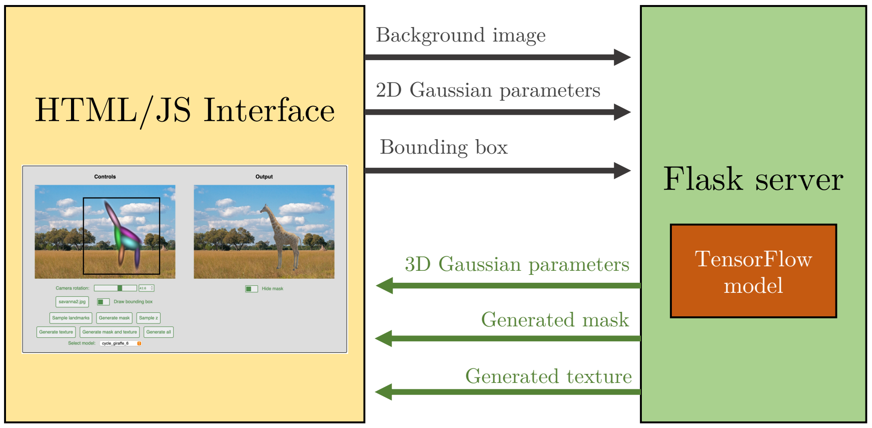


Figure 5.2: The structure of the Flask web app interface. Projection and rendering of the 3D Gaussians are handled in JavaScript on the client side while the TensorFlow model runs on the Flask server.

5.2 Structure

Computation in our application is divided across its two parts: the front end HTML/JavaScript interface and the back end Flask server (Fig. 5.2). On the client side, a WebGL shader program takes 3D Gaussian parameters passed from the Flask server and implements the perspective camera projection function π. The resulting projected 2D Gaussian parameters are written to a texture, and are extracted from that texture outside of the shader program and stored as JavaScript variables.

These parameters are then available both to be sent back to the server for use in the TensorFlow model, and as input for a separate WebGL shader program for rendering the Gaussians in the webpage. Implementing both the projection program and 2D Gaussian renderer as WebGL shaders enables smooth, real-time manipulation of the Gaussians.

Chapter 6

Conclusion

The method presented in this paper demonstrates how a coarse 3D representation can be learned from unposed 2D images with only mask supervision through the task of object image generation. By simplifying the representation our model learns to a set of 3D Gaussians, we differ from previous work that seeks instead to explicitly model 3D features from natural images. Our low-dimensional Gaussian parameterization not only learns more consistent 3D spaces, but allows for immediate interaction and manipulation by the user during inference.

While successful in these respects, our method also lacks certain desired features. The model is easily confused by ambiguously posed silhouettes, leading to conflicts between the inferred Gaussians and the generated image. Attempting to strengthen the connection between the 3D Gaussian representation and generated output, we experimented both with the joint learning of shape and texture, as well as utilizing a transmittance-enabled rendering function that can produce occlusion. Both of these attempts did not achieve the desired effect and call for further experiments.

A possible future direction of this work is the association of more information with each of the individual Gaussians. As the method stands now, both the mask and texture generators receive the Gaussians simply as projected 2D maps. For both generators, associating depth with these Gaussian maps would assist in interpreting the pose of the Gaussian representation and generating accurate output. Furthermore, the texture generator receives only a single vector $\mathbf{z}_i$ for describing the texture of the entire object. Associating texture vectors to each Gaussian individually would enable modularity and greater flexibility in texture synthesis.

Bibliography

[1] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3D point clouds. In *ICML*, pages 40–49. PMLR, 2018.

[2] Mathieu Blondel, Olivier Teboul, Quentin Berthet, and Josip Djolonga. Fast differentiable sorting and ranking. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 950–959. PMLR, 13–18 Jul 2020.

[3] Boyang Deng, Kyle Genova, Soroosh Yazdani, Sofien Bouaziz, Geoffrey Hinton, and Andrea Tagliasacchi. Cvxnet: Learnable convex decomposition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[4] Kyle Genova, Forrester Cole, Avneesh Sud, Aaron Sarna, and Thomas Funkhouser. Local deep implicit functions for 3D shape. In *CVPR*, pages 4857–4866, 2020.

[5] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, 2014.

[6] Philipp Henzler, Niloy J Mitra, and Tobias Ritschel. Escaping Plato's cave: 3D shape from adversarial rendering. In *ICCV*, pages 9984–9993, 2019.

[7] Seunghoon Hong, Xinchen Yan, Thomas Huang, and Honglak Lee. Learning hierarchical semantic image manipulation through structured representations. In *NeurIPS*, 2018.

[8] Tomas Jakab, Ankush Gupta, Hakan Bilen, and Andrea Vedaldi. Unsupervised learning of object landmarks through conditional image generation. In *NeurIPS*, 2018.

[9] Tomas Jakab, Ankush Gupta, Hakan Bilen, and Andrea Vedaldi. Self-supervised learning of interpretable keypoints from unlabelled videos. In *CVPR*, 2020.

[10] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016.

[11] Angjoo Kanazawa, Shubham Tulsiani, Alexei A Efros, and Jitendra Malik. Learning category-specific mesh reconstruction from image collections. In *ECCV*, pages 371–386, 2018.

[12] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *CVPR*, 2019.

[13] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *CVPR*, pages 8110–8119, 2020.

[14] Yunji Kim, Seonghyeon Nam, In Cho, and Seon Joo Kim. Unsupervised keypoint learning for guiding class-conditional video prediction. In *NeurIPS*, 2019.

[15] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *ICLR*, 2014.

[16] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. Pointrend: Image segmentation as rendering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9799–9808, 2020.

[17] Markus Knoche, István Sárándi, and Bastian Leibe. Reposing humans by warping 3D features. In *CVPR Workshop on Towards Human-Centric Image/Video Synthesis*, 2020.

[18] Nilesh Kulkarni, Abhinav Gupta, David F Fouhey, and Shubham Tulsiani. Articulation-aware canonical surface mapping. In *CVPR*, pages 452–461, 2020.

[19] Donghoon Lee, Sifei Liu, Jinwei Gu, Ming-Yu Liu, Ming-Hsuan Yang, and Jan Kautz. Context-aware synthesis and placement of object instances. In *NIPS*, pages 10393–10403, 2018.

[20] Jiahui Lei, Srinath Sridhar, Paul Guerrero, Minhyuk Sung, Niloy Mitra, and Leonidas J. Guibas. Pix2Surf: Learning parametric 3D surface models of objects from images. In *ECCV*, 2020.

[21] Yuheng Li, Krishna Kumar Singh, Utkarsh Ojha, and Yong Jae Lee. MixNMatch: Multifactor disentanglement and encoding for conditional image generation. In *CVPR*, 2020.

[22] Yiyi Liao, Katja Schwarz, Lars Mescheder, and Andreas Geiger. Towards unsupervised learning of generative models for 3D controllable image synthesis. In *CVPR*, 2020.

[23] Jae Hyun Lim and Jong Chul Ye. Geometric GAN. *arXiv preprint arXiv:1705.02894*, 2017.

[24] Chen-Hsuan Lin, Ersin Yumer, Oliver Wang, Eli Shechtman, and Simon Lucey. ST-GAN: Spatial transformer generative adversarial networks for image compositing. In *CVPR*, 2018.

[25] Tony Lindeberg. Image matching using generalized scale-space interest points. *Journal of Mathematical Imaging and Vision*, 52(1):3–36, 2015.

[26] Dominik Lorenz, Leonard Bereska, Timo Milbich, and Bjorn Ommer. Unsupervised part-based disentangling of object shape and appearance. In *CVPR*, pages 10955–10964, 2019.

[27] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *ECCV*, 2020.

[28] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *ICLR*, 2018.

[29] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *ECCV*, 2016.

[30] 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.

[31] Pavel Ostyakov, Roman Suvorov, Elizaveta Logacheva, Oleg Khomenko, and Sergey I. Nikolenko. SEIGAN: Towards compositional image generation by simultaneously learning to segment, enhance, and inpaint. *arXiv preprint arXiv:1811.07630*, 2018.

[32] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *CVPR*, 2019.

[33] Varun Ramakrishna, Daniel Munoz, Martial Hebert, James Andrew Bagnell, and Yaser Sheikh. Pose machines: Articulated pose estimation via inference machines. In *ECCV*, 2014.

[34] Helge Rhodin, Nadia Robertini, Christian Richardt, Hans-Peter Seidel, and Christian Theobalt. A versatile scene model with differentiable visibility applied to generative pose estimation. In *ICCV*, pages 765–773, 2015.

[35] Tim Saliman, Ian Goodfellow, Wojciech Zaremba, and Vicki Cheung. Improved techniques for training GANs. In *NeurIPS*, 2016.

[36] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. GRAF: Generative radiance fields for 3D-aware image synthesis. In *NeurIPS*, 2020.

[37] Ken Shoemake. Arcball: A user interface for specifying three-dimensional orientation using a mouse. In *Proceedings of the Conference on Graphics Interface '92*, page 151–156, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[38] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhofer. Deepvoxels: Learning persistent 3D feature embeddings. In *CVPR*, pages 2437–2446, 2019.

[39] 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.

[40] Ayush Tewari, Mohamed Elgharib, Gaurav Bharaj, Florian Bernard, Hans-Peter Seidel, Patrick Pérez, Michael Zollhofer, and Christian Theobalt. Stylerig: Rigging StyleGAN for 3D control over portrait images. In *CVPR*, pages 6142–6151, 2020.

[41] James Thewlis, Hakan Bilen, and Andrea Vedaldi. Unsupervised learning of object frames by dense equivariant image labelling. In *NIPS*, 2017.

[42] James Thewlis, Hakan Bilen, and Andrea Vedaldi. Unsupervised learning of object landmarks by factorized spatial embeddings. In *ICCV*, 2017.

[43] Dustin Tran, Rajesh Ranganath, and David M. Blei. Deep and hierarchical implicit models. *arXiv preprint arXiv:1702.08896*, 7, 2017.

[44] Yi-Hsuan Tsai, Xiaohui Shen, Zhe Lin, Kalyan Sunkavalli, Xin Lu, and Ming-Hsuan Yang. Deep image harmonization. In *CVPR*, 2017.

[45] Shubham Tulsiani, Hao Su, Leonidas J. Guibas, Alexei A. Efros, and Jitendra Malik. Learning shape abstractions by assembling volumetric primitives. In *Computer Vision and Pattern Regognition (CVPR)*, 2017.

[46] Mehmet Ozgur Turkoglu, William Thong, Luuk Spreeuwers, and Berkay Kicanaoglu. A layer-based sequential framework for scene generation with GANs. In *AAAI*, pages 8901–8908, 2019.

[47] Kohei Yamashita, Shohei Nobuhara, and Ko Nishino. 3D-GMNet: Single-view 3D shape recovery as a gaussian mixture. *BMVC*, 2020.

[48] Jianwei Yang, Anitha Kannan, Dhruv Batra, and Devi Parikh. LR-GAN: Layered recursive generative adversarial networks for image generation. In *ICLR*, 2017.

[49] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, pages 586–595, 2018.

[50] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*, 2017.

[51] 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.

[52] Silvia Zuffi, Angjoo Kanazawa, Tanya Berger-Wolf, and Michael J. Black. Three-D safari: Learning to estimate zebra pose, shape, and texture from images “in the wild”. In *ICCV*, pages 5358–5367, 2019.