BlockGAN: Learning 3D Object-aware Scene Representations from Unlabelled Images

Thu Nguyen-Phuoc
University of Bath

Christian Richardt
University of Bath

Long Mai
Adobe Research

Yong-Liang Yang
University of Bath

Niloy Mitra
Adobe Research & UCL

Abstract

We present BlockGAN, an image generative model that learns object-aware 3D scene representations directly from unlabelled 2D images. Current work on scene representation learning either ignores scene background or treats the whole scene as one object. Meanwhile, work that considers scene compositionality treats scene objects only as image patches or 2D layers with alpha maps. Inspired by the computer graphics pipeline, we design BlockGAN to learn to first generate 3D features of background and foreground objects, then combine them into 3D features for the whole scene, and finally render them into realistic images. This allows BlockGAN to reason over occlusion and interaction between objects' appearance, such as shadow and lighting, and provides control over each object's 3D pose and identity, while maintaining image realism. BlockGAN is trained end-to-end, using only unlabelled single images, without the need for 3D geometry, pose labels, object masks, or multiple views of the same scene. Our experiments show that using explicit 3D features to represent objects allows BlockGAN to learn disentangled representations both in terms of objects (foreground and background) and their properties (pose and identity).

1 Introduction

The computer graphics pipeline has achieved impressive results in generating high-quality images, while offering users a great level of freedom and controllability over the generated images. This has many applications in creating and editing content for the creative industries, such as films, games, scientific visualisation, and more recently, in generating training data for computer vision tasks. However, the current pipeline, ranging from generating 3D geometry and textures, rendering, compositing and image post-processing, can be very expensive in terms of labour, time, and costs.

Recent image generative models, in particular generative adversarial networks [GANs; 16], have greatly improved the visual fidelity and resolution of generated images [7, 25, 26]. Conditional GANs [37] allow users to manipulate images, but require labels during training. Recent work on unsupervised disentangled representations using GANs [11, 26, 39] relaxes this need for labels. The ability to produce high-quality, controllable images has made GANs an increasingly attractive alternative to the traditional graphics pipeline for content generation. However, most work focuses on *property* disentanglement, such as shape, pose and appearance, without considering the compositionality of the images, i.e., scenes being made up of multiple objects. Therefore, they do not offer control over individual objects in a way that respects the interaction of objects, such as consistent lighting and shadows. This is a major limitation of current image generative models, compared to the graphics pipeline, where 3D objects are modelled individually in terms of geometry and appearance, and combined into 3D scenes with consistent lighting.

Preprint. Under review.

Even when considering object compositionality, most approaches treat objects as 2D layers combined using alpha compositing [14, 51, 54]. Moreover, they also assume that each object's appearance is independent [4, 8, 14]. While this layering approach has led to good results in terms of object separation and visual fidelity, it is fundamentally limited by the choice of 2D representation. Firstly, it is hard to manipulate properties that require 3D understanding, such as pose or perspective. Secondly, object layers tend to bake in appearance and cannot adequately represent view-specific appearance, such as shadows or material highlights changing as objects move around in the scene. Finally, it is non-trivial to model the appearance interactions between objects, such as scene lighting that affects objects' shadows on a background.

We introduce BlockGAN, a generative adversarial network that learns 3D object-oriented scene representations directly from unlabelled 2D images. Instead of learning 2D layers of objects and combining them with alpha compositing, BlockGAN learns to generate 3D object features and to combine them into deep 3D scene features that are projected and rendered as 2D images. This process closely resembles the computer graphics pipeline where scenes are modelled in 3D, enabling reasoning over occlusion and interaction between object appearance, such as shadows or highlights. During test time, each object's pose can be manipulated using 3D transforms directly applied to the object's deep 3D features. We can also add new objects to the generated image by introducing more 3D object features to the 3D scene features, even when BlockGAN was trained with scenes containing fewer objects. This shows that BlockGAN has learnt a non-trivial representation of objects and their interaction, instead of merely memorizing images.

BlockGAN is trained end-to-end in an unsupervised manner directly from unlabelled 2D images, without any multi-view images, paired images, pose labels, or 3D shapes. We experiment with BlockGAN on a variety of synthetic and natural image datasets. In summary, our main contributions are:

- BlockGAN, an unsupervised image generative model that learns an object-aware 3D scene representation directly from unlabelled 2D images, disentangling both between objects and individual object properties (pose and identity);
- showing that BlockGAN can learn to separate objects even from cluttered backgrounds; and
- demonstrating that BlockGAN's object features can be added and manipulated to create novel scenes that are not observed during training.

2 Related work

GANs. Unsupervised GANs learn to map samples from a latent distribution to data categorised as real by a discriminator network. Conditional GANs enable control over the generated image content, but require labels during training. Recent work on unsupervised disentangled representation learning using GANs provides controllability over the final images without the need for labels. Loss functions can be designed to maximize mutual information between generated images and latent variables [11, 22]. However, these models do not guarantee which factors can be learnt, and have limited success when applied to natural images. Network architectures can play a vital role in both improving training stability [9] and controllability of generated images [26, 39]. We also focus on designing an appropriate architecture to learn object-level disentangled representations. We show that injecting inductive biases about how the 3D world is composed of 3D objects enables BlockGAN to learn 3D object-aware scene representations directly from 2D images, thus providing control over both 3D pose and appearance of individual objects.

3D-aware neural image synthesis. Introducing 3D structures into CNNs can improve the quality [38, 42, 45, 49] and controllability of the image generation process [39, 40, 58]. This can be achieved with explicit 3D representations, like occupancy voxel grids [44, 58], meshes, or shape templates [30, 47], in conjunction with handcrafted differentiable renderers [10, 19, 34, 35]. Renderable deep 3D representations can also be learnt directly from images [39, 48, 49]. HoloGAN [39] further shows that adding inductive biases about the 3D structure of the world enables unsupervised disentangled feature learning between shape, appearance and pose. However, these learnt representations are either object-centric (i.e., no background), or treat the whole scene as one object. Thus, they do not consider scene compositionality, i.e., components that can move independently. BlockGAN, in contrast, is designed to learn object-aware 3D representations that are combined into a unified 3D scene representation.

Object-aware image synthesis. Recent methods decompose image synthesis into generating components like layers or image patches, and combining them into the final image [31, 51, 54]. This includes conditional GANs that use segmentation masks [41, 50], scene graphs [24], object labels, key points

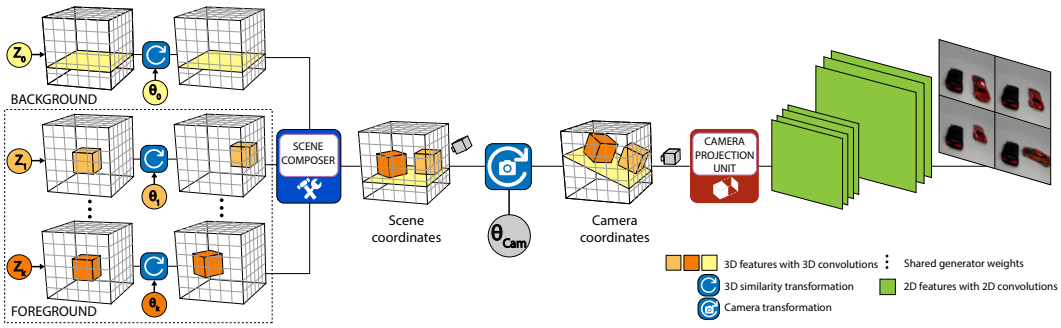

Figure 1: BlockGAN's generator network. Each noise vector $\mathbf{z}_i$ is mapped to deep 3D object features, which are transformed to the desired 3D pose $\boldsymbol{\theta}_i$. Object features are combined into 3D scene features, where the camera pose $\boldsymbol{\theta}_{\text{cam}}$ is applied, before projection to 2D features that produce the image $\mathbf{x}$.

or bounding boxes [20, 43], which have shown impressive results for natural image datasets. Recently, unsupervised methods [3, 14, 15, 29, 51, 56] learned object disentanglement for multi-object scenes on simpler synthetic datasets (single-colour objects, simple lighting, and material). Other approaches successfully separate foreground from background objects in natural images, but make strong assumptions about the size of objects [54] or independent object appearance [4, 8]. These methods treat object components as image patches or 2D layers with corresponding masks, which are combined via alpha compositing at the pixel level to generate the final stage. The work closest to ours learns to generate multiple 3D primitives (cuboids, spheres and point clouds), renders them into *separate* 2D layers with a handcrafted differentiable renderer, and alpha-composes them based on their depth ordering to create the final image [32]. Despite the explicit 3D geometry, this method does not handle cluttered backgrounds and requires extra supervision in the shape of labelled images with and without foreground objects.

BlockGAN takes a different approach. We treat objects as *learnt 3D features* with corresponding 3D poses, and learn to combine them into 3D scene features. Not only does this provide control over 3D pose, but also enables learning of realistic lighting and shadows. Our approach allows adding more objects into the 3D scene features to generate images with multiple objects, which are not observed at training time.

3 Method

Inspired by the computer graphics pipeline, we assume that each image $\mathbf{x}$ is a rendered 2D image of a 3D scene composed of K 3D foreground objects $\{O_1,...,O_K\}$ in addition to the background O_0:

$$\mathbf{x} = p\big(f(\underbrace{O_0}_{\text{background}}, \underbrace{O_1,...,O_K}_{\text{foreground}})\big), \tag{1}$$

where the function f combines multiple objects into unified scene features that are projected to the image $\mathbf{x}$ by p. We assume each object O_i is defined in a canonical orientation and generated from a noise vector $\mathbf{z}_i$ by a function g_i before being individually posed using parameters $\boldsymbol{\theta}_i$: $O_i = g_i(\mathbf{z}_i, \boldsymbol{\theta}_i)$.

We inject the inductive bias of compositionality of the 3D world into BlockGAN in two ways. (1) The generator is designed to first generate 3D features for each object independently, before transforming and combining them into unified scene features, in which objects interact. (2) Unlike other methods that use 2D image patches or layers to represent objects, BlockGAN directly learns from unlabelled images how to generate objects as 3D features. This allows our model to disentangle the scene into separate 3D objects and allows the generator to reason over 3D space, enabling object pose manipulation and appearance interaction between objects. BlockGAN, therefore, learns to both generate and render the scene features into images that can fool the discriminator.

Figure 1 illustrates the BlockGAN generator architecture. Each noise vector $\mathbf{z}_i$ is mapped to 3D object features O_i. Objects are then transformed according to their pose $\boldsymbol{\theta}_i$ using a 3D similarity transform, before being combined into 3D scene features using the *scene composer* f. The scene features are transformed into the camera coordinate system before being projected to 2D features to render the final images using the *camera projector* function p. During training, we randomly sample both the noise vectors $\mathbf{z}_i$ and poses $\boldsymbol{\theta}_i$. During test time, objects can be generated with a given identity $\mathbf{z}_i$ in the desired pose $\boldsymbol{\theta}_i$.

BlockGAN is trained end-to-end using only unlabelled 2D images, without the need for any labels, such as poses, 3D shapes, multi-view inputs, masks, or geometry priors like shape templates, symmetry or smoothness terms. We next explain each component of the generator in more detail.

3.1. Learning 3D object representations. Each object $O_i \in \mathbb{R}^{H_o \times W_o \times D_o \times C_o}$ is a deep 3D feature grid generated by $O_i = g_i(\mathbf{z}_i, \boldsymbol{\theta}_i)$, where g_i is an object generator that takes as input a noise vector $\mathbf{z}_i$ controlling the object appearance, and the object's 3D pose $\boldsymbol{\theta}_i = (s_i, \mathbf{R}_i, \mathbf{t}_i)$, which comprises its uniform scale $s_i \in \mathbb{R}$, rotation $\mathbf{R}_i \in \mathrm{SE}(3)$ and translation $\mathbf{t}_i \in \mathbb{R}^3$. The object generator g_i is specific to each category of objects, and is shared between objects of the same category. We assume that 3D scenes consist of at least two objects: the background O_0 and one or more foreground objects $\{O_1, \ldots, O_K\}$. This is different to object-centric methods that only assume a single object with a simple white background [48], or only deal with static scenes whose object components cannot move independently [39]. We show that, even when BlockGAN is trained with only one foreground and background object, we can add an arbitrary number of foreground objects to the scene at test time.

To generate 3D object features, BlockGAN implements the style-based strategy, which helps to disentanglement between pose and identity [39] while improving training stability [26]. As illustrated in Figure 2, the noise vector $\mathbf{z}_i$ is mapped to affine parameters – the "style controller" – for adaptive instance normalization [AdaIN; 21] after each 3D convolution layer. However, unlike HoloGAN [39], which learns 3D features directly for the whole scene, BlockGAN learns 3D features for *each* object, which are then transformed to their target poses using similarity transforms, and combined into 3D *scene* features. We implement these 3D similarity transforms by trilinear resampling of the 3D features according to the translation, rotation and scale parameters $\boldsymbol{\theta}_i$; samples falling outside the feature tensor are clamped to zero. This allows BlockGAN to not only separate object pose from identity, but also to disentangle multiple objects in the same scene.

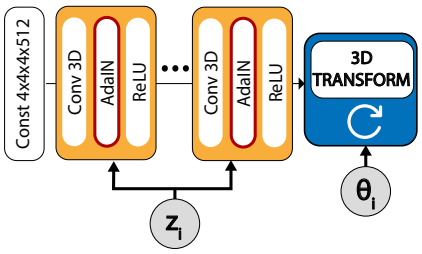

Figure 2: BlockGAN's object generator. Each object starts with a constant tensor that is learnt with the rest of the network.

3.2. Scene composer function. We combine the 3D object features $\{O_i\}$ into scene features $S = f(O_0, O_1, \ldots, O_K) \in \mathbb{R}^{H_s \times W_s \times D_s \times C_s}$ using a scene composer function f. For this, we use the element-wise maximum as it achieves the best image quality compared to element-wise summation and a multi-layer perceptron (MLP); please see our supplemental document for an ablation. Additionally, the maximum is invariant to permutation and allows a flexible number of input objects to add new objects into the scene features during test time, even when trained with fewer objects (see Section 4.3).

3.3. Learning to render. Instead of using a handcrafted differentiable renderer, we aim to learn rendering directly from unlabelled images. HoloGAN showed that this approach is more expressive as it is capable of handling unlabelled, natural image data. However, their projection model is limited to a weak perspective, which does not support foreshortening – an effect that is observed when objects are close to real (perspective) cameras. We therefore introduce a graphics-based perspective projection function $p\colon \mathbb{R}^{H_s \times W_s \times D_s \times C_s} \mapsto \mathbb{R}^{H_c \times W_c \times C_c}$ that transforms the 3D scene features into camera space using a projective transform, and then learns the projection of the 3D features to a 2D feature map.

The computer graphics pipeline implements perspective projection using a projective transform that converts objects from world coordinates (our scene space) to camera coordinates [36]. We implement this camera transform like the similarity transforms used to manipulate objects in Section 3.1, by resampling the 3D scene features according to the viewing volume (frustum) of the virtual perspective camera (see Figure 3). For correct perspective projection, this transform must be a projective transform, the superset of similarity transforms [53]. Specifically, the viewing frustum, in scene space, can be defined relative to the camera's pose $\boldsymbol{\theta}_{\text{cam}}$ using the angle of view,

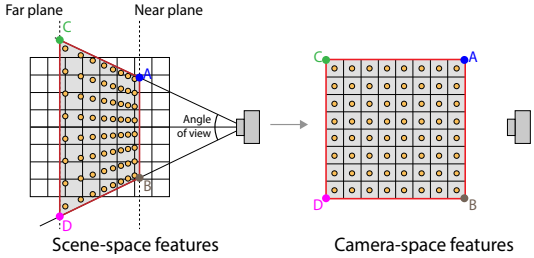

Figure 3: **Left:** The camera's viewing volume (frustum) overlaid on scene-space features. We trilinearly resample the scene features based on the viewing volume at the orange dots. **Right:** The resulting camera-space features before projection to 2D.

and the distance of the near and far planes. The
camera-space features are a new 3D tensor of features, of size $H_c \times W_c \times D_c \times C_s$, whose corners are mapped to the corners of the camera's viewing frustum using the unique projective 3D transform computed from the coordinates of corresponding corners using the Direct Linear Transform [18].

In practice, we combine the object and camera transforms into a single transform by multiplying both transform matrices and resampling the object features in a single step, directly from object to camera space. This is computationally more efficient than resampling twice, and advantageous from a sampling theory point of view, as the features are only interpolated once, not twice, and thus less information is lost by the resampling. The combined transform is a fixed, differentiable function with parameters $(\boldsymbol{\theta}_i, \boldsymbol{\theta}_{\text{cam}})$. The individual objects are then combined in camera space before the final projection.

After the camera transform, the 3D features are projected into view-specific 2D feature maps using the *learnt* camera projection $p' : \mathbb{R}^{H_c \times W_c \times D_c \times C_s} \mapsto \mathbb{R}^{H_c \times W_c \times C_c}$. This function ensures that occlusion correctly shows nearby objects in front of distant objects. Following the RenderNet projection unit [38], we reshape the 3D camera-space features (with depth D_c and C_s channels) into a 2D feature map with $(D_c \cdot C_s)$ channels, followed by a per-pixel MLP (i.e., 1×1 convolution) that outputs C_c channels.

3.4. Loss functions. We train BlockGAN adversarially using the non-saturating GAN loss [16]. For natural images with cluttered backgrounds, we also add a style discriminator loss [39]. In addition to classifying the images as real or fake, this discriminator also looks at images at the feature level. Given image features $\boldsymbol{\Phi}_l$ at layer l, the style discriminator classifies the mean $\boldsymbol{\mu}(\boldsymbol{\Phi}_l)$ and standard deviation $\boldsymbol{\sigma}(\boldsymbol{\Phi}_l)$ over the spatial dimensions, which describe the image "style" [21]. This more powerful discriminator discourages the foreground generator to include parts of the background within the foreground object(s). We provide detailed network and loss definitions in the supplemental material.

4 Experiments

Datasets. We train BlockGAN on images at 64×64 pixels, with increasing complexity in terms of number of foreground objects (1–4) and texture (synthetic images with simple shapes and simple to natural images with complex texture and cluttered background). These datasets include the synthetic CLEVRn [23], SYNTH-CARn and SYNTH-CHAIRn, and the real REAL-CAR [55], where n is the number of foreground objects. Additional details and results are included in the supplementary material.

Implementation details. We assume a fixed and known number of objects of the same type. Fore- and background generators have similar architectures and the same number of output channels, but foreground generators have twice as many channels in the learnt constant tensor. Since foreground objects are smaller than the background, we set scale=1 for the background object, and randomly sample scales <1 for foreground objects. Please see our supplemental material for more implementation details. We will make our code publicly available.

4.1. Qualitative results. Despite being trained with only unlabelled images, Figure 4 shows that BlockGAN learns to disentangle different objects within a scene: foreground from background, and between multiple foreground objects. More importantly, BlockGAN also provides explicit control and enables smooth manipulation of each object's pose $\boldsymbol{\theta}_i$ and identity $\mathbf{z}_i$. Figure 5 shows results on natural images with a cluttered background, where BlockGAN is still able to separate objects and enables 3D object-centric modifications. Since BlockGAN combines deep object features into scene features, changes in an object's properties also influence its shadows and highlights adapt to the object's movement. These effects can be better observed in the supplementary animations.

4.2. Quantitative results. We evaluate the visual fidelity of BlockGAN's results using Kernel Inception Distance [KID; 5], which has an unbiased estimator and works even for a small number of images. Note that KID does *not* measure the quality of object disentanglement, which is the main contribution of BlockGAN. We first compare with a vanilla GAN [WGAN-GP; 17] using a publicly available implementation[1]. Secondly, we compare with LR-GAN [54], a 2D-based method that learns to generate image background and foregrounds separately and recursively. Finally, we compare with HoloGAN, which learns 3D scene representations that separate camera pose and identity, but does not consider object disentanglement. For LR-GAN and HoloGAN, we use the authors' code. We tune

[1] https://github.com/LynnHo/DCGAN-LSGAN-WGAN-WGAN-GP-Tensorflow

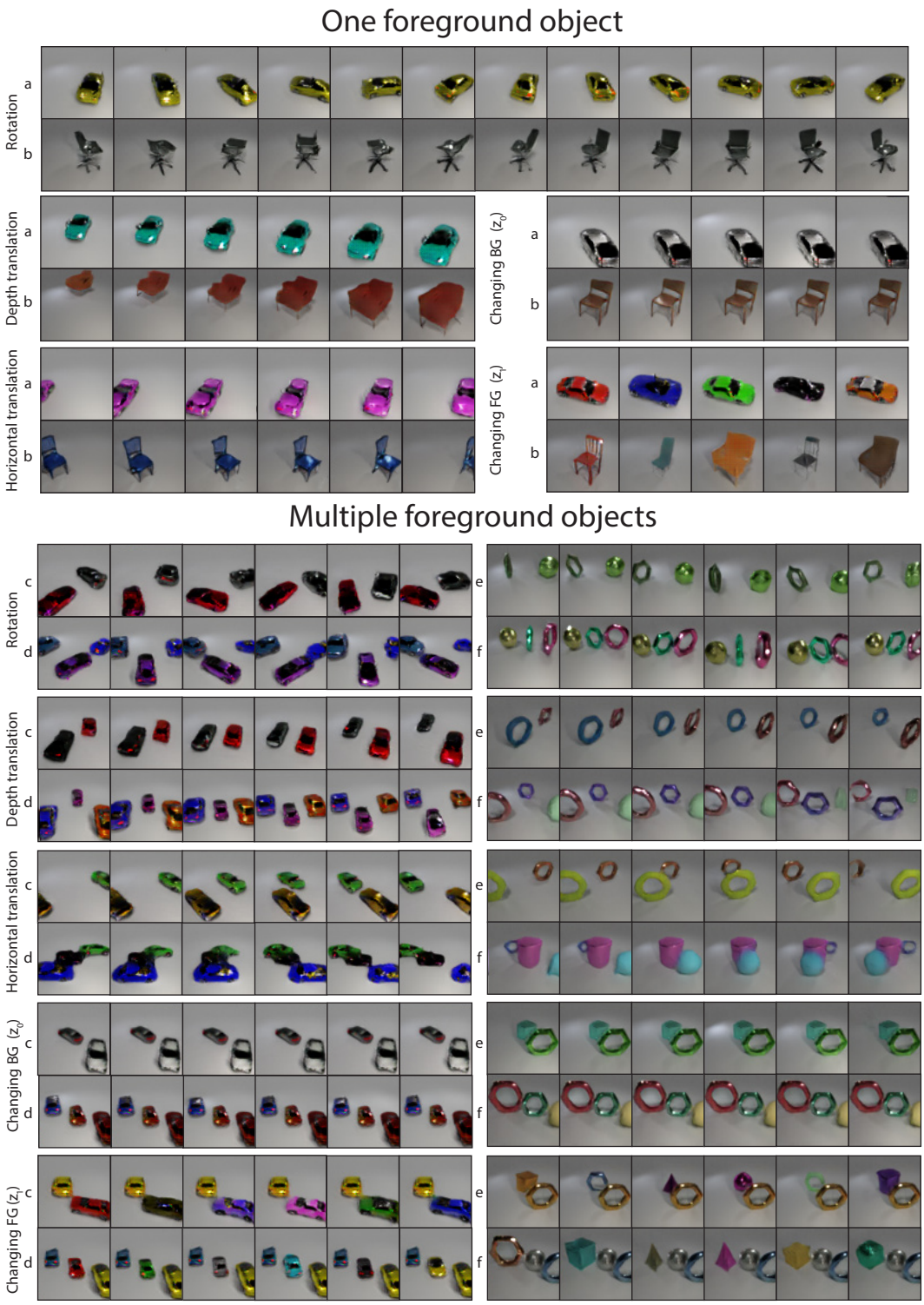

Figure 4: BlockGAN enables explicit spatial manipulation of individual objects (rotation, translation) and changing the identity of background or foreground objects across different datasets: (a) SYNTH-CAR1, (b) SYNTH-CHAIR1, (c) SYNTH-CAR2, (d) SYNTH-CAR3, (e) CLEVR2 and (f) CLEVR3. Notice how the shadows and highlights change as objects move around in the scene, and how changing the background lighting affects the appearance of foreground objects. Figure 5 shows similar results on natural images. Please refer to the supplemental material for animated results.

hyperparameters and then compute the KID for 10,000 images generated by each model (samples by all methods are included in the supplementary material). Table 1 shows that BlockGAN generates images with competitive or better visual fidelity than other methods.

Table 1: KID estimates (mean ± std), lower is better, between real images and images generated by BlockGAN and other GANs. BlockGAN achieves competitive KID scores while providing control of each object in the generated images (which is not measured by KID).

Method	SYNTH-CAR1 64×64	SYNTH-CHAIR1 64×64	REAL-CAR 64×64	CLEVR2 64×64
WGAN-GP [17]	0.141 ± 0.002	0.111 ± 0.002	0.035 ± 0.001	0.076 ± 0.002
LR-GAN [54]	$\mathbf{0.038 \pm 0.001}$	0.036 ± 0.002	$\mathbf{0.014 \pm 0.001}$	0.052 ± 0.001
HoloGAN [39]	0.070 ± 0.001	0.058 ± 0.002	0.028 ± 0.002	0.032 ± 0.001
BlockGAN (ours)	0.039 ± 0.001	$\mathbf{0.031 \pm 0.001}$	0.016 ± 0.001	$\mathbf{0.021 \pm 0.001}$

4.3. Scene manipulation beyond training data. We show that at test time, 3D object features learnt by BlockGAN can be realistically manipulated in ways that have not been observed during training time. First, we show that the learnt 3D object features can also be reused to add more objects to the scene at test time, thanks to the compositionality inductive bias and our choice of scene composer function. Here we use BlockGAN trained on datasets with only *one* foreground object and *one* background, and show that more foreground objects of the same category can be added to the same scene at *test time*. Figure 5 shows that 2–4 new objects are added and manipulated just like the original objects while maintaining realistic shadows and highlights. Second, we apply spatial manipulations that were not part of the similarity transform used during training, such as horizontal stretching, or slicing and combining different foreground objects. Figure 5 shows that object features can be geometrically modified intuitively, without needing explicit 3D geometry or multi-view supervision during training.

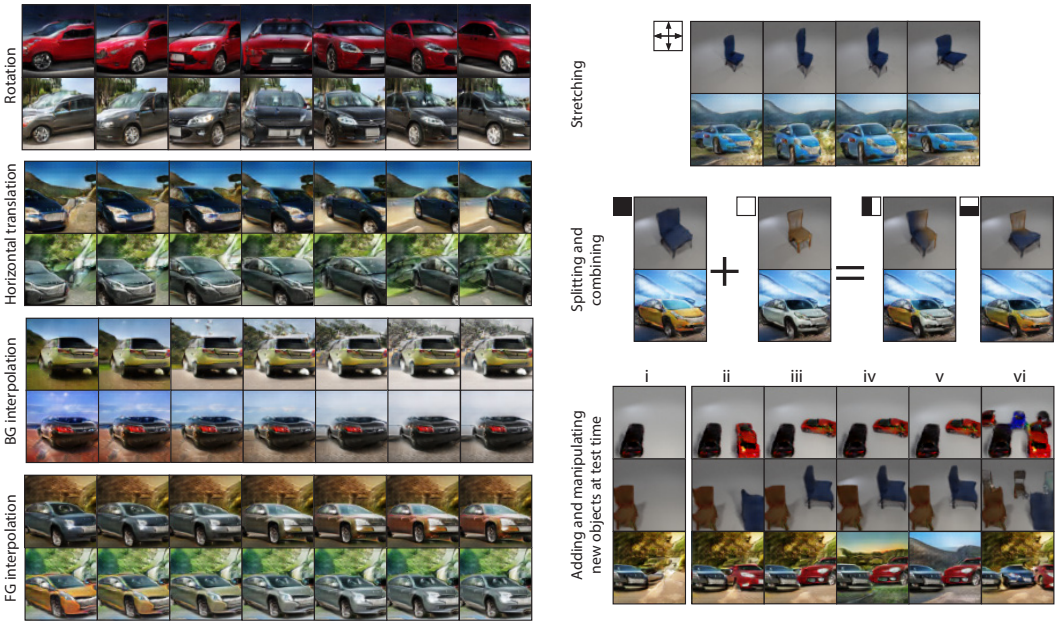

Figure 5: **Left:** REAL-CAR. Even for natural images with cluttered backgrounds, BlockGAN can still disentangle objects in a scene well. Note that interpolating the background ($\mathbf{z}_0$) affects the appearance of the car in a meaningful way, showing the benefit of 3D scene features. **Right:** Test-time geometric modification of the learnt 3D object features (unless stated, background is fixed): splitting and combining (top), stretching (middle), and adding and manipulating new objects after training (bottom). Bottom row shows (i) Original scene, (ii) new object added, (iii) manipulated, (iv,v) different background appearance, and (vi) more objects added. Note the realistic lighting and shadows.

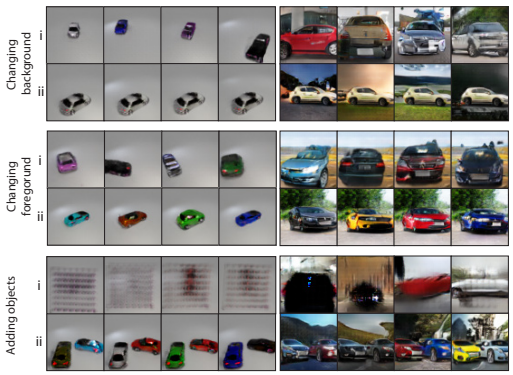

Figure 6: Comparison between (i) LR-GAN [54] and (ii) BlockGAN for SYNTH-CAR1 (left) and REAL-CAR (right).

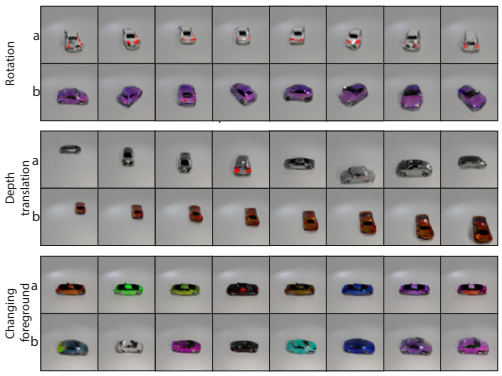

Figure 7: Different manipulations applied to BlockGAN trained on (a) a dataset with imbalanced rotations, and (b) a balanced dataset.

4.4. Comparison to 2D-based LR-GAN. LR-GAN [54] first generates a 2D background layer, and then generates and combines foreground layers with the generated background using alpha-compositing. Both BlockGAN and LR-GAN show the importance of combining objects in a contextually relevant manner to generate visually realistic images (see Table 1). However, LR-GAN does not offer explicit control over object location. More importantly, LR-GAN learns an *entangled* representation of the scene: sampling a different background noise vector also changes the foreground (Figure 6). Finally, unlike BlockGAN, LR-GAN does not allow adding more foreground objects during test time. This demonstrates the benefits of learning disentangled *3D* object features compared to a 2D-based approach.

4.5. Ablation study: Non-uniform pose distribution. For the natural REAL-CAR dataset, we observe that BlockGAN has difficulties learning the full 360° rotation of the car, even though fore- and background are disentangled well. We hypothesise that this is due to the mismatch between the true (unknown) pose distribution of the car, and the uniform pose distribution we assume during training. To test this, we create a synthetic dataset similar to SYNTH-CAR1 with a limited range of rotation, and train BlockGAN with a *uniform* pose distribution (details in supplement). With the imbalanced dataset, Figure 7 (bottom) shows correct disentangling of foreground and background. However, rotation of the car only produces images with (near-)frontal views (top), while depth translation results in cars that are randomly rotated sideways (middle). We observe similar behaviour for the natural REAL-CAR dataset. This suggests that learning object disentanglement and full 3D pose rotation might be two independent problems. While assuming a uniform pose distribution already enables good object disentanglement, learning the pose distribution from the training data would likely improve the quality of 3D transforms.

In our supplemental material, we include comparisons to HoloGAN [39] as well as additional ablation studies on comparing different scene composer functions, using a perspective camera versus a weak-perspective camera, and adopting the style discriminator for scenes with cluttered backgrounds.

5 Discussion and Future Work

We introduced BlockGAN, an image generative model that learns 3D object-aware scene representations from unlabelled images. We show that BlockGAN can learn a disentangled scene representation both in terms of objects and their properties, which allows geometric manipulations not observed during training. Most excitingly, even when BlockGAN is trained with fewer or even single objects, additional 3D object features can be added to the scene features at test time to create novel scenes with multiple objects. In addition to computer graphics applications, this opens up exciting possibilities, such as combining BlockGAN with models like BiGAN [12] or ALI [13] to learn powerful object representations for scene understanding and reasoning.

Future work can adopt more powerful relational learning models [28, 46, 52] to learn more complex object interactions such as inter-object shadowing or reflections. Currently, we assume prior knowledge of object category and the number of objects for training. We also assume object poses are uniformly distributed and independent from each other. Therefore, the ability to learn this information directly

from training images would allow BlockGAN to be applied to more complex datasets with a varying number of objects and different object categories, such as COCO [33] or LSUN [57].

Acknowledgments and Disclosure of Funding

We thank Lucas Theis for helpful discussions and feedbacks. We received support from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 665992, the EPSRC Centre for Doctoral Training in Digital Entertainment (EP/L016540/1), RCUK grant CAMERA (EP/M023281/1), an EPSRC-UKRI Innovation Fellowship (EP/S001050/1), and an NVIDIA Corporation GPU Grant. We received a gift from Adobe.

Broader Impact

BlockGAN is an image generative model that learns an object-oriented 3D scene representation directly from unlabelled 2D images. Our approach is a new machine learning technique that makes it possible to generate unseen images from a noise vector, with unprecedented control over the identity and pose of multiple independent objects as well as the background. In the long term, our approach could enable powerful tools for digital artists that facilitate artistic control over realistic procedurally generated digital content.

At training time, our network performs a task somewhat akin to scene understanding, as our approach learns to disentangle between multiple objects and individual object properties (specifically their pose and identity). At test time, our approach enables sampling new images with control over pose and identity for each object in the scene, but does not directly take any image input. However, it is possible to embed images into the latent space of generative models [1]. A highly realistic generative image model and a good image fit would then make it possible to approximate the input image and, more importantly, to edit the individual objects in a pictured scene. Similar to existing image editing software, this enables a wide range of creative applications, but also potential ill-intended image manipulations.

References

[1] Rameen Abdal, Yipeng Qin, and Peter Wonka. Image2StyleGAN: How to embed images into the StyleGAN latent space? In *ICCV*, 2019.

[2] Kara-Ali Aliev, Artem Sevastopolsky, Maria Kolos, Dmitry Ulyanov, and Victor Lempitsky. Neural point-based graphics. arXiv:1906.08240, 2019.

[3] Titas Anciukevičius, Christoph Lampert, and Paul Henderson. Object-centric image generation with factored depths, locations, and appearances. *arXiv:2004.00642*, 2020.

[4] Adam Bielski and Paolo Favaro. Emergence of object segmentation in perturbed generative models. In *NeurIPS*, 2019.

[5] Mikołaj Bińkowski, Dougal J. Sutherland, Michael Arbel, and Arthur Gretton. Demystifying MMD GANs. In *ICLR*, 2018.

[6] Blender Online Community. *Blender – a 3D modelling and rendering package*. Blender Foundation, 2020. URL https://www.blender.org/.

[7] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *ICLR*, 2019.

[8] Mickaël Chen, Thierry Artières, and Ludovic Denoyer. Unsupervised object segmentation by redrawing. In *NeurIPS*, 2019.

[9] Ting Chen, Mario Lucic, Neil Houlsby, and Sylvain Gelly. On self modulation for generative adversarial networks. In *ICLR*, 2019.

[10] Wenzheng Chen, Huan Ling, Jun Gao, Edward Smith, Jaakko Lehtinen, Alec Jacobson, and Sanja Fidler. Learning to predict 3D objects with an interpolation-based differentiable renderer. In *NeurIPS*, 2019.

[11] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. In *NIPS*, 2016.

[12] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. In *ICLR*, 2017.

[13] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Alex Lamb, Martín Arjovsky, Olivier Mastropietro, and Aaron C. Courville. Adversarially learned inference. In *ICLR*, 2017.

[14] Martin Engelcke, Adam R. Kosiorek, Oiwi Parker Jones, and Ingmar Posner. GENESIS: Generative scene inference and sampling with object-centric latent representations. In *ICLR*, 2020.

[15] S. M. Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Koray Kavukcuoglu, and Geoffrey E Hinton. Attend, infer, repeat: Fast scene understanding with generative models. In *NIPS*, 2016.

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

[17] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C. Courville. Improved training of Wasserstein GANs. In *NIPS*, 2017.

[18] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004. ISBN 0521540518. doi: 10.1017/CBO9780511811685.

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

[20] Tobias Hinz, Stefan Heinrich, and Stefan Wermter. Generating multiple objects at spatially distinct locations. In *ICLR*, 2019.

[21] Xun Huang and Serge Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. In *ICCV*, 2017.

[22] Insu Jeon, Wonkwang Lee, and Gunhee Kim. IB-GAN: Disentangled representation learning with information bottleneck GAN. https://openreview.net/forum?id=ryljV2A5KX, 2018.

[23] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *CVPR*, 2017.

[24] Justin Johnson, Agrim Gupta, and Li Fei-Fei. Image generation from scene graphs. In *CVPR*, 2018.

[25] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. In *ICLR*, 2018.

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

[27] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[28] Thomas Kipf, Elise van der Pol, and Max Welling. Contrastive learning of structured world models. In *ICLR*, 2020.

[29] Adam Kosiorek, Hyunjik Kim, Yee Whye Teh, and Ingmar Posner. Sequential attend, infer, repeat: Generative modelling of moving objects. In *NeurIPS*, 2018.

[30] Jean Kossaifi, Linh Tran, Yannis Panagakis, and Maja Pantic. GAGAN: Geometry-aware generative adverserial networks. In *CVPR*, 2018.

[31] Hanock Kwak and Byoung-Tak Zhang. Generating images part by part with composite generative adversarial networks. arXiv:1607.05387, 2016.

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

[33] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. In *ECCV*, 2014.

[34] Shichen Liu, Tianye Li, Weikai Chen, and Hao Li. Soft rasterizer: A differentiable renderer for image-based 3D reasoning. In *ICCV*, 2019.

[35] Matthew M. Loper and Michael J. Black. OpenDR: An approximate differentiable renderer. In *ECCV*, 2014.

[36] Stephen Marschner, Peter Shirley, Michael Ashikhmin, Michael Gleicher, Naty Hoffman, Garrett Johnson, Tamara Munzner, Erik Reinhard, William B. Thompson, Peter Willemsen, and Brian Wyvill. *Fundamentals of Computer Graphics*. A K Peters/CRC Press, 4th edition, 2015.

[37] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. arXiv:1411.1784, 2014.

[38] Thu Nguyen-Phuoc, Chuan Li, Stephen Balaban, and Yongliang Yang. RenderNet: A deep convolutional network for differentiable rendering from 3D shapes. In *NeurIPS*, 2018.

[39] Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. HoloGAN: Unsupervised learning of 3D representations from natural images. In *ICCV*, 2019.

[40] Kyle Olszewski, Sergey Tulyakov, Oliver Woodford, Hao Li, and Linjie Luo. Transformable bottleneck networks. *ICCV*, 2019.

[41] Dim P. Papadopoulos, Youssef Tamaazousti, Ferda Ofli, Ingmar Weber, and Antonio Torralba. How to make a pizza: Learning a compositional layer-based GAN model. In *CVPR*, 2019.

[42] Eunbyung Park, Jimei Yang, Ersin Yumer, Duygu Ceylan, and Alexander C. Berg. Transformation-grounded image generation network for novel 3D view synthesis. In *CVPR*, 2017.

[43] Scott E Reed, Zeynep Akata, Santosh Mohan, Samuel Tenka, Bernt Schiele, and Honglak Lee. Learning what and where to draw. In *NIPS*, 2016.

[44] Konstantinos Rematas and Vittorio Ferrari. Neural voxel renderer: Learning an accurate and controllable rendering tool. In *CVPR*, 2020.

[45] Helge Rhodin, Mathieu Salzmann, and Pascal Fua. Unsupervised geometry-aware representation for 3D human pose estimation. In *ECCV*, 2018.

[46] Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Timothy Lillicrap. A simple neural network module for relational reasoning. In *NIPS*, 2017.

[47] Zhixin Shu, Mihir Sahasrabudhe, Riza Alp Guler, Dimitris Samaras, Nikos Paragios, and Iasonas Kokkinos. Deforming autoencoders: Unsupervised disentangling of shape and appearance. In *ECCV*, 2018.

[48] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhöfer. DeepVoxels: Learning persistent 3D feature embeddings. In *CVPR*, 2019.

[49] Vincent Sitzmann, Michael Zollhöfer, and Gordon Wetzstein. Scene representation networks: Continuous 3D-structure-aware neural scene representations. In *NeurIPS*, 2019.

[50] Mehmet Özgür Türkoğlu, Luuk Spreeuwers, William Thong, and Berkay Kicanaoglu. A layer-based sequential framework for scene generation with GANs. In *AAAI*, 2019.

[51] Sjoerd van Steenkiste, Karol Kurach, and Sylvain Gelly. A case for object compositionality in deep generative models of images. In *NeurIPS Workshops*, 2018.

[52] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NIPS*, 2017.

[53] Xinchen Yan, Jimei Yang, Ersin Yumer, Yijie Guo, and Honglak Lee. Perspective transformer nets: Learning single-view 3D object reconstruction without 3D supervision. In *NIPS*, 2016.

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

[55] Linjie Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. A large-scale car dataset for fine-grained categorization and verification. In *CVPR*, 2015.

[56] Yanchao Yang, Yutong Chen, and Stefano Soatto. Learning to manipulate individual objects in an image. *arXiv preprint arXiv:2004.05495*, 2020.

[57] Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. LSUN: Construction of a large-scale image dataset using deep learning with humans in the loop. arXiv:1506.03365, 2015.

[58] Jun-Yan Zhu, Zhoutong Zhang, Chengkai Zhang, Jiajun Wu, Antonio Torralba, Josh Tenenbaum, and Bill Freeman. Visual object networks: Image generation with disentangled 3D representations. In *NeurIPS*, 2018.

A Additional results

A.1. Comparison to *entangled* 3D scene representation. We compare BlockGAN with HoloGAN [39], which also learns deep 3D scene features but does not consider object disentanglement. In particular, HoloGAN only considers one noise vector $\mathbf{z}$ for identity and one pose $\boldsymbol{\theta}$ for the entire scene, and does not consider translation $\mathbf{t}$ as part of $\boldsymbol{\theta}$. While HoloGAN works well with object-centred scenes, it struggles with moving foreground objects. Figure 8 shows that HoloGAN tends to associate each pose $\boldsymbol{\theta}$ with a fixed object's identity (i.e., moving objects erroneously changes identity of both foreground and background), while changing $\mathbf{z}$ only changes a small part of the background. BlockGAN, on the other hand, can separate identity and pose for *each* object, while being able to learn scene-level effects such as lighting and shadows.

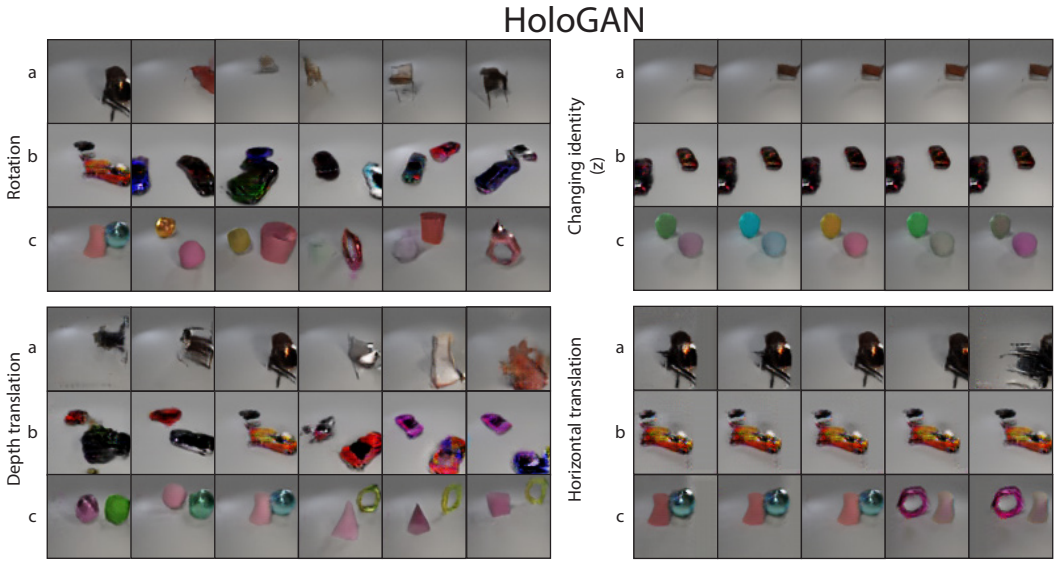

Figure 8: Samples from HoloGAN [39] trained on the datasets (a) SYNTH-CHAIR1, (b) SYNTH-CAR2 and (c) CLEVR2. HoloGAN tends to associate each pose $\boldsymbol{\theta}$ with a fixed object's identity, i.e., moving objects erroneously changes identity of both foreground and background (see top left, bottom left and right), while changing the noise vector $\mathbf{z}$ only changes a small part of the background (top right).

A.2. Increasing the number of foreground objects. To test the capability of our method, we train BlockGAN on CLEVR6 (6 foreground objects). As shown in Figure 9, BlockGAN is still capable of generating and manipulating 3D object features, although the background generator now also produces foreground objects (Figure 9f). Moreover, rotating individual object leads to changes in object's depth (Figure 9a).

Interestingly, we notice that BlockGAN now generates images with more or less than 6 objects, despite being trained with images that contain exactly 6 objects (Figure 9g). We hypothesise that BlockGAN's failure in this case is due to our assumption that the poses of all objects are independent from each other (during training, we randomly sample the pose θ for each object). This is not true in the physical world (also in the CLEVR dataset) where objects do not intersect. The more objects there are in the scene, the stronger the interdependence between objects' poses becomes. Therefore, for future work, we hope to adopt more powerful relation learning structures to learn objects' pose directly from training images. Another interesting direction is to design object-aware discriminators, which are capable of recognising fake images when the generators produce samples with more objects than the training images.

B Additional ablation studies

B.1. Learning without the perspective camera. Here we show the advantage of implementing the perspective camera explicitly, compared to using a weak-perspective projection like HoloGAN [39]. Since a perspective camera directly affects foreshortening, it provides strong cues for BlockGAN to

solve the scale/depth ambiguity. This is especially important for BlockGAN to learn to project and reason over occlusion by concatenating the depth and channel dimension, followed by an MLP. Since the MLP is flexible, BlockGAN trained *without* a perspective camera, therefore, tends to learn to associate an object's identity with scale and depth, while changing depth only changes the object's appearance (see Figure 10).

B.2. Scene composer function. We consider and compare three scene composer functions: (i) element-wise summation, (ii) element-wise maximum, and (iii) an MLP (multi-layer perceptron). We train BlockGAN with each function and compare their performance in terms of visual quality (KID score) in Table 2. While all three functions can successfully combine objects into a scene, the

Figure 9: Qualitative results of BlockGAN trained on CLEVR6. (a) Rotating a foreground object, (b) Horizontal translation, (c) Depth translation, (d) Changing foreground object 1, (e) Changing foreground object 3, (f) Changing the background object, and (g) Random samples.

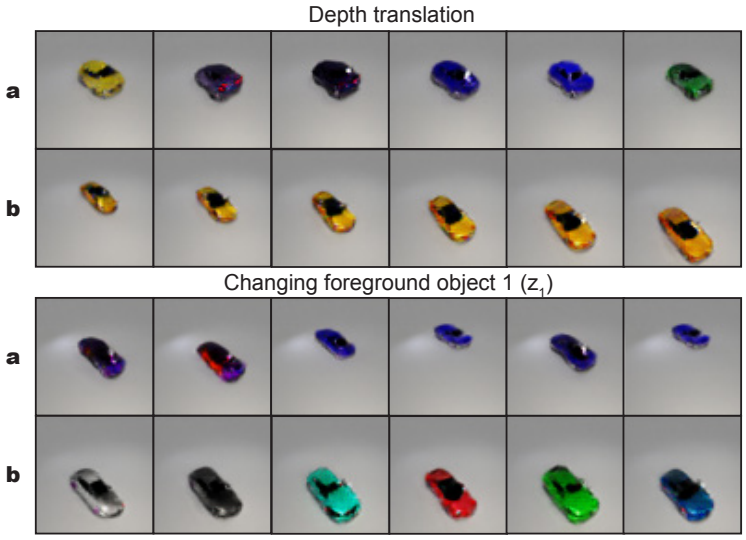

Figure 10: The effect of modelling the perspective camera explicitly (b) compared to using a weak-perspective camera (a). Note that with the weak-perspective camera (a), translation along the depth dimension (top) leads to identity changes without any translation in depth, while changing the noise vector $\mathbf{z}_1$ (bottom) changes both depth translation and, to a lesser extent, the object identity. Using a perspective camera correctly disentangles position and identity (b).

element-wise maximum performs best and easily generalises to multiple objects. Therefore, we use the element-wise maximum for BlockGAN.

Table 2: KID estimates for different scene composer functions.

Method	SYNTH-CAR1 (64×64)	SYNTH-CHAIR1 (64×64)
Sum	0.040 ± 0.002	0.038 ± 0.001
MLP	0.044 ± 0.001	0.033 ± 0.001
Max	$\mathbf{0.039 \pm 0.002}$	$\mathbf{0.031 \pm 0.001}$

B.3. Learning without the style discriminator. When BlockGAN is trained with a standard discriminator on datasets with a cluttered background, such as the REAL-CAR dataset, the foreground object features tend to include part of the background object. This creates visual artefacts when objects move in the scene (indicated by red arrows in Figure 11a). We hypothesise that these artefacts should be picked up by the discriminator since generated images should look unrealistic. Therefore, we add more powerful style discriminators [39] to the original discriminator at different layers (see Section D for details). Figure 11b shows that the generator is indeed discouraged from adding background information to the foreground object features, leading to cleaner results.

Figure 11: With a standard discriminator (a), a part of the background appearance is baked into the foreground object (see red arrows). Adding the style discriminator (b) cleanly separates the car from the background.

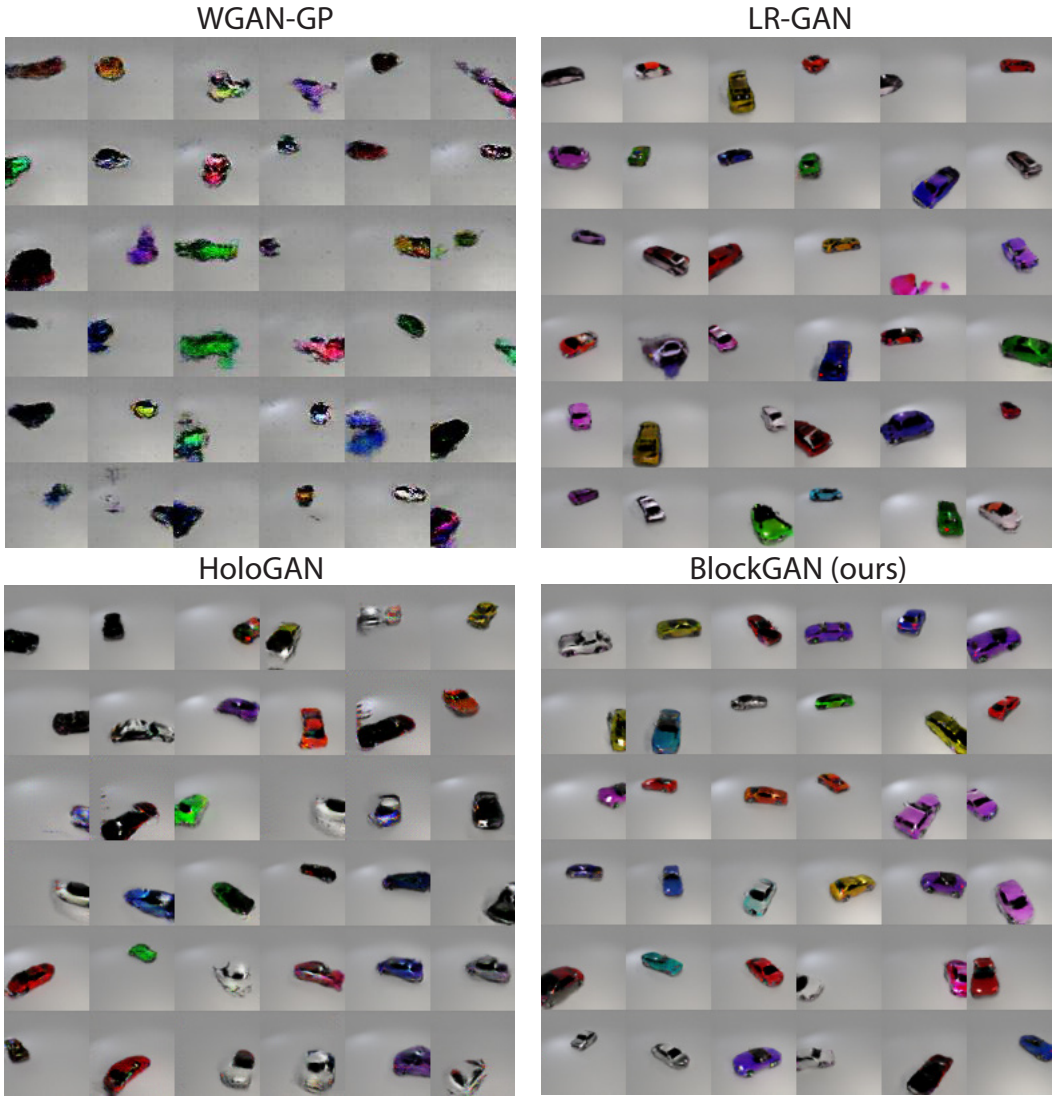

Figure 12: Samples from WGAN-GP, LR-GAN, HoloGAN and our BlockGAN trained on SYNTHCAR1.

C Comparison to other methods

In Figure 12, 13, 14 and 15, we show generated samples by a vanilla GAN (WGAN-GP [17]), 2D object-aware LR-GAN [54], 3D-aware HoloGAN [39] and our BlockGAN. Compared to other models, BlockGAN produces samples with competitive or better quality, *and* offers explicit control over the poses of objects in the generated images. Notice that although LR-GAN is designed to handle foreground and background objects explicitly, for CLEVR2 with two foreground objects, this method struggles and tends to always place one foreground object at the image centre (see Figure 14).

Implementation details For WGAN-GP, we use a publicly available implementation[2]. For LR-GAN and HoloGAN, we use the code provided by the authors. We conduct hyperparameter search for these models, and report best results for each method. Note that for HoloGAN, we modify the 3D transformation to add translation during training, since this method assumes that foreground objects are at the image centre.

[2] https://github.com/LynnHo/DCGAN-LSGAN-WGAN-WGAN-GP-Tensorflow

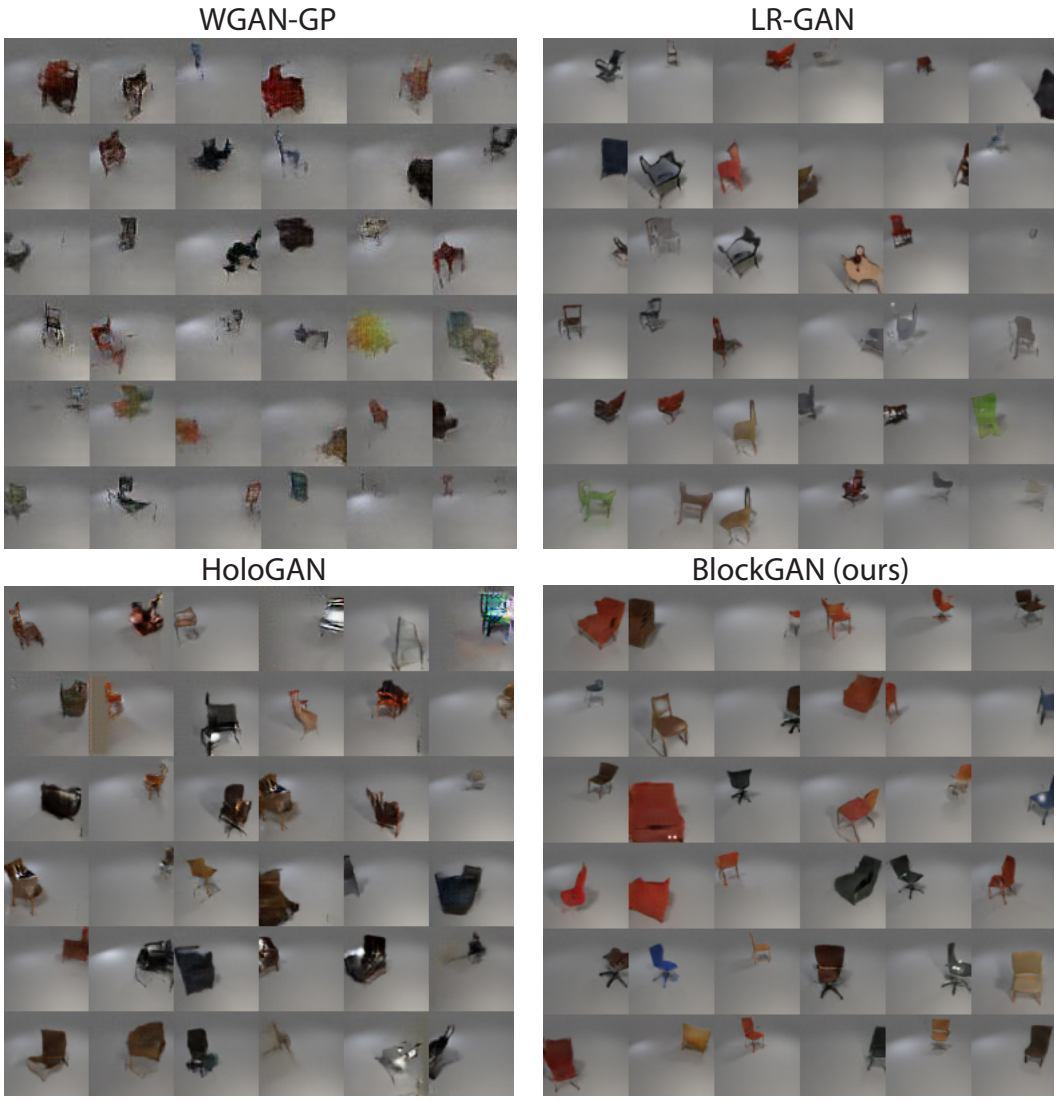

Figure 13: Samples from WGAN-GP, LR-GAN, HoloGAN and our BlockGAN trained on SYNTH-CHAIR1.

Figure 14: Samples from WGAN-GP, LR-GAN, HoloGAN and our BlockGAN trained on CLEVR2.

Figure 15: Samples from WGAN-GP, LR-GAN, HoloGAN and our BlockGAN trained on REAL-CARS.

D Loss function and style discriminator

For datasets with cluttered backgrounds like the natural REAL-CAR dataset, we adopt *style discriminators* in addition to the normal image discriminator (see the benefit in Figure 11). Style discriminators perform the same real/fake classification task as the standard image discriminator, but at the feature level across different layers. In particular, style discriminators classify the mean $\boldsymbol{\mu}$ and standard deviation $\boldsymbol{\sigma}$ of the features $\boldsymbol{\Phi}_l$ at different levels l (which are believed to describe the image "style"). The mean $\boldsymbol{\mu}(\boldsymbol{\Phi}_l(\mathbf{x}))$ and variance $\boldsymbol{\sigma}(\boldsymbol{\Phi}_l(\mathbf{x}))$ of the features $\boldsymbol{\Phi}_l(\mathbf{x})$ are computed across batch and spatial dimensions independently using:

$$\boldsymbol{\mu}(\boldsymbol{\Phi}_l(\mathbf{x})) = \frac{1}{N\times H\times W}\sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{w=1}^{W}\boldsymbol{\Phi}_l(\mathbf{x})_{nhw}, \tag{2}$$

$$\boldsymbol{\sigma}(\boldsymbol{\Phi}_l(\mathbf{x})) = \sqrt{\frac{1}{N\times H\times W}\sum_{n=1}^{N}\sum_{h=1}^{H}\sum_{w=1}^{W}\left(\boldsymbol{\Phi}_l(\mathbf{x})_{nhw}-\boldsymbol{\mu}(\boldsymbol{\Phi}_l(\mathbf{x}))\right)^2+\epsilon}. \tag{3}$$

The style discriminators are implemented as MLPs with sigmoid activation functions for binary classification. A style discriminator at layer l is written as

$$L^l_{\text{style}}(\mathrm{G}) = \mathbb{E}_{\mathbf{z},\theta}[-\log \mathrm{D}_l(\mathrm{G}(\mathbf{z},\theta))]. \tag{4}$$

The total loss therefore can be written as

$$L_{\text{total}}(\mathrm{G}) = L_{\text{GAN}}(\mathrm{G}) + \lambda_{\text{s}}\cdot\sum_l L^l_{\text{style}}(\mathrm{G}). \tag{5}$$

We set $\lambda_{\text{s}}=1$ for all natural datasets and $\lambda_{\text{s}}=0$ for synthetic datasets.

E Datasets

We modify the CLEVR dataset [23] to add a larger variety of colours and primitive shapes. Additionally, we use the scene setups provided by CLEVR to render the remaining synthetic datasets (SYNTH-CARn and SYNTH-CHAIRn, with n foreground objects each). These include a fixed, grey background, a virtual camera with fixed parameters but random location jittering, and random lighting. We also use the render script from CLEVR to randomly place foreground objects into the scene and render them. We render all image at resolution 128×128, and bi-linearly downsample them to 64×64 for training. For the natural CAR dataset, each image is first scaled such that the smaller side is 64, then it is cropped to produce a 64×64 pixel crop. During training, we randomly move the 64×64 cropping window before cropping the image. Figure 16 includes samples from our generated datasets, and Table 3 lists the range of pose parameters used for each dataset during training.

Link for 3D textured chair models:
https://keunhong.com/publications/photoshape/

Link for CLEVR:
https://github.com/facebookresearch/clevr-dataset-gen

Link for natural CAR dataset:
http://mmlab.ie.cuhk.edu.hk/datasets/comp_cars/

Table 3: Datasets used in our paper (n = number of foreground objects). 'Azimuth' describes object rotation about the up-axis. 'Elevation' refers to the camera's elevation above ground. 'Scaling' is the scale factor applied to foreground objects. 'Horiz. transl.' and 'Depth transl.' are horizontal/depth translation of objects relative to the global origin. Ranges represent uniform random distributions.

Name	# Images	Azimuth	Elevation	Scaling	Horiz. transl.	Depth transl.
SYNTH-CARn	80,000	0° – 359°	45°	0.5 – 0.6	–5 – 5	–5 – 5
SYNTH-CHAIRn	100,000	0° – 359°	45°	0.5 – 0.6	–5 – 5	–5 – 5
CLEVRn [23]	100,000	0° – 359°	45°	0.5 – 0.6	–4 – 4	–4 – 4
REAL-CARS [55]	139,714	0° – 359°	0° – 35°	0.5 – 0.8	–3 – 4	–5 – 6

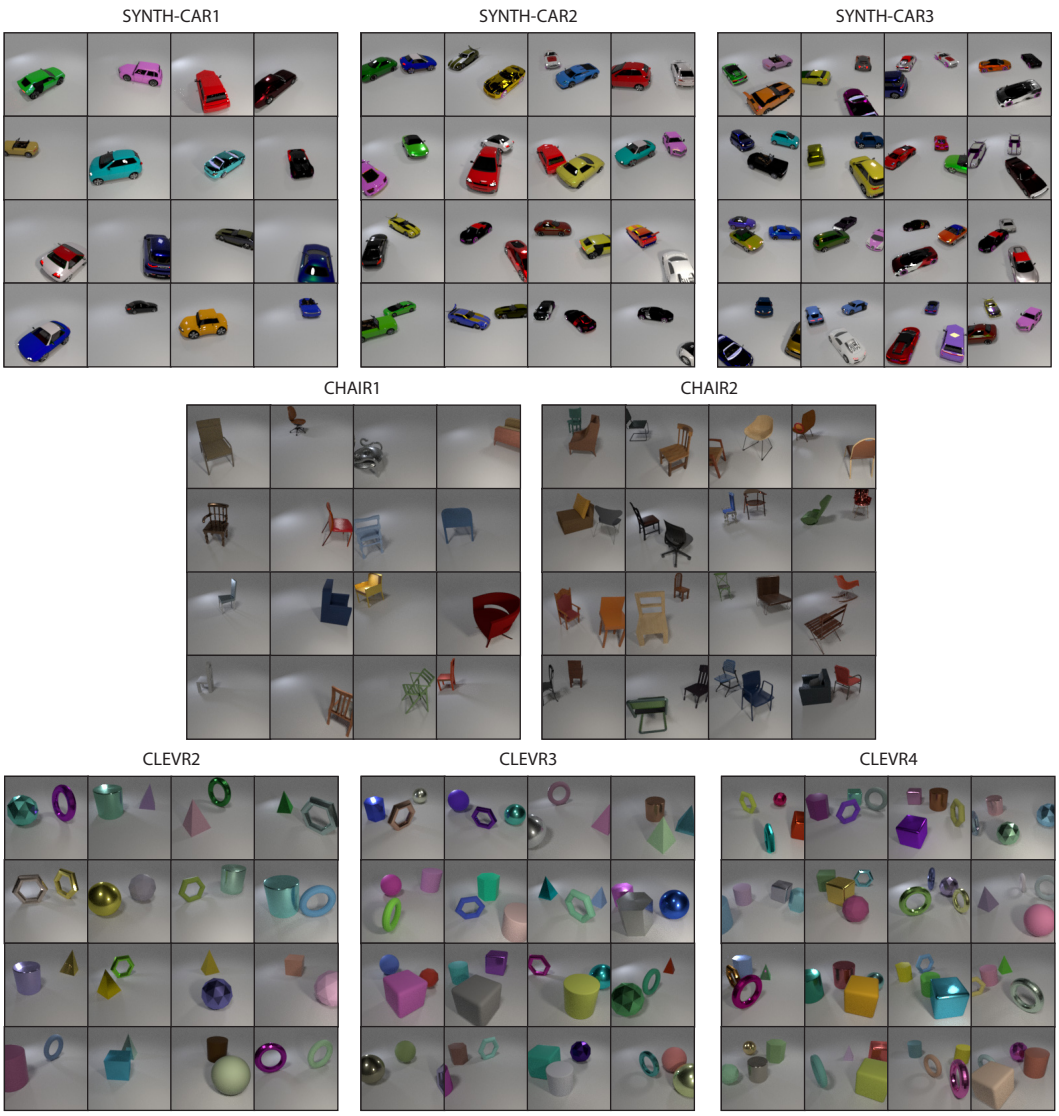

Figure 16: Samples from the synthetic datasets.

E.1. Additional details for the imbalanced rotation ablation study. To generate the imbalanced rotation dataset, we use the same general setup as SYNTH-CAR1. However, instead of sampling the car's rotation about the up-axis uniformly, we sample the rotation uniformly from the front/left/back/right viewing directions $\pm 15°$. In other words, the car is only seen from the front/left/back/right 30°, respectively, and there are four evenly spaced gaps of 60° that are never observed, for example views from the front-right.

F Implementation

F.1. Training details.

Virtual camera model. We assume a virtual camera with a focal length of 35 mm and a sensor size of 32 mm (Blender's default values), which corresponds to an angle of view of $2\arctan\frac{32\text{mm}}{2\times 35\text{mm}} = 49.1$ degrees (we use the same setup for natural images).

Sampling. We initialise all weights using $\mathcal{N}(0,0.2)$ and biases as 0. For CLEVRn, we use noise vector dimensions of $|\mathbf{z}_0| = 20$ for the background, and $|\mathbf{z}_i| = 60$ (for $i = 1,\ldots,n$) for the foreground

objects, to account for their relative visual complexity. Similarly, for SYNTH-CARn and SYNTH-CHAIRn, we use $|\mathbf{z}_0| = 30$ and $|\mathbf{z}_i| = 90$ (for $i = 1,...,n$), to account for their relative visual complexity. For the natural REAL-CAR dataset, we use $|\mathbf{z}_0| = 100$ and $|\mathbf{z}_1| = 200$. Note that we only feed $\mathbf{z}$ to the 3D features of each object, and not to the 3D scene features and 2D features. Table 3 provides the ranges we use for sampling the pose $\boldsymbol{\theta}_i$ of foreground objects during training.

Training. We train BlockGAN using the Adam optimiser [27], with $\beta_1 = 0.5$ and $\beta_2 = 0.999$. We use the same learning rate for both the discriminator and the generator. Empirically, we find that updating the generator twice for every update of the discriminator achieves images with the best visual fidelity. We use a learning rate of 0.0001 for all synthetic datasets. For the natural CARS dataset, we use a learning rate of 0.00005. We train all datasets with a batch size of 64 for 50 epochs. Training takes 1.5 days for the synthetic datasets and 3 days for the natural REAL-CARS dataset.

Infrastructure. All models were trained using a single GeForce RTX 2080 GPU.

F.2. Network architecture. We describe the network architecture for the BlockGAN foreground object generator in Table 4, the BlockGAN background generator in Table 5, and the overall BlockGAN generator in Tables 6 and 7 for synthetic and real datasets, respectively. Note that we use ReLU for the synthetic datasets and LReLU for the natural CAR dataset after the AdaIN layer. The discriminator is described in Table 8.

In terms of the notation in Section 3 of the main paper, object features have dimensions $H_o \times W_o \times D_o \times C_o = 16 \times 16 \times 16 \times 64$, scene features have the same dimensions $H_s \times W_s \times D_s \times C_s = 16 \times 16 \times 16 \times 64$, and camera features have dimensions $H_c \times W_c = 16 \times 16$ (before up-convolutions to 64×64) with $C_c = 64$ channels for synthetic datasets and $C_c = 256$ channels for natural image datasets.

As GANs empirically tend to perform better on category-specific datasets, we decided to start with this assumption. A promising future direction is to adopt a shared rendering layer for objects generated by different category-specific generators, similar to Aliev et al. [2].

Table 4: Network architecture of the BlockGAN foreground (FG) object generator.

Layer type	**Kernel size**	**Stride**	**Normalisation**	**Output dimension**
Learnt constant tensor	—	—	AdaIN	$4 \times 4 \times 4 \times 512$
UpConv	$3 \times 3 \times 3$	2	AdaIN	$8 \times 8 \times 8 \times 128$
UpConv	$3 \times 3 \times 3$	2	AdaIN	$16 \times 16 \times 16 \times 64$
3D transformation	—	—	—	$16 \times 16 \times 16 \times 64$

Table 5: Network architecture of the BlockGAN background (BG) object generator.

Layer type	**Kernel size**	**Stride**	**Normalisation**	**Output dimension**
Learnt constant tensor	—	—	AdaIN	$4 \times 4 \times 4 \times 256$
UpConv	$3 \times 3 \times 3$	2	AdaIN	$8 \times 8 \times 8 \times 128$
UpConv	$3 \times 3 \times 3$	2	AdaIN	$16 \times 16 \times 16 \times 64$
3D transformation	—	—	—	$16 \times 16 \times 16 \times 64$

Table 6: Network architecture of the BlockGAN generator for all synthetic datasets.

Layer type	Kernel size	Stride	Activation	Norm.	Output dimension
$n\times$FG generator (Table 4)	—	—	ReLU	—	$16\times16\times16\times64$
BG generator (Table 5)	—	—	ReLU	—	$16\times16\times16\times64$
Element-wise maximum	—	—	—	—	$16\times16\times16\times64$
Concatenate	—	—	—	—	$16\times16\times(16\cdot64)$
Conv	1×1	1	ReLU	—	$16\times16\times64$
UpConv	4×4	2	ReLU	AdaIN	$32\times32\times64$
UpConv	4×4	2	ReLU	AdaIN	$64\times64\times64$
UpConv	4×4	1	ReLU	AdaIN	$64\times64\times3$

Table 7: Network architecture of the BlockGAN generator for the REAL-CARS dataset. Differences to the synthetic foreground object generator in Table 6 are highlighted in blue.

Layer type	Kernel size	Stride	Activation	Normal.	Output dimension
FG generator (Table 4)	—	—	LReLU	—	$16\times16\times16\times64$
BG generator (Table 5)	—	—	LReLU	—	$16\times16\times16\times64$
Element-wise maximum	—	—	—	—	$16\times16\times16\times64$
Concatenate	—	—	—	—	$16\times16\times(16\cdot64)$
Conv	1×1	1	LReLU	—	$16\times16\times256$
UpConv	4×4	2	LReLU	AdaIN	$32\times32\times128$
UpConv	4×4	2	LReLU	AdaIN	$64\times64\times64$
UpConv	4×4	1	LReLU	AdaIN	$64\times64\times3$

Table 8: Network architecture of the BlockGAN discriminator for both synthetic and real datasets.

Layer type	Kernel size	Stride	Activation	Normalisation	Output dimension
Conv	5×5	2	LReLU	IN/Spectral	$32\times32\times64$
Conv	5×5	2	LReLU	IN/Spectral	$16\times16\times128$
Conv	5×5	2	LReLU	IN/Spectral	$8\times8\times256$
Conv	5×5	2	LReLU	IN/Spectral	$4\times4\times512$
Fully connected	—	—	Sigmoid	None/Spectral	1