Controlling Image Synthesis with Emergent and Designed Priors

by

Lucy Chai

B.S.E., University of Pennsylvania (2017)
M.Phil., University of Cambridge (2018)

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2023

Authored By: Lucy Chai
Department of Electrical Engineering and Computer Science
August 31, 2023

Certified By: Phillip Isola
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted By: Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee for Graduate Students

Controlling Image Synthesis with Emergent and Designed Priors

by

Lucy Chai

Submitted to the Department of Electrical Engineering and Computer Science
on August 31, 2023, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Image synthesis has developed at an unprecedented pace over the past few years, giving us new abilities to create synthetic yet photorealistic content. Typically, unconditional synthesis takes in a tensor of random numbers as input and produces a randomly generated image that mimics real-world content, with little to no way of controlling the result. The work contained in this thesis explores two avenues of obtaining controllable content from image generative models using emergent and designed priors. Emergent priors leverage the capabilities of a pre-trained generator to infer how the world operates, simply by training on large quantities of data. On the other hand, designed priors use built-in constraints to enforce desired properties about the world. Using emergent priors, we can control content by discovering factors of variation and compositional properties in the latent space of synthesis models. We further add coordinate information and camera inputs as designed controls to generate continuous-resolution and 3D-consistent imagery.

Acknowledgments

First and foremost, I'd like to thank my PhD advisor, Phillip Isola, for his constant support and guidance throughout this journey. I am forever grateful for his willingness to take me on as an inexperienced junior student transitioning into a new field. My early experiences working with Phil, helping me define a project and staying in the office helping me write until 3AM the night of my first deadline, were invaluable towards developing me as a researcher. Phil also helped to connect me with the broader computer vision community, paving the way for fruitful collaborations that I've had later on. I will always admire his excitement and optimism for research.

I would also like to thank my internship mentors, Richard Zhang and Noah Snavely, my collaborators at Adobe – Michaël Gharbi, Jun-Yan Zhu, and Eli Shechtman – and my collaborators at Google – Richard Tucker and Zhengqi Li. Under Richard's guidance I gained more experience in synthesis techniques, and I learned a lot about paper writing and presentation skills from him. Working with Noah's team helped to grow my experience in 3D vision, and I am thankful for their helpful insights in this area and their patient guidance throughout the project. I am so grateful to have the opportunity to work with both teams, and much of the work presented in this thesis would not have happened without these collaborators.

My first officemates and early collaborators David Bau and Jonas Wulff were instrumental in my development as a researcher. I am thankful for their guidance in finding my footing as a junior student, answering basic questions, sharing their tricks for experiment infrastructures, and of course their help as coauthors in my projects.

My other committee members Antonio Torralba and Fredo Durand are both my role models as successful senior researchers in the field, as well as instructors for some of my favorite lectures at MIT. I am honored to have them as part of my committee and grateful for their time and feedback on my thesis work.

I would like to acknowledge my other co-authors on various projects, including Jingwei Ma, Minyoung Huh, Tongzhou Wang, Ser-Nam Lim, Ali Jahanian, Stephanie Fu, Netanel Tamir, Shobhita Sundaram, and Tali Dekel. I have learned something new from working with each of them and glad we had the opportunity to work together.

Thanks to my labmates – Yonglong Tian, Yen-Chen Lin, Tongzhou Wang, Caroline Chan, Minyoung Huh, Joseph Suarez, Hyojin Bahng, Shobhita Sundaram, Akarsh Kumar, Swami Sankaranarayanan, Stephanie Fu, and Kevin Frans – for always fostering a supportive and friendly community. I am lucky that my labmates are also some of my closest friends, and I am grateful for the time we spent together both inside and outside of the lab on retreats, skiing, and unofficial social events. I also owe thanks to my friends from the MIT community and beyond – Irene Kuang, Monica Agrawal, Camille Biscarrat, Luke Anderson, Prafull Sharma, Wei-Chiu Ma, Ching-Yao Chuang, Shuang Li, Xavi Puig, Manel Baradad, Joanna Materzynska, Dave Epstein, Andrew Liu, Ben Kompa, Seth Musser, Anthony Tabet, Jesse Mu, and Daniel Rothchild – for their companionship and encouragement while sharing the various ups and downs of the PhD experience with me.

My family has been a constant source of support during this time and all the years beforehand. I am grateful to them for always investing in my education and encouraging me to pursue my goals. These opportunities would not be possible without the environment they provided for me. I would also like to thank my partner, Patrick, for always believing in me and encouraging me throughout this journey.

Finally, I'd like to acknowledge my funding sources that made these projects possible: the National Science Foundation Graduate Research Fellowship, Adobe Research Fellowship, and Meta PhD Fellowship.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Images are the predominant data format that we use to capture and represent our observations of the world. With modern cameras, pictures are effortless and instantaneous to take, allowing us to easily collect them in large quantities. In addition, further technological improvements in data storage and file sharing have led us to create and share these images as large datasets, consisting of thousands, millions, or even billions of observations ranging from images of a single domain to uncurated images scraped from the internet with various levels of annotation.

The image synthesis task aims to regenerate the world via the image datasets that we use for training. Each image from the dataset represents a sampled point from the entire world, and synthesis models aim to recreate the world from these discrete data samples. Notably, this involves generating images that look similar to, but are not exactly alike any given image in the training dataset, thus filling out the manifold of possible realistic images that we can observe in the world.

To generate new image samples, the basic mechanism in synthesis models is to learn a mapping from randomly sampled noise vectors to a structured image representation. Because the distribution of possible real images is difficult to sample from directly, the generation process initiates from a distribution that is easy to sample from, for instance a multivariate Gaussian with predefined dimensionality. This random input noise is what gives the generator the ability to produce diverse samples, such that different samples from the noise distribution result in different output images.

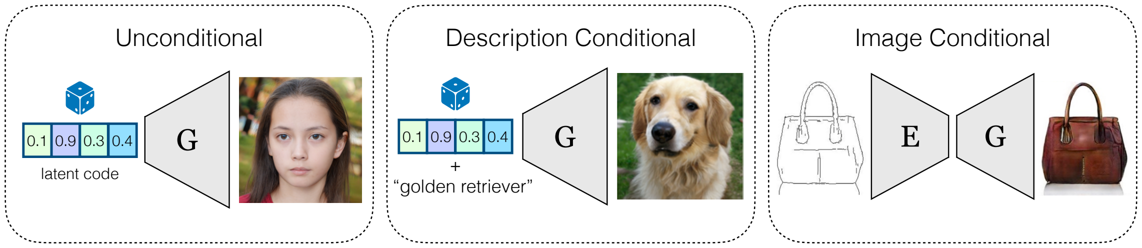


Figure 1-1: **Variations of image synthesis.** *(Left)* Unconditional synthesis relies entirely on the latent code, a randomly sampled noise vector, to generate the resulting image. This form of synthesis often works best when training on single-domain datasets. *(Center)* Additional conditioning information can be provided to the generator using descriptive embedding vectors representing class category or text descriptions. This additional annotated input helps expand the possible range of images that the generator can produce. *(Right)* Providing images as input to the generator conveys additional structural priors about the resulting output. Images are samples from [109, 19, 96].

1.1 Variations of the Synthesis Problem

While the key idea in synthesis is to map noise samples to images, there are several ways in which this goal can be realized via various training techniques. The basis for the works presented in this thesis rely on the Generative Adversarial Network (GAN) framework. First introduced by Goodfellow *et al.* [69], the GAN objective trains two competing networks – a generator and discriminator. These two networks are trained in a alternating fashion in which the generator network attempts to synthesize images and the discriminator network aims to distinguish the generator samples from real dataset images.

As the generator and discriminator networks must maintain a delicate balance for successful training, subsequent works have further developed the GAN training framework with techniques to improve synthesis quality [106, 107], increase scale [19, 200, 201, 104], stabilize training [153, 108], and evaluate the realism of the generated results [82, 196, 15].

The most basic variation of the synthesis task is unconditional synthesis, in which the only input to the generator network is the random noise latent vector (Fig. 1-1-left). Because the latent code distribution must be able to represent the entire range of possible training images, typically unconditional GANs have been most successful in domain-specific settings in which the training dataset consists of images from a single category such as aligned faces [144, 107] or specific type of scene [271]. Consequently the generator only produces images within the narrow range of the particular training domain.

For datasets with corresponding annotations, it is also possible to provide these labels as input into the generator as an embedding vector which provides coarse descriptive information about the resulting image (Fig. 1-1-center). Conditioning on class category information has helped scale up GAN training to the thousand categories of ImageNet [194, 19], and the rise of recent large-scale image and text paired datasets with developments in language modelling enable the usage of text descriptions as conditioning input [104, 201]. Compared to unconditional generation, additional forms of descriptive conditioning enable a single generator model to synthesize a broader range of possible images.

Rather than inputting descriptive conditioning as embedding vectors, an alternative form of the synthesis task takes images as input for the image-to-image translation task [96, 293, 171, 38] (Fig. 1-1-right). In these cases, the GAN objective is typically used as objective function to improve image quality, while the input image provides the local structure information that the desired output should contain.

In recent years, diffusion models based on iterative denoising operations have surged in popularity as an alternative to the GAN training framework [85, 86, 225, 226]. However, these same basic variations of image synthesis – unconditional, description-conditional, and image-conditional – are still relevant within the diffusion pipeline for various downstream applications.

1.2 Making Synthesis Controllable

Drawing samples from the latent code distribution allows us to create infinitely many random images, but it lacks any form of control over the generated content. However, oftentimes we have specific preferences about the content we want to create. In that case, how can we incorporate our desired criteria into the synthesis process, so that we can modify the generated images in predictable ways?

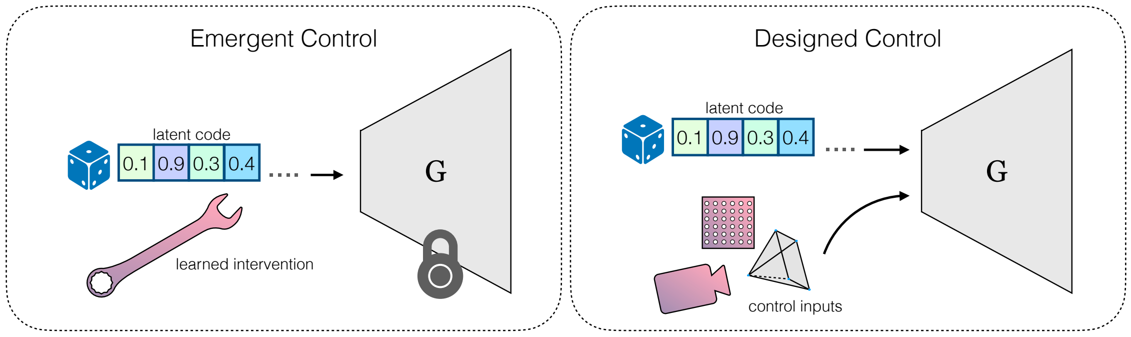


Figure 1-2: **Controlling image synthesis.** Random samples from the latent space offers limited control over the resulting generated image, while conditioning on descriptive embeddings offers a coarse level of control over the output. In this thesis, we explore additional strategies for controlling the synthesized content. *(Left)* Emergent control takes advantages of the image prior that a pretrained generator learns from vast amounts of data, and adjusts the latent code to achieve desired image variations. *(Right)* Designed control trains a model to obey additional conditioning input to the generator. While conditioning on descriptive embeddings is one form of designed control, here we design models that are incorporate spatial transformations for fine-grained positional control.

The works in this thesis present methods for adding *control* to the synthesis process. Rather than using generative models to create individual samples of static images, we expand the capabilities of image generation, allowing for targeted modifications and adjustments according to user criteria or desired outcomes. Controllable synthesis can be categorized under two general strategies: *emergent* control leverages learned properties from a pre-trained model, while *designed* control builds in desired constraints into the model training procedure.

1.2.1 Emergent Control

By training on large collections of images, synthesis models learn to infer certain characteristics about the world based on patterns in the training data. With emergent control, we start with a generator trained purely for the synthesis task, and use this pretrained generator as an image prior with baked-in knowledge about properties of realistic images. Then, we keep the synthesis model fixed and enable additional editing capabilities by learning interventions on the input latent codes to satisfy the desired criteria on the corresponding outputs. One advantage of this setup is that we can use approximate objective functions to adjust the generated content – the resulting output will balance between satisfying the learning objective while still producing a realistic image due to the priors from the frozen generator.

1.2.2 Designed Control

The limitation of emergent control is that it is constrained by what a generator is capable of creating and the rules of the world that the generator infers by training on large amounts of data. Rather than relying on the generator to infer these properties indirectly from data, we can provide our desired controls as input and encourage the model to obey these additional controls via auxiliary constraints while also training for the synthesis task. Class category, text description, and image conditioning are all forms of designed controls, in which the provided input tells us properties of how the desired output should look.

In contrast, the designed controls presented in this thesis will incorporate *spatial awareness* into the model by conditioning on coordinates. This form of control does not tell the model what content it should produce, but gives it knowledge about the location within a larger context where the generation process is occurring. As a result, we can break the synthesis procedure into spatial pieces, and navigate through continuous 2D space and 3D volumes by applying transformations on these input coordinates.

One point to note is that emergent control and designed control are not necessarily mutually exclusive. We can take models trained with designed controls, such as class-conditional generators, and discover elements of emergent control after model training. Alternatively, we can also leverage the emergent properties of large models as a prior and combine them with additional designed objectives to finetune them towards additional editing behaviors.

1.3 Outline

This works covered in this thesis explore methods for obtaining controllable content from synthesis models via emergent and designed techniques. Part I focuses on emergent control, in which we can generate image variations using the trained model as an image prior combined with subsequent learned adaptations to the latent code input. More specifically:

– Chapter 2 explores creating image variations via latent-space perturbations and self-supervised objectives. In particular, we study the abilities of a generative model to fit simple transformations such as camera movements and color changes. We find that the models reflect the biases of the datasets on which they are trained (e.g., centered objects), but also exhibit some capacity for generalization, and we further conduct experiments to quantify the limits of these transformations and introduce techniques to mitigate these limits.

– Chapter 3 investigates compositional properties in generated imagery, allowing us to interchange parts of different images using approximate image templates. We find that combining a trained regressor network and a pretrained generator provides a strong image prior, allowing us to create composite images from a collage of disjoint image parts while maintaining global consistency. Our regression approach enables more localized editing compared to direct editing in the latent space and also readily extends to a number of related applications, such as image inpainting or example-based image editing.

In Part II, we design our model to be aware of spatial layout using coordinate conditioning and geometric transformations during the training process.

– In Chapter 4, we describe continuous-scale training, a process that samples patches at random scales to train a new generator with variable output resolutions. This avoids the fixed reoslution training assumption of typical image synthesis, which resizes training images to a common size while discarding high resolution information. Conditioning the generator on a target scale allows us to generate higher resolution images than previously possible, without adding additional layers to the model. On various natural image domains, we demonstrate arbitrary scale synthesis with both coherent global layouts and realistic local details.

– Chapter 5 builds a model to synthesize unbounded nature scenery while enabling arbitrarily large camera motion and maintaining a persistent 3D world model. Our scene representation consists of an extendable, planar scene layout grid, which can be rendered from arbitrary camera poses via a 3D decoder and volume rendering, and a panoramic skydome. Based on this representation, we learn a generative world model solely from single-view internet photos. Our method enables simulating long flights through 3D landscapes, while maintaining global scene consistency—for instance, returning to a previously visited camera position yields the same view of the scene. Our approach enables scene extrapolation beyond the limited camera ranges of current 3D generative models, while also supporting a persistent, camera-independent world representation that stands in contrast to auto-regressive 3D prediction methods.

Part I

Emergent Control in Image Synthesis

Chapter 2

Natural Image Variations from Latent Space Manipulations

With minor modifications from:

On the "Steerability" of Generative Adversarial Networks

Ali Jahanian*, Lucy Chai*, Phillip Isola; ICLR 2020.

Using a pretrained image generator as an image prior, we start by exploring the possible image variations that the model can create. We define self-supervised objectives using simple image transformations, and optimize latent vectors to match these targets. The result is a balance between satisfying the given objective function, while still maintaining image realism. We quantify the model's ability to achieve these target transformations, dependent on the per-class variability present in the training dataset.

2.1 Introduction

Science fiction has long dreamed of virtual realities filled of synthetic content as rich as, or richer, than the real world (e.g., *The Matrix*, *Ready Player One*). While traditional computer graphics can render photorealistic 3D scenes, they lack the ability to automatically *synthesize* detailed content. Generative models like GANs,

*Equal Contribution

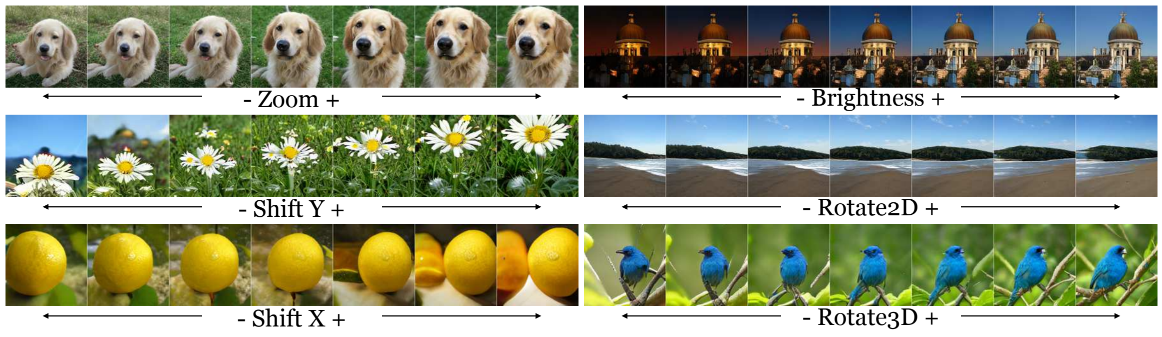


Figure 2-1: **Examples of learned latent space trajectories.** These learned manipulation vectors correspond to visual transformations like camera shift and zoom.

in contrast, can create content from scratch, but we do not currently have tools for navigating the generated scenes in the same kind of way as you can walk through and interact with a 3D game engine.

This chapter explores the degree to which you can navigate the visual world of a GAN. Figure 2-1 illustrates the kinds of transformations we explore. Consider the dog at the top-left. By moving in some direction of GAN latent space, can we hallucinate walking toward this dog? As the figure indicates, and as we will show, the answer is yes. However, as we continue to zoom in, we quickly reach limits. Once the dog face fills the full frame, continuing to walk in this direction fails to increase the zoom. A similar effect occurs in the daisy example (row 2 of Fig. 2-1), where a direction in latent space moves the daisy up and down, but cannot move it out of frame.

We hypothesize that these limits are due to biases in the distribution of images on which the GAN is trained. For example, if the training dataset consists of centered dogs and daises, the same may be the case in GAN-generated images. Nonetheless, we find that *some* degree of transformation is possible. When and why can we achieve certain transformations but not others?

In this chapter, we quantify the degree to which we can achieve basic visual transformations by navigating in GAN latent space. In other words, are GANs "steerable" in latent space?[*] We analyze the relationship between the data distribution on which the model is trained and the success in achieving these transformations. From

[*]We use the term "steerable" in analogy to the classic steerable filters of Freeman and Adelson [60].

our experiments, it is possible to shift the distribution of generated images to some degree, but we cannot extrapolate entirely out of the dataset's support. In particular, attributes can be shifted in proportion to the variability of that attribute in the training data. We further demonstrate an approach to increase model steerability by jointly optimizing the generator and latent direction, together with data augmentation on training images. One of the current criticisms of generative models is that they simply interpolate between datapoints, and fail to generate anything truly new, but our results add nuance to this story. It *is* possible to achieve distributional shift, but the ability to create realistic images from a modified distributions relies on sufficient diversity in the dataset along the dimension that we vary. Our main findings are:

- A simple walk in GAN latent space achieves camera motion and color transformations in the output image space. These walks are learned in self-supervised manner without labeled attributes or distinct source and target images.
- The linear walk is as effective as more complex non-linear walks, suggesting that the models learn to roughly linearize these operations without explicit training.
- We quantify a relationship between dataset variability and how much we can shift the model distribution.
- The transformations are a general-purpose framework that work with different model architectures, e.g. BigGAN, StyleGAN, and DCGAN, and illustrate different disentanglement properties in their respective latent spaces.
- Data augmentation and jointly training the walk trajectory and the generator weights allows us to achieve larger transformation effects.

2.2 Related Work

Interpolations in latent space. Traditional approaches to image editing with GAN latent spaces find linear directions that correspond to changes in labeled attributes, such as smile-vectors and gender-vectors for faces [182, 107]. However these manipulations are not exclusive to GANs; in flow-based generative models, linearly interpolating between two encoded images allow one to edit a source image toward attributes of the target [116]. Möllenhoff and Cremers [158] proposes a modified GAN formulation by treating data as directional k-currents, where moving along tangent planes naturally corresponds to interpretable manipulations. Upchurch et al. [243] removes the generative model entirely and instead interpolates in the intermediate feature space of a pretrained classifier, again using feature mappings of source and target sets to determine an edit direction. Unlike these approaches, we learn our latent-space trajectories in a self-supervised manner without labeled attributes or distinct source and target images. Instead, we learn to approximate editing operations on individual source images. We find that linear trajectories in latent space can capture simple image manipulations, e.g., zoom-vectors and shift-vectors, although we also obtain similar results using nonlinear trajectories.

Dataset bias. Biases from training data and network architecture both impact generalization capacity in learned models [237, 66, 4]. Dataset biases partly comes from human preferences in taking photos: we tend to take pictures in "canonical" views that are not fully representative of the entire visual world [154, 97]. Consequently, models inherit these biases. This may result in models that misrepresent the given task – such as tendencies towards texture bias rather than shape bias on ImageNet classifiers [66] – and in turn limits their generalization performance on similar objectives [7]. Our latent space trajectories transform the output corresponding to various image editing operations, but ultimately we are constrained by biases in the data and cannot extrapolate arbitrarily far beyond the data's support.

Generative models for content creation. The recent progress in generative models has opened interesting avenues for content creation [19, 107], including applications that enable users to fine-tune the generated output [217, 292, 10]. A by-product the current work is enable users to modify image properties by turning a single knob – the magnitude of the learned transformation in latent space. We further demonstrate that these image manipulations are not just a simple creativity tool; they also provide us with a window into biases and generalization capacity of these models.

Applications of latent space editing. Image manipulations using generative models suggest several interesting downstream applications. For example, Denton et al. [46] learns linear walks corresponding to various facial characteristics – they use these to measure biases in facial attribute detectors, whereas we study biases in the generative model that originate from training data. Shen et al. [210] also assumes linear latent space trajectories and learns paths for face attribute editing according to semantic concepts such as age and expression, thus demonstrating disentanglement of the latent space. White [259] suggests approaches to improve the learned manipulations, such as using spherical linear interpolations, resampling images to remove biases in attribute vectors, and using data augmentation as a synthetic attribute for variational autoencoders. Goetschalckx et al. [68] applies a linear walk to achieve transformations corresponding to cognitive properties of an image. Unlike these works, we do not require an attribute detector or assessor function to learn the latent space trajectory, and therefore our loss function is based on image similarity between source and target images. In addition to linear walks, we explore using non-linear walks parametrized by neural networks for editing operations.

2.3 Method

Generative models such as GANs [69] learn a mapping function G such that $G : z \to x$. Here, z is the latent code drawn from a Gaussian density and x is an output, e.g., an image. Our goal is to achieve transformations in the output space by moving in latent space, as shown in Fig. 2-2. In general, this goal also captures the idea in equivariance, in which transformations in the input space result in equivalent transformations in the output space (c.f. Hinton et al. [83], Cohen et al. [41], Lenc and Vedaldi [126]).

2.3.1 Objective

We want to learn an N-dimensional vector representing the optimal path in latent space for a given transformation. The vector is multiplied with continuous parameter

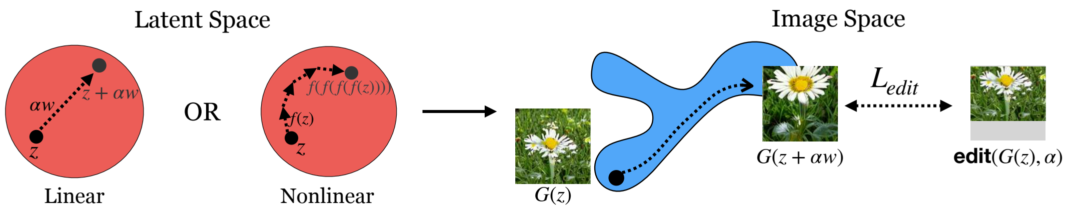


Figure 2-2: **Learning latent space manipulations.** We aim to find a path in z space to transform the generated image $G(z)$ to its edited version $\mathtt{edit}(G(z,\alpha))$, e.g., an $\alpha\times$ zoom. This walk results in the generated image $G(z+\alpha w)$ when we choose a linear walk, or $G(f(f(...(z)))$ when we choose a non-linear walk.

α which signifies the step size: large α values correspond to a greater degree of transformation, while small α values correspond to a lesser degree. Formally, we learn the walk w by minimizing the objective function:

$$w^* = \arg\min_w \mathbb{E}_{z,\alpha}[\mathcal{L}(G(z+\alpha w), \mathtt{edit}(G(z), \alpha))]. \tag{2.1}$$

Here, $\mathcal{L}$ measures the distance between the generated image after taking an α-step in the latent direction $G(z+\alpha w)$ and the target $\mathtt{edit}(G(z),\ \alpha)$ derived from the source image $G(z)$. We use $L2$ loss as our objective $\mathcal{L}$, however we also obtain similar results when using the LPIPS perceptual image similarity metric [280] (see Appendix A.2.4). Note that we can learn this walk in a fully self-supervised manner – we perform the $\mathtt{edit}(\cdot)$ operation on an arbitrary generated image and subsequently the vector to minimize the objective. Let $\mathtt{model(}\alpha\mathtt{)}$ denote the optimized transformation vector w^* with the step size α, defined as $\mathtt{model(}\alpha\mathtt{)} = G(z+\alpha w^*)$.

The previous setup assumes linear latent space walks, but we can also learn non-linear trajectories in which the walk direction depends on the current latent space position. For the non-linear walk, we learn a function, $f^*(z)$, which corresponds to a small ϵ-step transformation $\mathtt{edit}(G(z), \epsilon)$. To achieve bigger transformations, we apply f recursively, mimicking discrete Euler ODE approximations. Formally, for a fixed ϵ, we minimize

$$\mathcal{L} = \mathbb{E}_{z,n}[||G(f^n(z)) - \mathtt{edit}(G(z), n\epsilon))||], \tag{2.2}$$

where $f^n(\cdot)$ is an nth-order function composition $f(f(f(...)))$, and $f(z)$ is parametrized with a neural network. We discuss further implementation details in Appendix A.1.4. We use this function composition approach rather than the simpler setup of $G(z + \alpha\text{NN}(z))$ because the latter learns to ignore the input z when α takes on continuous values, and is thus equivalent to the previous linear trajectory (see Appendix A.1.3 for further details).

2.3.2 Quantifying steerability

We further seek to quantify how well we can achieve desired image manipulations under each transformation. We compare the distribution of a given attribute, e.g., “luminance", in the dataset versus in images generated after walking in latent space.

For color transformations, we consider the effect of increasing or decreasing the α coefficient corresponding to each color channel. To estimate the color distribution of generated images, we randomly sample $N = 100$ pixels per image before and after taking a step in latent space. Then, we compute the pixel value for each channel, or the mean RGB value for luminance, and normalize the range between 0 and 1.

For zoom and shift, we rely on an object detector to capture the central object in the image class. We use a *MobileNet-SSD v1* [142] detector to estimate object bounding boxes, and average over image classes recognizable by the detector. For each successful detection, we take the highest probability bounding box corresponding to the desired class and use that to quantify the amount of transformation. For the zoom operation, we use the area of the bounding box normalized by the area of the total image. For shift in the X and Y directions, we take the center X and Y coordinates of the bounding box, and normalize by image width or height.

Truncation parameters (as used in Brock et al. [19], Karras et al. [107]) trade off between image diversity and sample quality. When comparing generated images to the dataset distribution, we use the largest possible truncation for the model and perform similar cropping and resizing of the dataset as done during model training (see Brock et al. [19]). When comparing the attributes of generated distributions under different α magnitudes to each other but not to the dataset, we reduce truncation to 0.5 to ensure better performance of the object detector.

2.3.3 Reducing transformation limits

Equations 2.1 and 2.2 learn a latent space walk assuming a pre-trained generative model, thus keeping the model weights fixed. The previous approach allows us to understand the latent space organization and limitations in the model's transformation capacity. To overcome these limits, we explore adding data augmentation by editing the training images with each corresponding transformation, and train the generative model with this augmented dataset. We also introduce a modified objective function that jointly optimizes the generator weights and a linear walk vector:

$$G^*, w^* = \arg\min_{G,w} \left(\mathcal{L}_{edit} + \mathcal{L}_{GAN}\right), \tag{2.3}$$

where the edit loss encourages low $L2$ error between learned transformation and target image:

$$\mathcal{L}_{edit} = L2\left(G(z{+}\alpha w) - \texttt{edit}(G(z), \alpha)\right). \tag{2.4}$$

The GAN loss optimizes for discriminator error:

$$\mathcal{L}_{GAN} = \max_{D} \left(\mathbb{E}_{z,\alpha}[D(G(z{+}\alpha w))] - \mathbb{E}_{x,\alpha}[D(\texttt{edit}(x, \alpha))]\right), \tag{2.5}$$

where we draw images x from the training dataset and perform data augmentation by applying the `edit` operation on them. This optimization approach encourages the generator to organize its latent space so that the transformations lie along linear paths, and when combined with data augmentation, results in larger transformation ranges which we demonstrate in Sec. 2.4.4

2.4 Experiments

We demonstrate our approach using BigGAN [19], a class-conditional GAN trained on 1000 ImageNet categories. We learn a shared latent space walk by averaging across the image categories, and further quantify how this walk affects each class differently. We focus on linear walks in latent space for the main text, and show additional results on nonlinear walks in Sec. 2.4.3 and Appendix A.2.4. We also conduct experiments on StyleGAN [107], which uses an unconditional style-based generator architecture in Sec. 2.4.3 and Appendix A.2.5.

2.4.1 What transformations can we achieve in latent space?

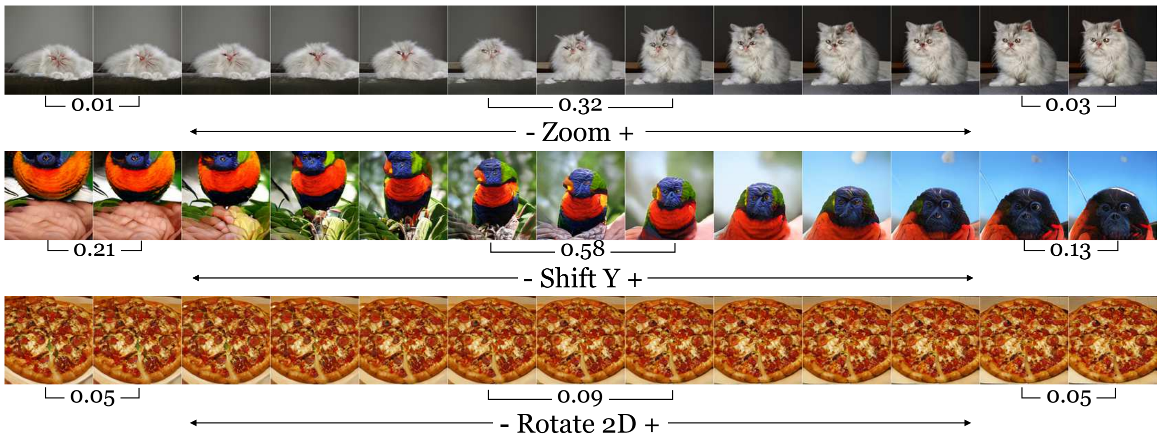


Figure 2-3: **Visualizing transformation limits.** As we increase the magnitude of w^*, the operation either does not transform the image any further, or the image becomes unrealisitic. Below each figure we also indicate the average LPIPS perceptual distance between 200 sampled image pairs of that category. Perceptual distance decreases as we move farther from the source (center image), which indicates that the images are converging.

We show qualitative results of the learned transformations in Fig. 2-1. By steering in the generator latent space, we learn a variety of transformations on a given source image (shown in the center panel of each transformation). Interestingly, several priors come into play when learning these image transformations. When we shift a daisy downwards in the Y direction, the model hallucinates that the sky exists on the top of the image. However, when we shift the daisy up, the model inpaints the remainder of the image with grass. When we alter the brightness of a image, the model transitions between nighttime and daytime. This suggests that the model can extrapolate from the original source image, and still remain consistent with the image context.

However, increasing the step size of α reveals that the degree to which we can achieve each transformation is limited. In Fig. 2-3 we observe two potential failure

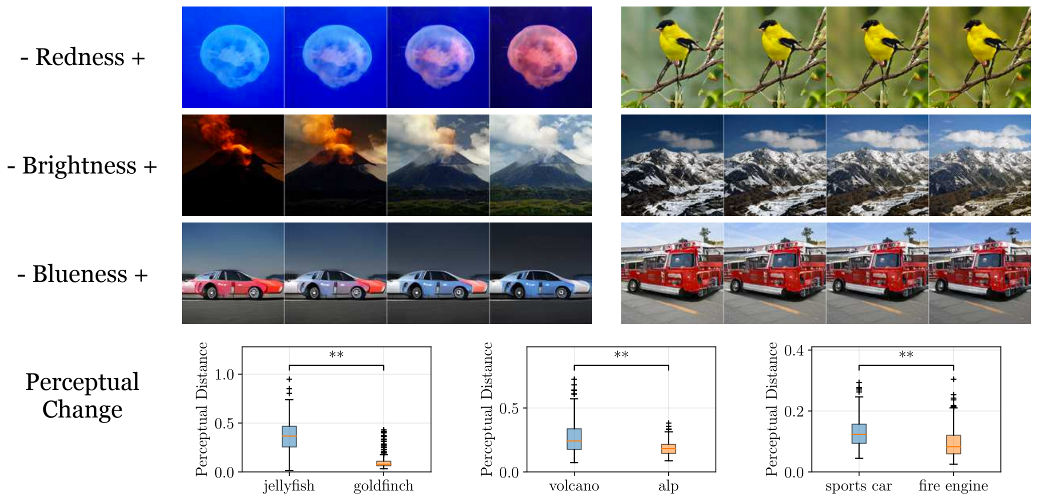


Figure 2-4: **Per-class variation on image transformations.** Each row shows how a single latent direction w^* affects two different ImageNet classes. We observe that changes are consistent with semantic priors (e.g., “Volcanoes" explode, “Alps" do not). Boxplots show the LPIPS perceptual distance before and after transformation for 200 samples per class.

cases: one in which the the image becomes unrealistic, and the other in which the image fails to transform any further. When we try to zoom in on a Persian cat, we observe that the cat no longer increases in size beyond some point, and in fact consistently undershoots the target zoom. On the other hand, when we try to zoom out on the cat, we observe that it begins to fall off the image manifold, and does not become any smaller after some point. Indeed, the perceptual distance (using LPIPS) between images decreases as we push α towards the transformation limits. Similar trends hold with other transformations: we are able to shift a lorikeet up and down to some degree until the transformation yields unrealistic output, and despite adjusting α on the rotation vector, we are unable to rotate a pizza. Are the limitations to these transformations governed by the training dataset? We seek to investigate and quantify these biases in the next sections.

An intriguing characteristic of the learned trajectory is that the amount it affects the output depends on the image class. In Fig. 2-4, we investigate the impact of the walk for different image categories under color transformations. By moving in the direction of a redness vector, we are able to successfully recolor a jellyfish, but we are unable to change the color of a goldfinch, which remains yellow which slight changes in background textures. Likewise, increasing brightness changes an erupting volcano to a dormant one, but does not have much effect on Alps, which only transitions between night and day. In the third example, we use our latent walk to turn red sports cars to blue, but it cannot recolor firetrucks. Again, perceptual distance over image samples confirms these qualitative observations: a 2-sample t-test yields $t = 20.77$, $p < 0.001$ for jellyfish/goldfinch, $t = 8.14$, $p < 0.001$ for volcano/alp, and $t = 6.84$, $p < 0.001$ for sports car/fire engine. We hypothesize that the different impact of the shared transformation on separate image classes relates to the variability in the underlying dataset. The overwhelming majority of firetrucks are red, but sports cars appear in a variety of colors. Therefore, our color transformation is constrained by the dataset biases of individual classes.

With shifting, we can move the distribution of the center object by varying α. In the underlying model, the center coordinate of the object is most concentrated at half of the image width and height, but after applying the shift in X and shift in Y transformation, the mode of the transformed distribution varies between 0.3 and 0.7 of the image width/height. To quantify the distribution changes, we compute the area of intersection between the original model distribution and the distribution after applying each transformation and observe that the intersection decreases as we increase or decrease the magnitude of α. However, our transformations are limited to a certain extent – if we increase α beyond 150 pixels for vertical shifts, we start to generate unrealistic images, as evidenced by a sharp rise in FID and converging modes in the transformed distributions (Fig. 2-5 columns 2 & 3).

We perform a similar procedure for zoom, by measuring the area of the bounding box for the detected object under different magnitudes of α. Like shift, we observe that subsequent increases in α magnitude start to have smaller and smaller effects on the mode of the resulting distribution (Fig. 2-5 last column). Past an 8x zoom in or out, we observe an increase in the FID signifying decreasing image quality. Interestingly for zoom, the FID under zooming in and zooming out is anti-symmetric, indicating that how well we can zoom-in and retain realisitic images differs from that of zooming out. These trends are consistent with the plateau in transformation behavior that we

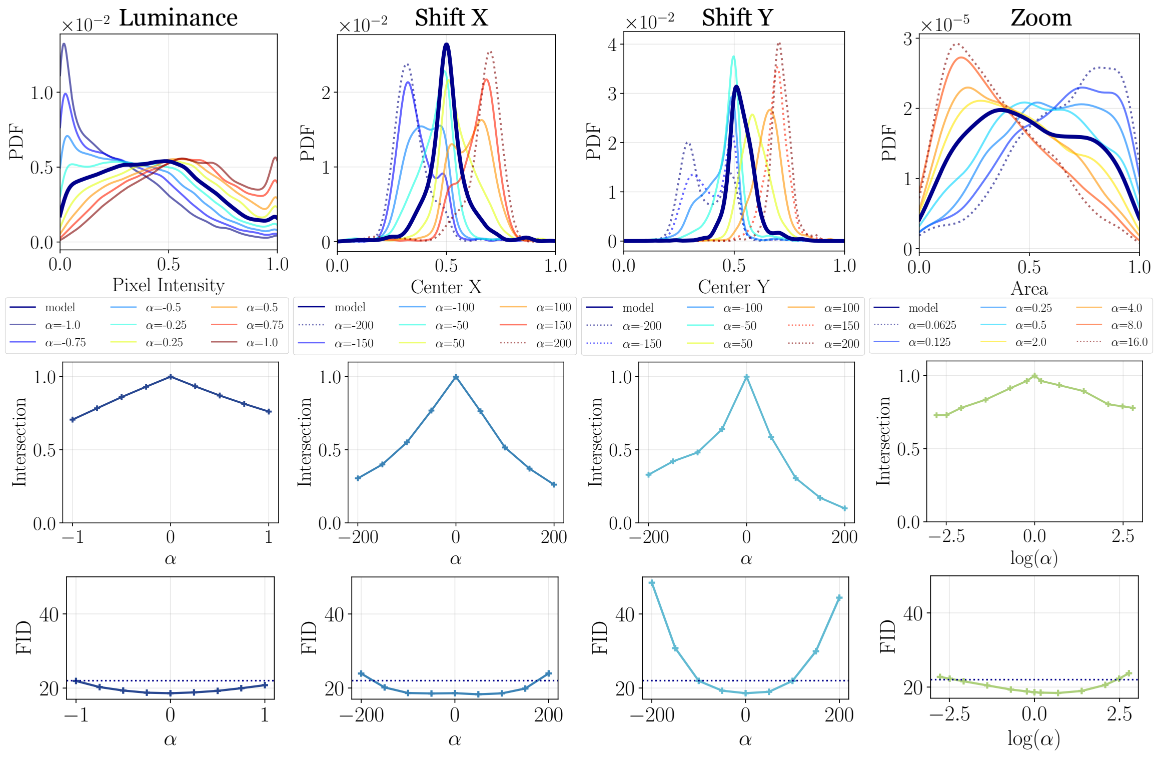


Figure 2-5: **Quantifying the extent of transformations.** We compare the attributes of generated images under the raw model output $G(z)$, compared to the distribution under a learned transformation `model`(α). We measure the intersection between $G(z)$ and `model`(α), and also compute the FID on the transformed image to limit our transformations to the natural image manifold.

qualitatively observe in Fig. 2-3. Although we can arbitrarily increase the α step size, after some point we are unable to achieve further transformation and risk deviating from the natural image manifold.

2.4.2 How does the data affect the transformations?

Is the extent to which we can transform each class, as we observed in Fig. 2-4, due to limited variability in the underlying dataset for each class? One way of quantifying this is to measure the difference in transformed model means, `model(+`α`)` and `model(-`α`)`, and compare it to the spread of the dataset distribution. For each class, we compute standard deviation of the dataset with respect to our statistic of interest (pixel RGB value for color, and bounding box area and center value for zoom and shift transformations respectively). We hypothesize that if the amount of transformation is biased depending on the image class, we will observe a correlation between the distance of the mean shifts and the standard deviation of the data distribution.

Concretely, we define the change in model means under a given transformation as:

$$\Delta\mu_k = \mu_{k,\texttt{model(+}\alpha^*\texttt{)}} - \mu_{k,\texttt{model(-}\alpha^*\texttt{)}} \tag{2.6}$$

for a given class k and we set α^* to be largest and smallest α values used in training. The degree to which we achieve each transformation is a function of α, so we use the same α value for all classes – one that is large enough to separate the means of $\mu_{k,\texttt{model(}\alpha^*\texttt{)}}$ and $\mu_{k,\texttt{model(-}\alpha^*\texttt{)}}$ under transformation, but also for which the FID of the generated distribution remains below a threshold T of generating reasonably realistic images (for our experiments we use $T = 22$).

In Fig. 2-6 we plot the standard deviation σ of the dataset on the x-axis, and the model $\Delta\mu$ under a $+\alpha^*$ and $-\alpha^*$ transformation on the y-axis, as defined in Eq. 2.6. We sample randomly from 100 classes for the color, zoom and shift transformations, and generate 200 samples of each class under the positive and negative transformations. We use the same setup of drawing samples from the model and dataset and computing the statistics for each transformation as described in Sec. 2.4.1.

Indeed, we find that the width of the dataset distribution, captured by the standard deviation of random samples drawn from the dataset for each class, relates to how much we can transform. There is a positive correlation between the spread of the dataset and the magnitude of $\Delta\mu$ observed in the transformed model distributions, and the slope of all observed trends differs significantly from zero ($p < 0.001$ for all transformations). For the zoom transformation, we show examples of two extremes along the trend. For the "robin" class the spread σ in the dataset is low, and subsequently, the separation $\Delta\mu$ that we are able to achieve by applying $+\alpha^*$ and $-\alpha^*$ transformations is limited. On the other hand, for "laptops", the dataset spread is broad, and we are able to attain wider shifts in the model distribution.

From these results, we conclude that the amount of transformation we can achieve relates to the dataset variability. Consistent with our qualitative observations in

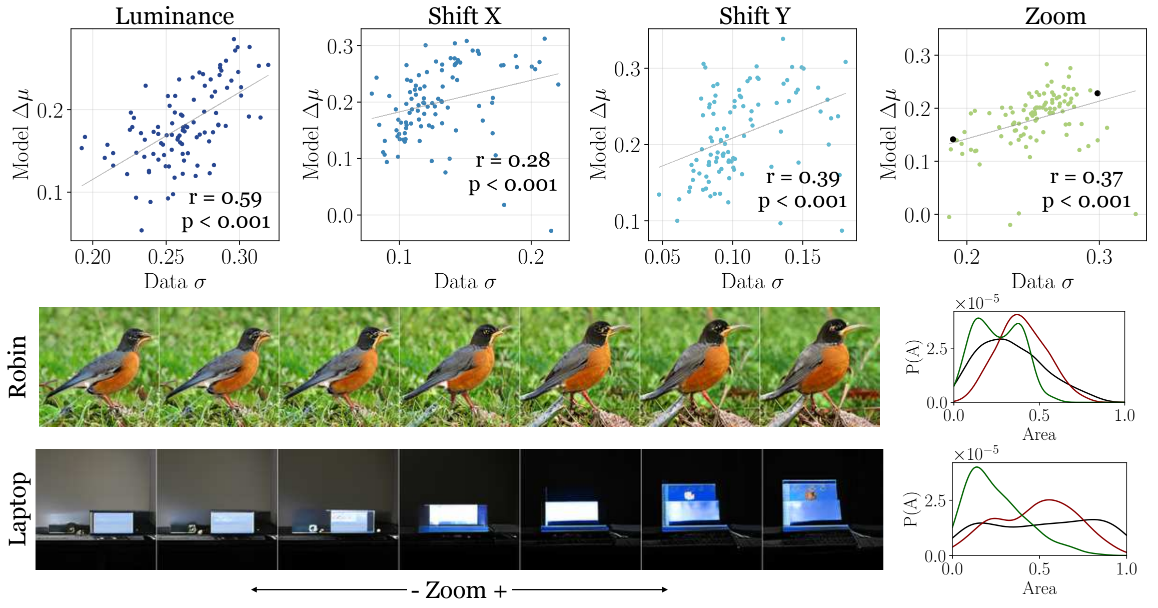


Figure 2-6: **Understanding per-class biases.** We observe a correlation between the variability in the training data for ImageNet classes, and our ability to shift the distribution under latent space transformations. Classes with low variability (e.g., robin) limit our ability to achieve desired transformations, in comparison to classes with a broad dataset distribution (e.g., laptop). To the right, we show the distribution of the zoom attribute in the dataset (black) and under $+\alpha$ (red) and $-\alpha$ (green) transformations for these two examples.

Fig. 2-4, we find that if the images for a particular class have adequate coverage over the entire range of a given transformation, then we are better able to move the model distribution to both extremes. On the other hand, if the images for a given class are less diverse, the transformation is limited by this dataset bias.

2.4.3 Alternative architectures and walks

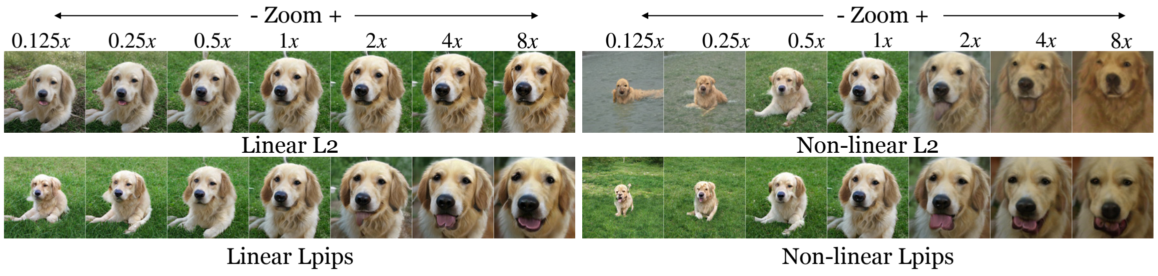


Figure 2-7: **Comparison of linear and nonlinear walks.** For the zoom operation, the linear walk undershoots the targeted level of transformation, but maintains more realistic output.

We ran an identical set of experiments using the nonlinear walk in the BigGAN latent space (Eq 2.2) and obtained similar quantitative results. To summarize, the Pearson's correlation coefficient between dataset σ and model $\Delta\mu$ for linear walks and nonlinear walks is shown in Table 2.1, and full results in Appendix A.2.4. Qualitatively, we observe that while the linear trajectory undershoots the targeted level of transformation, it is able to preserve more realistic-looking results (Fig. 2-7). The transformations involve a trade-off between minimizing the loss and maintaining realistic output, and we hypothesize that the linear walk functions as an implicit regularizer that corresponds well with the inherent organization of the latent space.

	Luminance	Shift X	Shift Y	Zoom
Linear	0.59	0.28	0.39	0.37
Non-linear	0.49	0.49	0.55	0.60

Table 2.1: **Per-class correlation between data and model variability.** Pearson's correlation coefficient between dataset σ and model $\Delta\mu$ for measured attributes. p-value for slope < 0.001 for all transformations.

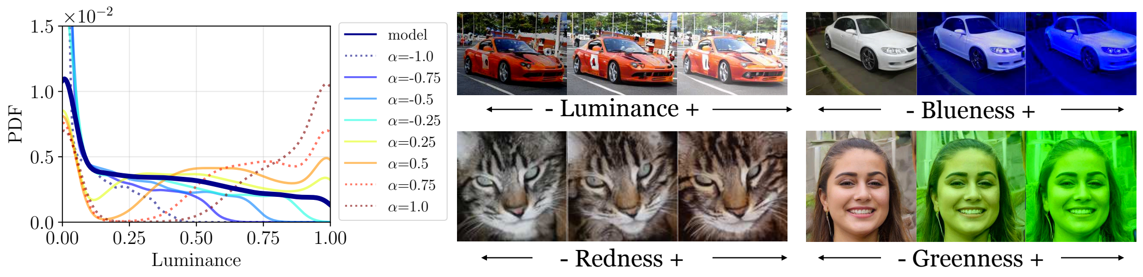


Figure 2-8: **Transformations in an alternative model and latent space.** Distribution for luminance transformation learned from the StyleGAN cars generator, and qualitative examples of color transformations on various datasets using StyleGAN.

To test the generality of our findings across model architecture, we ran similar experiments on StyleGAN, in which the latent space is divided into two spaces, z and W. As Karras et al. [107] notes that the W space is less entangled than z, we apply the linear walk to W and show results in Fig. 2-8 and Appendix A.2.5. One interesting aspect of StyleGAN is that we can change color while leaving other structure in the image unchanged. In other words, while green faces do not naturally exist in the dataset, the StyleGAN model is still able to generate them. This differs from the behavior of BigGAN, where changing color results in different semantics in the image, e.g., turning a dormant volcano to an active one. StyleGAN, however, does not preserve the exact geometry of objects under other transformations, e.g., zoom and shift (see Appendix A.2.5).

2.4.4 Towards steerable GANs

So far, we have frozen the parameters of the generative model when learning a latent space walk for image editing, and observe that the transformations are limited by dataset bias. Here we investigate approaches to overcome these limitations and increase model steerability. For these experiments, we use a class-conditional DCGAN model [182] trained on MNIST digits [124].

To study the effect of dataset biases, we train (1) a vanilla DCGAN and (2) a DCGAN with data augmentation, and then learn the optimal walk in Eq. 2.1 after the model has been trained – we refer to these two approaches in Fig. 2-9 as *argmin W* and *argmin W + aug*, respectively. We observe that adding data augmentation yields transformations that better approximate the target image and attain lower $L2$ error than the vanilla DCGAN (blue and orange curves in Fig. 2-9). Qualitatively, we observe that transformations using the vanilla GAN (*argmin W*) become patchy and unrealistic as we increase the magnitude of α, but when the model is trained with data augmentation (*argmin W + aug*), the digits retain their structural integrity.

Rather than learning the walk vector w assuming a frozen generator, we may also jointly optimize the model and linear walk parameter together, as we formalized in Eq. 2.3. This allows the model to learn an equivariance between linear directions in the latent space and the corresponding image transformations. We refer to this model as *argmin G, W* in Fig. 2-9. Compared to the frozen generator (in *argmin W* and *argmin W + aug*), the joint objective further decreases $L2$ error (green curve in Fig. 2-9). We show additional qualitative examples in Appendix A.2.8. The steerable range of the generator increases with joint optimization and data augmentation, which provides additional evidence that training data bias impacts the models' steerability and generalization capacity. We tried DCGAN on CIFAR10 as a more complicated dataset, however were unable to get steering to be effective – all three methods failed to produce realistic transformations and joint training in fact performed the worst. Finding the right steering implementation per GAN and dataset, especially for joint training, may be a difficult problem and an interesting direction for future work.

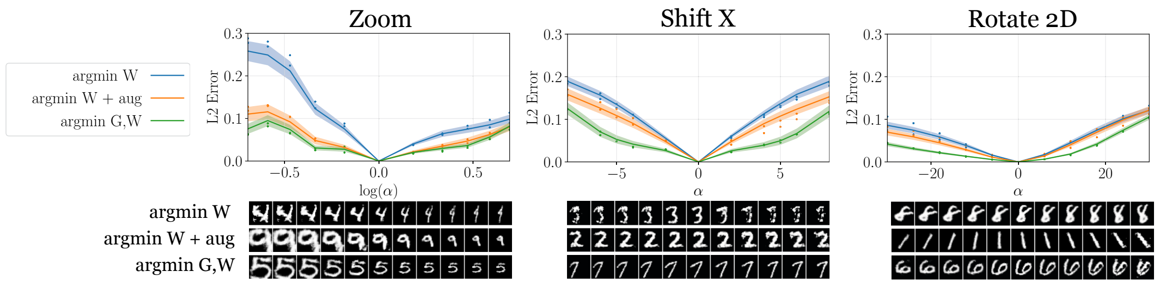


Figure 2-9: **Reducing the effect of transformation limits.** Using DCGAN trained on MNIST digits, we compare the $L2$ reconstruction errors on latent space walks for models trained with vanilla GANs without ($argmin\ W$) and with data augmentation ($argmin\ W + aug$). We also compare to jointly optimizing the generator and the walk parameters with data augmentation ($argmin\ G, W$), which achieves the lowest $L2$ error.

2.5 Conclusion

GANs are powerful generative models, but are they simply replicating the existing training datapoints, or can they to generalize beyond the training distribution? We investigate this question by exploring walks in the latent space of GANs. We optimize trajectories in latent space to reflect simple image transformations in the generated output, learned in a self-supervised manner. We find that the model is able to exhibit characteristics of extrapolation – we are able to “steer" the generated output to simulate camera zoom, horizontal and vertical movement, camera rotations, and recolorization. However, our ability to naively move the distribution is finite: we can transform images to some degree but cannot extrapolate entirely outside the support of the training data. To increase model steerability, we add data augmentation during training and jointly optimize the model and walk trajectory. Our experiments illustrate the connection between training data bias and the resulting distribution of generated images, and suggest methods for extending the range of images that the models are able to create.

Chapter 3

Exemplar-based Control via Compositionality in Latent Space

WITH MINOR MODIFICATIONS FROM:

USING LATENT SPACE REGRESSION
TO ANALYZE AND LEVERAGE COMPOSITIONALITY IN GANS.

LUCY CHAI, JONAS WULFF, PHILLIP ISOLA; ICLR 2021.

Learned vectors in latent space can transform generated content according to natural image variations, but each transformation requires a separate objective function and training process. In this chapter, we unify various image transformations under a single framework. Our method takes as input an approximate template of our desired result, and uses the priors of a pretrained generator to rectify this template into a realistic outcome. Unlike applying vectors in latent space, which can only change the magnitude of the target transformation, this method allows us to achieve diverse and multimodal editing behaviors using just a single trained model and image exemplars.

3.1 Introduction

Natural scenes are comprised of disparate parts and objects that humans can easily segment and interchange [14]. Recently, unconditional generative adversarial networks

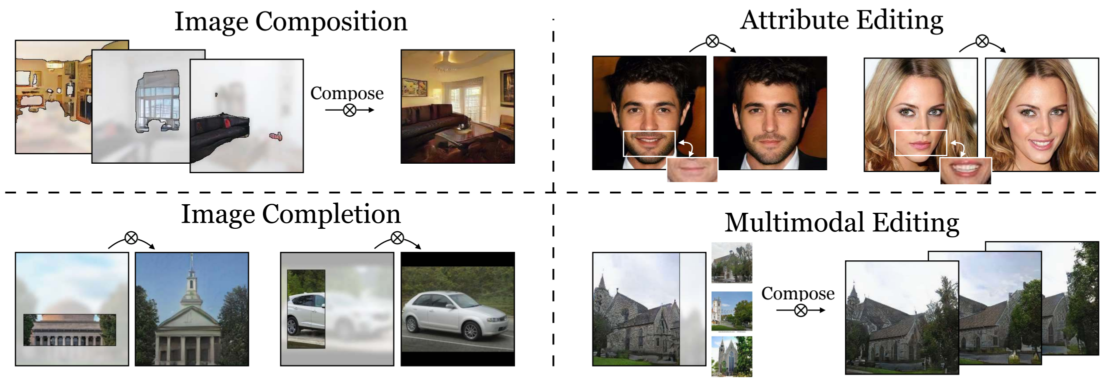


Figure 3-1: **Exemplar-based image editing with latent regression.** Simple latent regression on a fixed, pretrained generator can perform a number of image manipulation tasks based on single examples without requiring labelled concepts during training. We use this to probe the ability of GANs to compose scenes from image parts, suggesting that a compositional representation of objects and their properties exists already at the level of the latent code *.

[106, 109, 107, 182] have become capable of mimicking the complexity of natural images by learning a mapping from a latent space noise distribution to the image manifold. But how does this seemingly unstructured latent space produce a strikingly realistic and structured scene? Here, we use a latent regressor to probe the latent space of a pretrained GAN, allowing us to uncover and manipulate the concepts that GANs learn about the world in an unsupervised manner.

For example, given a church image, is it possible to swap one foreground tree for another one? Given only parts of the building, can the missing portion be realistically filled? To achieve these modifications, the generator must be compositional, i.e., understanding discrete and separate representations of objects. *We show that the pretrained generator – without any additional interventions – already represents these compositional properties in its latent code.* Furthermore, these properties can be manipulated using a regression network that predicts the latent code of a given image. The pixels of this image then provide us with an intuitive interface to control and modify the latent code. Given the modified latent code, the network then applies image priors learned from the dataset, ensuring that the output is always a coherent scene regardless of inconsistencies in the input (Fig. 3-1).

Our approach is simple – using a fixed pretrained generator, we train a regressor network to predict the latent code from an input image, while adding a masking modification to learn to handle missing pixels. To investigate the GAN's ability to produce a globally coherent version of a scene, we hand the regressor a rough, incoherent template of the scene we desire, and use the two networks to convert it into a realistic image. Even though our regressor is never trained on these unrealistic templates, it projects the given image into a reasonable part of the latent space, which the generator maps onto the image manifold. This approach requires no labels or clustering of attributes; all we need is a single example of approximately how we want the generated image to look. It only requires a forward pass of the regressor and generator, so there is low latency in obtaining the output image, unlike iterative optimization approaches that can require upwards of a minute to reconstruct an image.

We use the regressor to investigate the compositional capabilities of pretrained GANs across different datasets. Using input images composed of different image parts ("collages"), we leverage the generator to recombine this unrealistic content into a coherent image. This requires solving three tasks simultaneously – blending, alignment, and inpainting. We then investigate the GAN's ability to independently vary localized portions of a given image. In summary, our contributions are:

- We propose a latent regression model that learns to perform image reconstruction even in the case of incomplete images and missing pixels and show that the combination of regressor and generator forms a strong image prior.

- Using the learned regressor, we show that the representation of the generator is already compositional in the latent code, without having to resort to intermediate layer activations.

- There is no use of labelled attributes nor test-time optimization, so we can edit images based on a single example of the desired modification and reconstruct in real-time.

- We use the regressor to probe what parts of a scene can vary independently, and investigate the difference between image mixing using the encoder and interpolation in latent space.

- The same regressor setup can be used for a variety of other image editing applications, such as multimodal editing, scene completion, or dataset rebalancing.

3.2 Related Work

Image inversion. Given a target image, the GAN inversion problem aims to recover a latent code which best generates the target. Image inversion comes with a number of challenges, including 1) a complex optimization landscape and 2) the generator's inability to reconstruct out-of-domain images. To relax the domain limitations of the generator, one possibility is to invert to a more flexible intermediate latent space [1], but this may allow the generator to become overly flexible and requires regularization so the recovered latent code does not deviate too far from the latent manifold [179, 291, 262]. An alternative to increasing the generator's flexibility is to learn an ensemble of latent codes that approximate a target image when combined [72]. Due to challenging optimization, the quality of inversion depends on good initialization. A number of approaches use a hybrid of a latent regression network to provide an initial guess of the latent code with subsequent optimization of the latent code [11, 74] or the generator weights [292, 12, 170], while Huh et al. [92] investigates gradient-free approaches for optimization. Besides inverting whole images, a different use case of image inversion through a generator is to complete partial scenes. When using optimization, this is achieved by only measuring the reconstruction loss on the known pixels [17, 72, 2], whereas in feed-forward methods, the missing region must be provided explicitly to the model. Rather than inverting to the latent code of a pretrained generator, one can train the generator and encoder jointly, based on modifications to the Variational Autoencoder [115]. Donahue et al. [50], Donahue and Simonyan [49], Dumoulin et al. [51] use this setup to investigate the properties of latent representations learned during training, while Pidhorskyi et al. [178] and Heljakka et al. [80] demonstrate a joint learning method that can achieve comparable image quality to recent GANs. In our work, we investigate the *emergent priors of a pretrained GAN* using a masked latent regression network as an approximate image inverter. While such a regressor has lower reconstruction accuracy than optimization-based techniques, its lower latency allows us to investigate the learned priors in a computationally efficient way and edit images in real-time using such priors.

Composition in image domains. To join segments of disparate image sources into one cohesive output, early works use hand-designed features, such as Laplacian pyramids for seamless blending [22]. Hays and Efros [78] and Isola and Liu [95] employ nearest-neighbor approaches for scene composition and completion. More recently, a number of deep network architectures have been developed for compositional tasks. For discriminative tasks, Tabernik et al. [230] and Kortylewski et al. [119] train CNNs with modified compositional architectures to understand model interpretability and reason about object occlusion in classification. For image synthesis, Mokady et al. [157] and Press et al. [181] use an autoencoder to encode, disentangle, and swap properties between two sets of images, while Shocher et al. [215] mixes images in deep feature space while training the generator. Rather than creating models specifically for image composition or scene completion objectives, we investigate the ability of a pre-trained GAN to mix-and-match parts of its generated images. Related to our work, Besserve et al. [13] estimates the modular structure of GANs by learning a causal model of latent representations, whereas we investigate the GAN's compositional properties using image inversion. Due to the imprecise nature of image collages, compositing image parts also involves inpainting misaligned regions. However, in contrast to inpainting, in which regions have to be filled in otherwise globally consistent images [176, 93, 272, 276], the composition problem involves correcting inconsistencies as well as filling in missing pixels.

Image editing. A recent topic of interest is editing images using generative models. A number of works propose linear attribute vector editing directions to perform image manipulation operations [68, 98, 210, 116, 107, 182]. It is also possible to identify concepts learned in the generator's intermediate layers by clustering intermediate representations, either using segmentation labels [10] or unsupervised clustering [42], and change these representations to edit the desired concepts in the output image. Suzuki et al. [229] use a spatial feature blending approach which mixes properties of target images in the intermediate feature space of a generator. On faces, editing can be achieved using a 3D parametric model to supervise the modification [233, 234]. In our work, we do not require clusters or concepts in intermediate layers to be defined a priori, nor do we need distinct input and output domains for approximate collages and real images, as in image translation tasks [293, 3]. Unlike image manipulation using semantic maps [172, 73], our approach respects the style of the manipulation (e.g. the specific color of the sky), which is lost in the semantic map representation. Our method shares commonalities with Richardson et al. [188], although we focus on investigating compositional properties rather than image-to-image translation. In our approach, we only require a single example of the approximate target property we want to modify and use regression into the latent space as a fast image prior to create a coherent output. This allows us to create edits that are not contingent on labelled concepts, and we do not need to modify or train the generator.

3.3 Method

3.3.1 Latent code recovery in GANs

GANs provide a mapping from a predetermined input distribution to a complex output distribution, e.g. from a standard normal $\mathcal{Z}$ to the image manifold $\mathcal{X}$, but they are not easily invertible. In other words, given an image sample from the output distribution, it is not trivial to recover the sample from the input distribution that generated it. The image inversion objective aims to find the latent code z of GAN G that best recovers the desired target image x:

$$z^* = \arg\min_z(\text{dist}(G(z),\ x)), \tag{3.1}$$

using some metric of image distance dist, such as pixel-wise L_1 error or a metric based on deep features. This objective can be solved iteratively, using L-BFGS [138] or other optimizers. However, iterative optimization is slow – it takes a large number of iterations to converge, is prone to local minima, and must be performed for each target image x independently.

An alternative way of recovering the latent code z is to train a neural network to directly predict it from a given image x. In this case, the recovered latent code is simply the result of a feed-forward pass through a trained regressor network, $z^* = E(x)$, where E can be used for any $x \in \mathcal{X}$. To train the regressor (or encoder) network E, we use the latent encoder loss

$$\mathcal{L} = \mathbb{E}_{z \sim N(0,1),\, x=G(z)} \left[||x - G(E(x))||_2^2 + \mathcal{L}_p(x, G(E(x))) + \mathcal{L}_z(z, E(x)) \right]. \qquad (3.2)$$

We sample z randomly from the latent distribution, and pass it through a pretrained generator G to obtain the target image $x = G(z)$. Between the target image x and the recovered image $G(E(x))$, we use a mean square error loss to guide reconstruction and a perceptual loss $\mathcal{L}_p$ [281] to recover details. Between the original latent code z and the recovered latent code $E(x)$, we use a latent recovery loss $\mathcal{L}_z$. We use mean square error or a variant of cosine similarity for latent recovery, depending on the GAN's input normalization. Additional details can be found in Supp. Sec. B.1.1.

Throughout this paper the generators are frozen, and we only optimize the weights of the encoder E. When using ProGAN [106], we train the encoder network to directly invert to the latent code z. For StyleGAN [109], we encode to an expanded $\mathcal{W}^+$ latent space [1]. Once trained, the output of the latent regressor yields a latent code such that the reconstructed image looks perceptually similar to the target image.

3.3.2 Learning with missing data

When investigating the effect of localized parts of the input images, we might want to treat some image regions explicitly as "unknown", either to create buffer zones to avoid seams between different pasted parts or to explicitly let the image prior fill in unknown regions. In optimization approaches using Eqn. 3.1, this can be handled by optimizing only over the known pixels. However, a regressor network cannot handle

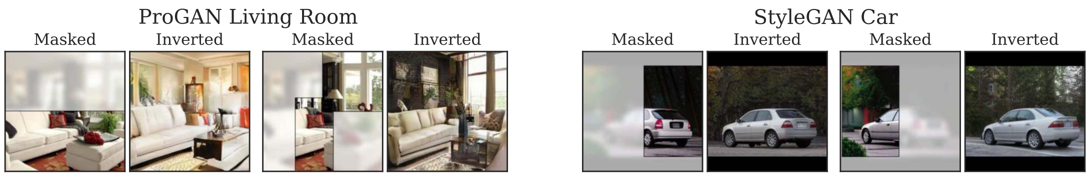


Figure 3-2: **Image completions using the latent space regressor.** Given a partial image, a masked regressor realistically reconstructs the scene in a way that is consistent with the given context. The completions ("Inverted") can vary depending on the exposed context region of the same input.

this naively – it cannot distinguish between unknown pixels and known pixels, and will try to fit the values of the unknown pixels. This can be mitigated with a small modification to the regression network, by indicating which pixels are known versus unknown as input (Fig. 3-3):

$$\mathcal{L} = \mathbb{E}_{z\sim N(0,1),\, x=G(z)}||x - G(E(x_m, m))||_2^2 + \mathcal{L}_p(x, G(E(x_m, m))) + \mathcal{L}_z(z, E(x_m, m)) \quad (3.3)$$

Instead of taking an image x as input, the encoder takes a masked image x_m and a mask m, where $x_m = x \otimes m$, and m is an additional channel of input. Intuitively, this masking operation is analogous to "dropout" [227] on pixels – it encourages the encoder to learn a flexible way of recovering a latent code that still allows the generator to reconstruct the image. Thus, given only partial images as input, the encoder is encouraged to map from the known pixels to a latent code that is semantically consistent with the rest of the image. This allows the generator to reproduce an image that is both likely under its prior and consistent with the observed region.

To obtain the masked image during training we take a small patch of random uniform noise u, upsample this noise to the full resolution using bilinear interpolation, and mask out all pixels where the upsampled noise is smaller than a sampled threshold $t \sim \mathcal{U}(0, 1)$ to simulate arbitrarily shaped mask boundaries. However, at test time, the exact form of the mask does not matter – the mask simply indicates where the generator should try to reconstruct or inpaint, and does not distinguish between the different image parts of the input (additional details in Supp. Sec. B.1.1 and B.2.3).

The regressor and generator pair enforces global coherence: when we obscure or modify parts of the input, the generator will create an output that is still overall consistent. By masking out arbitrary parts of the image (Eqn. 3.3), we allow the GAN to imagine a realistic completion of the missing pixels, which can vary based on the given context (Fig. 3-2). This suggests that the regressor inherently learns an unsupervised object representation, allowing it to complete objects from only partial hints even though the generator and regressor are never provided with structured concept labels during training.

3.3.3 Image composition using latent regression

The regressor E and generator G form an image prior to project any input image x_{input} onto the manifold of generated images $\mathcal{X}$, even if $x_{\text{input}} \notin \mathcal{X}$. We leverage this to investigate the compositional properties of the latent code. We extract parts of images (either generated by G or from real images), and combine them to form a collaged image x_{clg}. This extraction process does not need to be precise and can have obvious seams and missing pixels. At the same time, while x_{clg} is often not realistic, our encoder is aware of these missing pixels and can properly process them, as described in Sec. 3.3.2. We can therefore use the E and G to blend the seams and produce a realistic composite output. To create x_{clg}, we sample base images x_i and masks mask_i, and combine them via union; once we have formed the collage x_{clg}, we reproject via the regressor and generator to obtain the composite x_{rec}:

$$\begin{aligned} x_{\text{clg}} &= \bigcup_i \text{mask}_i \otimes x_i; \\ x_{\text{rec}} &= G(E(x_{\text{clg}}, \cup_i \text{mask}_i)). \end{aligned} \tag{3.4}$$

Note that each mask_i used to extract individual image parts in Eqn. 3.4 is not available to the encoder, only the union is provided in the form of a binary mask. Also, the regressor is trained solely for the latent recovery objective (Eqn. 3.3) and has never seen collaged images during training. To automate the process of extracting masked images, we use a pretrained segmentation network [265] and sample from the output classes (see Supp. Sec. B.1.2). However, the masked regressor is agnostic to how image

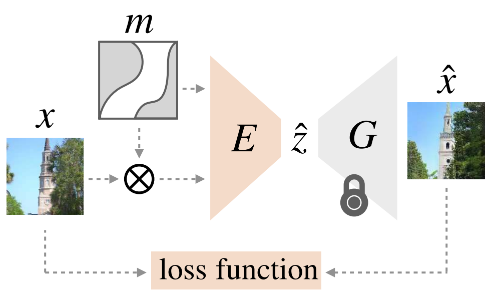


Figure 3-3: **Latent regression network training.** The latent space regressor E to predict the latent code $\hat{z}$ that, when passed through a fixed generator, reconstructs input x. At training and test time, we can also modify the encoder input with additional binary mask m (Eqn. 3.3). Inference requires only a forward pass and the input x can be unrealistic, as the encoder and generator serve as a prior to map the image back to the image manifold.

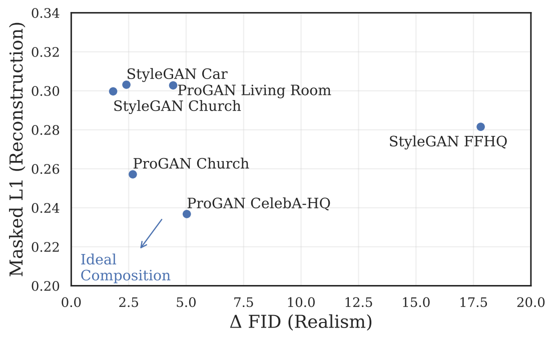


Figure 3-4: **Tradeoffs in realism and reconstruction.** Composition of unrealistic input collages is a balance of two factors: we want to reconstruct the input (low L_1), but still retain realistic images (low FID). Using automatic collages of synthesized image parts, we plot this tradeoff of masked L_1 error between the input collages and output composites, and FID change between the output composites and re-encoded images on different image domains.

parts are extracted; we also experiment with a saliency network [140], approximate rectangles, and user-defined masks in Supp. Sec. B.2.1 and B.2.4.

3.4 Experiments

Using pre-trained Progressive GAN [106] and StyleGAN2 [109] generators, we conduct experiments on CelebA-HQ and FFHQ faces and LSUN cars, churches, living rooms, and horses to investigate the compositional properties that GANs learn from data.

3.4.1 Image composition from approximate collages

The masked regressor and the pretrained GAN form an image prior which converts unrealistic input images into realistic scenes while maintaining properties of the input image. We use this property to investigate the ability of GANs to recombine synthesized collages; i.e., to join parts of different input images into a coherent composite output image. The goal of a truly "compositional" GAN would be to both preserve the input parts and unify them realistically in the output. As we do not have ground-truth composite images, we create them automatically using randomly extracted image parts. The regressor and generator must then simultaneously 1) blend inconsistencies between disparate image parts 2) correct misalignments between the parts and 3) inpaint missing regions, balancing global coherence of the output image with its similarity to the input collage.

Using extracted and combined image parts (Eqn. 3.4), we show qualitative examples of these input collages and the corresponding generated composite across a variety of domains (Fig. 3-5); note that the inputs are not realistic, often with imperfect detections and misalignments. However, the learned image prior from the generator and encoder fixes these inconsistencies to create realistic outputs.

To measure the tradeoff between the networks' ability to preserve the input and the realism of the composite image, we compute masked L_1 distance as a metric of reconstruction (lower is better)

$$\text{avg}(m \otimes |x - G(E(x, m))|) \tag{3.5}$$

and FID score [82] over 50k samples as a metric of image quality (lower is better). To isolate the realism of the composite image from the regressor network's native reconstruction capability (*i.e.* the ability to recreate a single image generated by G), we compare the difference in FID between the reconstructed composites (x_{rec} in Eqn. 3.4), and re-encoded images $G(E(G(z))$. In Fig. 3-4, we plot these two metrics for both ProGAN and StyleGAN across various dataset domains. Here, an ideal composition would attain zero L_1 error (perfect reconstruction of the input) and zero FID increase (preserves realism), but this is impossible, hence the generators demonstrate a balance of these two ideals along a Pareto front. Full results on FID, density & coverage [161], and L_1 reconstruction error and more random samples are in Supp. Sec. B.2.4.

3.4.2 Compositional properties across architectures

Given approximate and unrealistic input collages, the combination of regressor and generator imposes a strong image prior, thereby correcting the output so that it

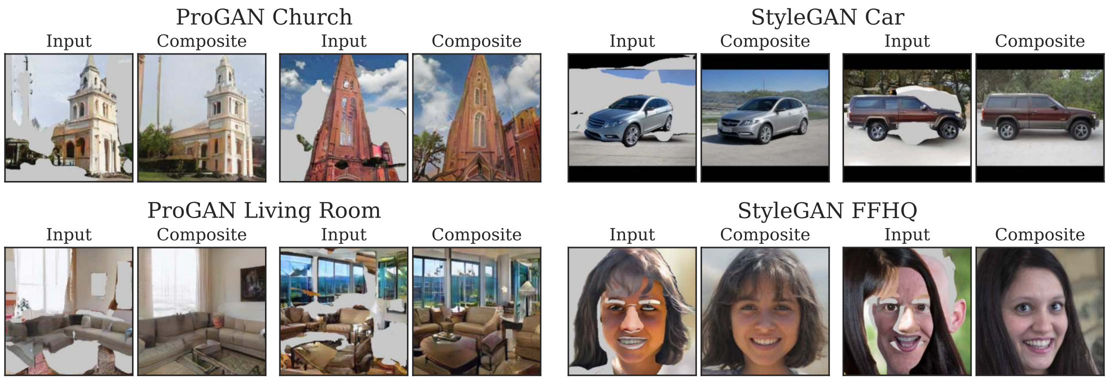


Figure 3-5: **Image collaging under a generative prior.** Trained only on a masked reconstruction objective, a regressor into the latent space of a pretrained GAN allows the generator to recombine components of its generated images, despite strong misalignments and missing regions in the input. Here, we show automatically generated collaged inputs from extracted image parts and the corresponding outputs of the generators.

becomes realistic. How much does the pretrained GAN and the regression network each contribute to this outcome? Here, we investigate a number of different image reconstruction methods spanning three major categories: autoencoder architectures without a pretrained GAN, optimization-based methods of recovering a GAN latent code without an encoder, and encoder-based methods paired with a pretrained GAN. For comparison, we use the same set of collages to compare the methods, generated from parts of random real images of the church and face domains. As some methods take several minutes to reconstruct a single image, we use 200 collages for each domain. Due to the smaller sample size, we use density here as a measure of realism (higher is better), which measures proximity to the real-image manifold [161] and compare to L_1 reconstruction (Eqn. 3.5); a perfect composite has high density and low L_1. We report additional metrics in Tab. B.3-B.4.

For the church domain, we first compare to autoencoding methods that train the generator and encoder portions jointly: DIP [242], Inpainting [272], CycleGAN [293], and SPADE [172]. For iterative optimization methods using only the pretrained generator, we compare direct LBFGS optimization of the latent code [138], Multi-Code Prior [72], and StyleGAN projection [107]. Third, we use our regressor network to directly predict the latent code in a feed-forward pass (Encode), and additionally further optimize the predicted latent to decrease reconstruction error (Enc+LBFGS). We

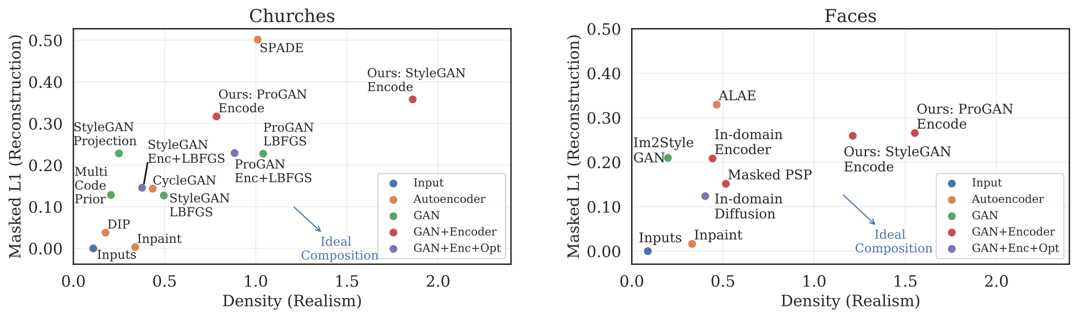


Figure 3-6: **Realism/reconstruction tradeoff for method variations.** Comparing reconstruction of image collages (masked L_1) to realism of the generated outputs on random church collages (left) and face collages (right) across different image reconstruction methods, broadly characterized as autoencoders, GAN-based optimization, GANs with an encoder to perform latent regression, and a combination of GAN, regression, and optimization. An ideal composition has low L_1 and high density (close to real image manifold), and each method exhibits different tradeoffs in reconstruction and realism.

provide additional details on each method in Supplementary Sec. B.2.4. Qualitatively, the different methods have varying degrees of realism when trying to reconstruct unrealistic input collages (we show examples in Supp. Fig. B-10); optimization-based methods such as Deep Image Prior, Multi-Code Prior, and StyleGAN projection tend to overfit and lead to unrealistic reconstructions with low density, whereas segmentation-based methods such as SPADE are not trained to reconstruct the input, leading to high L_1. Our StyleGAN encoder yields the most realistic composites with highest density, at the cost of distorting the unrealistic inputs. Fig 3-6-(left) illustrates this compositional tradeoff, where the encoder based methods perform slightly worse in L_1 reconstruction compared to optimization approaches, but maintain more realistic output and can reconstruct with lower computational time.

On the face domain, we compare the realism/reconstruction tradeoff of composite outputs of optimization-based Im2StyleGAN [1], Inpainting [272], autoencoder ALAE [178], and different regression networks including In-Domain Inversion [291], Pixel2Style2Pixel [188], and our regressor networks. We show qualitative examples in Supplementary Fig. B-11 and a comparison of reconstruction L_1 and density in Fig. 3-6-(right): our ProGAN and StyleGAN masked encoders can maintain closer proximity to the real image manifold (higher density) compared to the alternative methods, with much faster inference time compared to optimization-based methods such as Im2StyleGAN. On these same inputs, ALAE exhibits interesting compositional properties and is qualitatively able to correct misalignments in face patches, but the density of generated images is lower than that of the pretrained GANs. Again, no method can achieve both reconstruction and realism perfectly due to the imprecise nature of the input, and each method balances these factors differently.

3.4.3 How does composition differ from interpolation?

Combining images in pixel space and using the encoder bottleneck to rectify the input is only one way that a generator can mix different images. Another common method is a linear interpolation between two latent codes to obtain an output that has properties of both input images. Here, we investigate how these two approaches differ. When composing parts of two images, we desire that *part of a context image stays the same while the remaining portion changes to match a chosen target*: to achieve this *composition*, we select the desired pixels from our context image and the target modification, and pass the result through the encoder to obtain the blended image.

$$G(E(m_1 \otimes x_1 + m_2 \otimes x_2)) \qquad \triangleleft \quad \textit{composition} \tag{3.6}$$

How does this compare to directly interpolating in latent space? We compute the *latent α-blend* by performing a weighted average of the context and target latents:

$$G(\alpha * E(x_1) + (1-\alpha) * E(x_2)) \qquad \triangleleft \quad \textit{latent } \alpha\textit{-blend} \tag{3.7}$$

and the *pixel α-blend* by blending inputs in pixel space and using the encoder bottleneck to make the output more realistic:

$$G(E(\alpha * x_1 + (1-\alpha) * x_2)). \qquad \triangleleft \quad \textit{pixel } \alpha\textit{-blend} \tag{3.8}$$

We select the weight α to be proportional to the area of the target modification. Qualitatively, the composition method is better able to change the target region while keeping the context region fixed, e.g., add white windows while reducing changes in

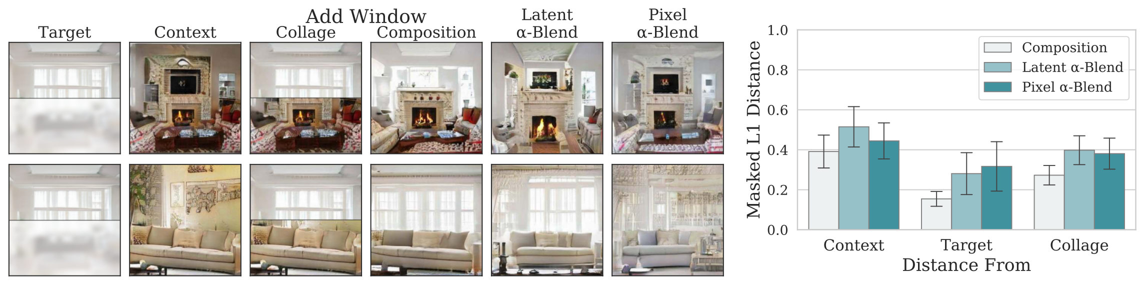
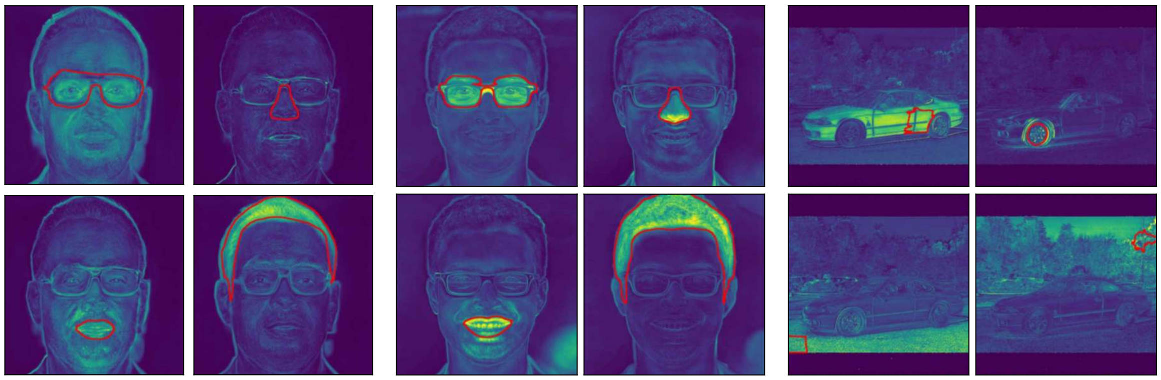


Figure 3-7: **Comparing composition and interpolation.** We aim to apply the same target modification (white windows; Target), to two context sources (Context), where the collage of the two images is shown in the third column (Collage). Compared to Latent and Pixel α-blending, inverting the collages into the latent space via the encoder (Composition) better matches the context and target regions, while at the same time ensuring global coherence between the target and context images.

the fireplace or couch in Fig. 3-7, whereas the other two α-blending methods are less faithful in preserving image content. To quantify this effect, we compute the masked L_1 distance (Eqn. 3.5) of the interpolated images to (1) the original masked context region, (2) to the original masked target region, and (3) to the input collage over 500 random samples. The composition method obtains lower distance from the target and context, and is also closer to the desired collage. Unlike interpolations using attribute vectors, composition manipulations do not need to be learned from data - they are based on a single context and target image, and also allow multiple possible directions of manipulation. We show additional interpolations and comparison to learned attribute vectors in Supp. Sec. B.2.2, and additional applications such as dataset rebalancing and one-shot image editing in Supp. Sec. B.2.1.

3.4.4 Investigating independence of image components

Given these unsupervised objected representations, we seek to investigate the independence of individual components by minimizing “leakage” of the desired edits. For example, if we change a facial expression from a frown to a smile, the hair style should not change. We quantify the independence of parts under the global coherence constraint imposed by the regressor and generator pair by first parsing a given face image x into separate semantic components (such as eyes, nose, etc.) represented as masks m_c. For each component, we generate N new faces x_n, and replace the component region



(a) Face variation, ProGAN (b) Face variation, StyleGAN (c) Part discovery for cars

Figure 3-8: **Image variation when replacing single parts.** In ProGAN (a), replacing single parts leads to change beyond the part that is being changed. Changes in StyleGAN (b) are visually more localized. This method can be used to find correlated regions even without semantic labels, shown for cars (c).

m_c in x by the corresponding region in x_n, yielding (after regression and generation) for each component c a set of N images $x_{c,n} = G(E(m_c \otimes x_n + (1 - m_c) \otimes x))$. We can now measure how much changing component c changes each pixel location by computing the normalized pixel-wise standard deviation of $x_{c,n}$ across the n different replacements as $v_c = \sigma_c / \sum_c \sigma_c$, where $\sigma_c = \sqrt{\mathbb{E}_n[(x_{n,c} - \mathbb{E}_n[x_{n,c}])^2]}$. For a given component c, we measure independence as the average variation outside of c that results from modifying c as $s_c = \mathbb{E}[(1 - m_c) \otimes v_c]$ (a lower s_c means higher independence). We repeat this experiment 100 times and use $N = 20$ samples.

Table 3.1: **Quantifying independence of image parts.** Face part independence across models (lower means more independent), measured as the variation outside of the replaced part, computed over difference replacements.

	Background	Skin	Eyes	Ears	Nose	Mouth	Hair	Average
ProGAN	0.167	0.177	0.043	0.024	0.030	0.038	0.170	0.093
StyleGAN	0.148	0.248	0.062	0.024	0.065	0.045	0.140	0.105

Table 3.1 shows the variations of ProGAN and StyleGAN. StyleGAN better separates the background from the face and reduces leakage when changing the hair; for the face parts, leakage is small for both networks. A notable exception is the "skin" area, which for StyleGAN is more globally entangled. This might be because StyleGAN is generally better able to reason about global effects such as illumination,

which are strongly reflected in the appearance of the skin, yet have a global impact on the image. Figure 3-8(a) and (b) qualitatively show examples for the variation maps v_c for different parts for ProGAN (a) and StyleGAN (b); the replaced part is marked in red. Lastly, this method can be utilized for unsupervised part discovery, as shown in Fig. 3-8(c). Here, changing the color of the rear door (top left) changes the appearance of the whole car body; a change of the tire (top right) is very localized, and the foreground (bottom left) and background (bottom right) are large parts varying together, but distinct from the car. More examples of part variations are shown in Supp. Sec. B.2.5.

3.5 Conclusion

Using a simple latent space regression model, we investigate the compositional properties of pretrained GAN generators. We train a regressor model to predict the latent code given an input image with the additional objective of learning to complete a scene with missing pixels. With this regressor, we can probe various properties and biases that the GAN learns from data. We find that, in creating scenes, the GAN allows for local degrees of freedom but maintains an overall degree of global consistency; in particular, this compositional representation of local structures is already present at the level of the latent code. This allows us to input approximate templates to the regressor model, such as partially occluded scenes or collages extracted from parts of images, and use the regressor and generator as image prior to regenerate a realistic output. The latent regression approach allows us to investigate how the GAN enforces independence of image parts, while being trained without knowledge of objects or explicit attribute labels. It only requires a single forward pass on the models, which enables real time image editing based on single image examples.

Part II

Image Synthesis with Designed Control

Chapter 4

Variable Resolution Synthesis with Continuous Coordinate Control

With minor modifications from:

Any-resolution Training
for High-resolution Image Synthesis

Lucy Chai, Michaël Gharbi, Eli Shechtman, Phillip Isola, Richard Zhang;
ECCV 2022.

We now turn to designed control mechanisms for synthesis. Unlike emergent control, which relies on the model's ability to infer patterns about the world from its training data, designed control explicitly builds in properties about the world via the provided control inputs. In this chapter, we develop a generator with controllable sampling rate and positional awareness, allowing us to train on and synthesize varied-resolution images by conditioning on spatial coordinates. This is in contrast to standard synthesis pipelines, which resize and downsample the training images to a fixed common size as the first preprocessing step. With our approach, we can leverage additional information in the high-resolution pixels that standard synthesis methods discard. By training in a patch-wise fashion, our methodology allows us to synthesize ultra-high-resolution outputs without requiring a larger model architecture or additional computational memory requirements compared to prior fixed-resolution models.

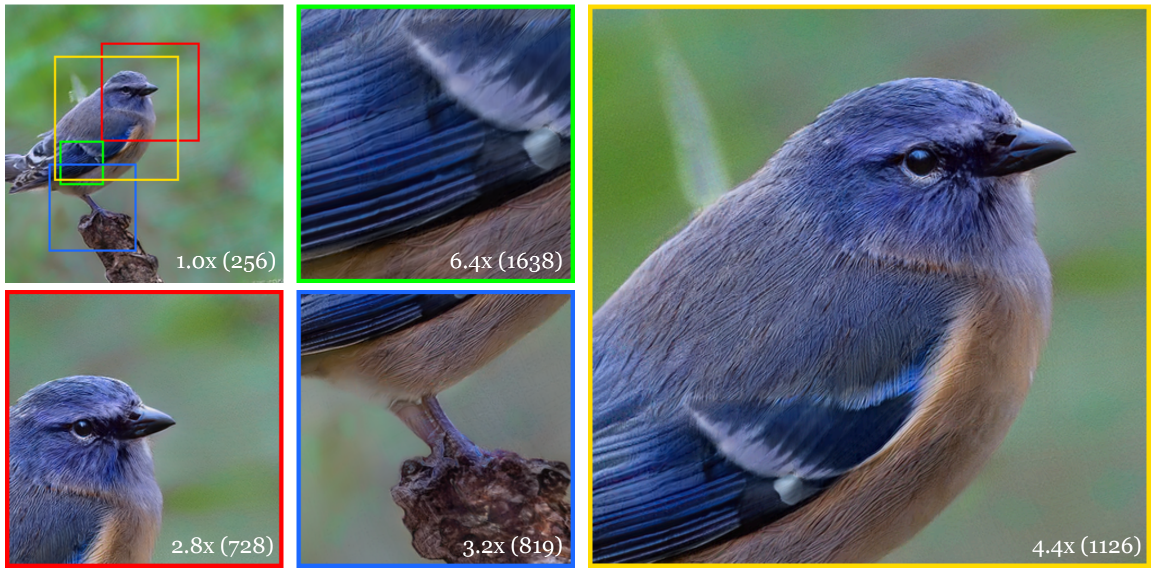


Figure 4-1: **Continuous resolution patch synthesis.** Trained on a dataset of varied-size images, our unconditional generator learns to synthesize patches at continuous scales to match the distribution of real patches. Here, we render crops of the image at different resolutions, indicating the target resolution for each. We indicate the region of each crop in the top-left panel, which is the image directly sampled without scale input.

4.1 Introduction

The first step of typical generative modeling pipelines is to build a dataset with a fixed, target resolution. Images above the target resolution are downsampled, removing high-frequency details, and data of insufficient resolution is omitted, discarding structural information about low frequencies. Our insight is that this process wastes potentially learnable information. We propose to embrace the natural diversity of image sizes, processing them at their *native* resolution.

Relaxing the fixed-dataset assumption offers new, previously unexplored opportunities. One can potentially simultaneously learn global structure – for which large sets of readily-available low-resolution images suffice – and fine-scale details – where even a handful of high-resolution images may be adequate, especially given their internal recurrence [213]. This enables generating images at higher resolutions than previously possible, by adding in higher-resolution images to existing fixed-size datasets.

This problem setting offers unique challenges, both in regards to modeling and scaling. First, one must generate images across multiple scales in order to compare with the target distribution. Naïvely downsampling the full-resolution image is suboptimal, as dataset images can have even 8× difference in scale. Secondly, generating and processing high-resolution images offers scaling challenges. Training at 1024 resolution already pushes current hardware to the limits in memory and training time, and are unable to fully make use of images above that resolution.

To bypass these issues, we design a generator to synthesize image crops at arbitrary scales, hence, performing *any-resolution* training. We modify the state-of-the-art StyleGAN3 [111] architecture to take a grid of continuous coordinates, defined on a bounded domain, as well as a target scale, inspired by recent work in coordinate-based conditioning [156, 8, 134, 33]. By keeping the latent code constant and varying the crop coordinates and scale, our generator can output patches of the same image [133, 204], but at various scales. This allows us to (1) efficiently generate at arbitrary scale, so that a discriminator can compare generations to a multi-resolution dataset, and (2) decouple high-resolution synthesis from increasing model size and memory requirements.

We first experiment with downsampled FFHQ images [106] as a controlled setting and find minimal degradation (FIDs varying by 0.3), even at highly skewed distributions with 98% low-resolution images and just 2% at higher, mixed resolution. Practically, this means we can leverage large-scale (>100k) lower-resolution datasets, such as LSUN Churches [271], Flickr Mountains [173], and Flickr Birds collected by us, and add a relatively small amount of high-resolution images (~ 6000), for continuous resolution synthesis beyond the 1024 resolution limit of current generators. To summarize, we:

- propose to train on mixed-resolution datasets from images in-the-wild.
- modify the generator to be amenable to such data, sampling patches at arbitrary scales during our *any-resolution* training procedure.
- demonstrate successful generations beyond 1024×1024, with fine details and coherent global structure, without a larger and more expensive generator.
- introduce a variant of the FID metric that captures image statistics at multiple scales, thus accounting for the details of high resolutions.

4.2 Related Work

Unconditional image synthesis. Recent generative models including GANs [70, 107, 111, 19], Variational Autoencoders [115], diffusion models [163, 85, 224, 226, 48], and autoregressive models [123, 245, 244] such as transformers [246, 31, 55] are rapidly improving in quality. Of these, we focus on GANs, which offer state-of-the-art performance along with efficient inference and effective editing properties. A key innovation in GANs has been multi-resolution supervision during training. Works such as LapGAN [45], the Progressive/StyleGAN family [106, 107, 109, 111], MSG-GAN [105], and AnyCost-GAN [136] have demonstrated stable training by growing the generator with additional layers that increase resolution by factors of two. Such a strategy even works for single-image GANs [204, 214], based on the observation that images share statistics across scales. While several works [108, 287, 284] show that data augmentations, such as small jitters in scale, can help stabilize training, they are processing the same, underlying fixed-resolution dataset. We draw upon the insights in these works for stable training, and seek to unlock training on an any-resolution dataset. Importantly, our generator does not use additional layers and can synthesize images at continuous scales, not only powers of two.

Coordinate-based functions. Coordinate-based encodings enable spatial conditioning and provide an inductive bias towards natural images [231]. Recent methods use point-based neural mappings to transform 2D or 3D coordinates to a color value for the purposes of unconditional generation [133, 37, 111, 220, 134, 5], conditional generation [205], 3D view synthesis [156, 203, 26], or fitting arbitrary signals [33, 151]. By oversampling the coordinate grid, one can generate a larger image at inference time. However, because these models keep the same fixed-scale dataset assumption during training, the outputs struggle to offer additional high-frequency details without a high-frequency training signal. We draw upon the innovations in coordinate-based functions to sample patches at different scales and locations, enabling us to efficiently train on multiple scales. MS-PIE [266] and MS-PE [37] add positional encodings for multi-scale synthesis, but retain a global image discriminator at smaller resolution.

Concurrently, ScaleParty [167] also samples patches, but their goal is to generate with cross-scale consistency while we focus on training with arbitrary size real images.

Extrapolation. One method of generating "infinite" resolution is extrapolating an image. Early texture synthesis works [53, 52, 258] focus on stationary textures. Recent advances [290] explore non-stationary textures, with large-scale structures and inhomogeneous patterns. Similar approaches operate by outpainting images, extending images beyond their boundaries in a conditional setting [232, 268, 148, 290]. Recent generative models synthesize large scenes [134, 35], typically casting synthesis as an outpainting problem [268, 148]. These methods are most effective for signals with a strong stationary component, such as landscapes, although extrapolation of structured scenes can be achieved in some domains [255]. Unlike textures, the images we wish to synthesize typically have a strong global structure. In a sense, we seek to extrapolate by "zooming in" or out, rather than "panning" beyond an image's boundaries.

Super-resolution. An alternative approach to generating high-resolution imagery would be to start with an off-the-shelf generative model and feed its outputs to a super-resolution method [89, 33, 252, 254], possibly exploiting the self-similarity properties of images [207, 67, 90, 213]. Applying super-resolution models is challenging, in part because of the specific blur kernels super-resolution models are trained on [277]. Furthermore, though generations continue to improve, there remains a persistent domain shift between synthesized and real images [251, 23]. Finally, super-resolution is a local, conditional problem where the global structure is dictated by the low-resolution input, and optionally an additional high-resolution reference image [288, 267, 146, 102, 263]. We synthesize plausible images unconditionally, leveraging a *set* of high-resolution images to produce both realistic global structure and fine details.

4.3 Methods

In standard GAN training, all training images share a common fixed resolution, which matches the generator's output size. We seek to exploit the variety of image resolutions available in the wild, learning from pixels that are usually discarded, to enable high-

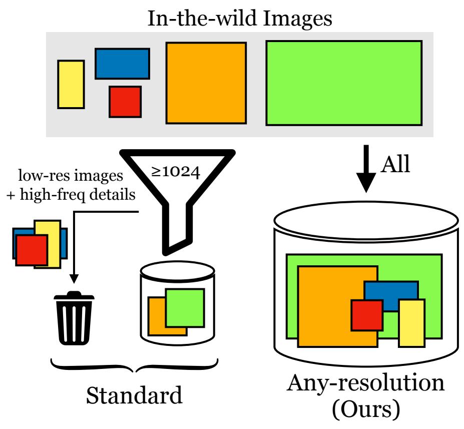


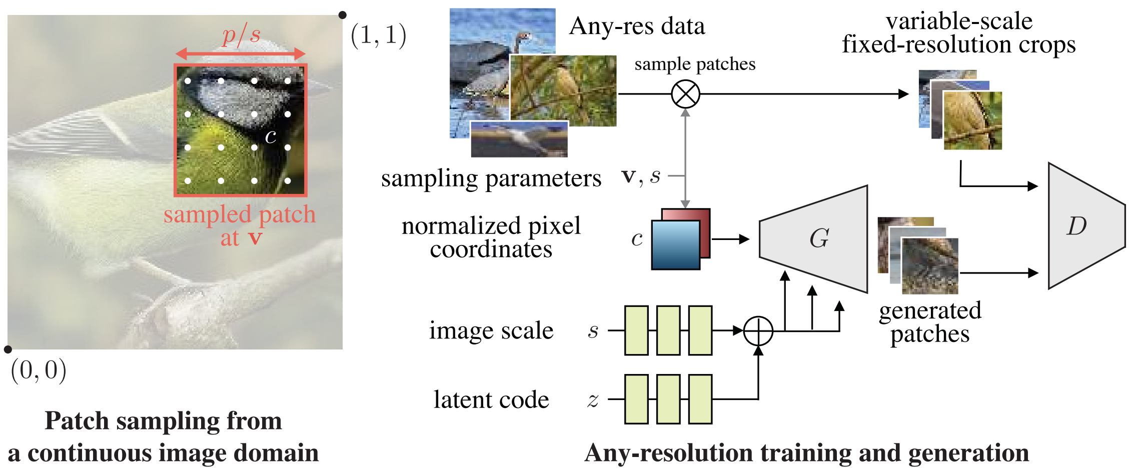
Figure 4-2: **Any-resolution data pipeline.** In-the-wild images and images collected from the internet are naturally captured at variable resolutions. Traditional dataset construction filters out low-resolution images and downsamples high-resolution images to a fixed, training resolution. We aim to keep images at their original resolution and train on these variable resolution images. This allows us to learn from the additional pixels present in high-resolution images.

and continuously-variable resolution synthesis. We achieve this by switching from the common fixed-resolution thinking, to a novel 'any-resolution' approach, where the original size of each training image is preserved (Fig 4-2). We introduce a new class of GAN generators that learn from this multi-resolution signal to synthesize images at any resolution (§ 4.3.1), and show how to train them by sampling patches at multiple scales to jointly supervise the global-structure and fine image details (§ 4.3.2).

4.3.1 Multi-resolution GAN

We design our approach to leverage state-of-the-art GANs. We keep the architecture of the discriminator unchanged. Since the discriminator operates at fixed resolution, we modify the generator to synthesize images at any resolution and receive the discriminator's fixed-resolution supervision. Our implementation builds on the StyleGAN3 framework [111], which is conditioned on a fixed coordinate grid. We modify this grid for any-resolution and patch-based synthesis.

Continuous-resolution generator. We treat each image as a continuous function defined on a bounded normalized coordinate domain $[0, 1] \times [0, 1]$. The generator G always generates patches at a fixed pixel resolution $p \times p$, but each patch implicitly corresponds to a square sub-region, centered at $\mathbf{v} \in [0, 1]^2$, of the larger image. Denoting the resolution of the larger image as $s \times s$, we have that the patch size is p/s in normalized coordinates (see Figure 4-3, left). During training, we sample patches from images at multiple scales s, either from the generator or from the multi-resolution



p/s
$(1,1)$
c
sampled patch at $\mathbf{v}$
$(0,0)$
Patch sampling from a continuous image domain
Any-res data
sample patches
variable-scale fixed-resolution crops
sampling parameters $\mathbf{v}, s$
normalized pixel coordinates c
G
D
image scale s
latent code z
generated patches
Any-resolution training and generation

Figure 4-3: **Anyres-GAN overview.** (Left) We parameterize images (real or synthetic) as continuous functions over a normalized domain and extract random patches at various scales s, but constant resolution p. (Right) To train, we sample crops at random scales and offsets $\mathbf{v}$ from full-size real images. The same crops are sampled from the generator, by passing it a grid of the desired coordinates $c_{\mathbf{v},s}$, and injecting the image scale s through modulation, in addition to a global latent code z.

dataset, before passing them to the fixed-resolution discriminator D. Formally, our generator takes three inputs: a regular grid of normalized continuous pixel coordinates $c_{\mathbf{v},s} \in \mathbb{R}^{p \times p \times 2}$, the resolution $s \in \mathbb{N}$ of the (implicit) larger image the patch is extracted from, and z, the latent code representing this larger image. It synthesizes the patch's pixel values at the sampled coordinates as:

$$G(z, c_{\mathbf{v},s}, s) = G(F(c_{\mathbf{v},s}); M(z, s)), \tag{4.1}$$

where F is a Fourier embedding of the continuous coordinates [111], and M is an auxiliary function that maps the latent code and sampling resolution into a set of modulation parameters for the StyleGAN3 generator (see § 4.3.3 for details). Our method therefore modifies two components from StyleGAN3. First, we replace the fixed coordinate grid with *patch-dependent* coordinates to train on variable-resolution images; these coordinates are adjusted to account for upsampling in StyleGAN3. Second, the added branch M injects scale information throughout the generator.

At test time, we can generate images at arbitrarily high resolutions by sampling the full continuous domain $[0, 1] \times [0, 1]$ at the desired sampling rate. Theoretically, the maximum resolution is infinite, but in practice the amount of detail that the model can generate is determined by generator resolution p and the resolutions of the training images.

4.3.2 Two-phase training

We train our generator in two phases. In the first, we want the generator to learn to generate globally-coherent images. For this, we disable the patch sampling mechanism and pretrain the generator at a fixed scale, corresponding to the full continuous image domain. That is, we fix $s = p$, and $\mathbf{v} = (0.5, 0.5)$, which is equivalent to standard fixed-resolution GAN training. Both the coordinate tensor $c_{\mathbf{v},s}$ and the scale conditioning variable s are constant in this phase, so we simply refer to the image generated as $G_{\text{fixed}}(z)$ and follow the training procedure of StyleGAN3. In the second phase, we enable patch sampling for both the real and synthetic images and continue training the generator using variable-scale patches, so it learns to synthesize fine details at any resolution. We found that using a copy of the pretrained fixed-scale generator G_{fixed} as a teacher model helps stabilize training in this phase.

Global fixed-resolution pretraining. During pretraining, we effectively resample all the training images to a fixed resolution $p \times p$, as in standard GAN training. Let $x \sim \mathcal{D}_{\text{fixed}}$ denote an image sampled from this fixed size dataset. We optimize a standard GAN objective with non-saturating logistic loss and R_1 regularization on the discriminator:

$$\begin{aligned} &V(D, G(z), x) = D(x) - D(G(z)), \quad R_1(D, x) = \|\nabla D(x)\|^2, \\ &G_{\text{fixed}} = \arg\min_G \max_D \; \mathbb{E}_{z,x\sim\mathcal{D}_{\text{fixed}}} \, V(D, G(z), x) - \frac{\lambda_{R_1}}{2} R_1(D, x). \end{aligned} \tag{4.2}$$

We use the recommended values for λ_{R_1}, depending on generator resolution p [111].

Mixed-resolution patch-based training. In the second phase, we enable multi-resolution sampling, alternating between extracting random crops from our any-resolution dataset and generating them with our continuous generator.

For synthetic patches, we sample a patch location $\mathbf{v}$ uniformly in the continuous domain $[0,1] \times [0,1]$; and an arbitrary image resolution $s \geq p$, corresponding to the implicit full image around the square patch. From those, we derive the sampling coordinate grid $c_{\mathbf{v},s}$, and synthesize the patch image $G(z, c_{\mathbf{v},s}, s)$, as described earlier.

For 'real' patches, we sample an image from our dataset. Because this image can have any resolution $s_{\text{im}} \geq p$, we crop it to a random square matching its smallest dimension, then Lanczos downsample this square to a random resolution $s \times s$ with $s_{\text{im}} \geq s \geq p$. Finally, we extract a random $p \times p$ crop from the downsampled image, recording its center $\mathbf{v}$. To preserve the generator's global coherence and continuous generation ability, we sample at global scale $s = p$ and $\mathbf{v} = (0.5, 0.5)$ (similar to the pretraining step) with probability 50%. We found that image quality at global resolution $s = p$ degrades otherwise, and we refer to these generated full images of size $p \times p$ as "base images." Our any-resolution GAN optimizes the following objective during this phase:

$$\begin{aligned} G^* = \arg\min_G \min_D \; & \mathbb{E}_{z,\{x,s,\mathbf{v}\}\sim\mathcal{D}} \, V(D, G(z, c_{\mathbf{v},s}, s), x) \\ & + \lambda_{\text{teacher}} \mathcal{L}_{\text{teacher}}(G, G_{\text{fixed}}, z) - \frac{\lambda_{R_1}}{2} R_1(D, x). \end{aligned} \tag{4.3}$$

We use $\lambda_{\text{teacher}} = 5$; other values offer slight tradeoffs between similarity to the base teacher model G_{fixed}, and FID score (see supplemental). $\mathcal{L}_{\text{teacher}}$ is an auxiliary loss to encourage faithfulness to the pretrained fixed-resolution generator G_{fixed}. The architecture of D remains the same as in the pretraining step; we found that modifying the discriminator setup did not further improve results (see supplemental).

Teacher model. For the second training phase above, we initialize G with the pretrained weights of G_{fixed}. Weights for discriminator D are also kept for fine-tuning. We keep a separate copy of G_{fixed} with frozen weights, the teacher, for additional supervision. We design a loss function that encourages the generated patch (at any resolution), to match the teacher's fixed-resolution output in the corresponding region, after downsampling and proper alignment [94]. Formally, this loss is given by:

$$\mathcal{L}_{\text{teacher}}(G, G_{\text{fixed}}, z) = d\left(m \odot w_{\mathbf{v},s}(G(z, c_{\mathbf{v},s}, s)),\; m \odot G_{\text{fixed}}(z)\right), \tag{4.4}$$

where d is the sum of a pixel-wise ℓ_1 loss, and the LPIPS perceptual distance [92, 281]. The warp function $w_{\mathbf{v},s}$ transforms and resamples the generated high-resolution patch using a band-limited Lanczos kernel, to project it in the coordinate frames of the low-resolution, global image $G_{\text{fixed}}(z)$. Because the warped patch does not cover the entire image domain, we multiply it with a binary mask m to indicate the valid pixels, prior to computing loss d.

4.3.3 Implementation details

Scale-conditioning. In addition to the pixel location $c_{\mathbf{v},s}$, we also pass the global resolution information s to the generator. Knowledge of the global image scale is important to enable continuous scale variations and proper anti-aliasing [33, 8]. We found it beneficial to explicitly inject this information into all intermediate layers of the generator. To achieve this, we use a dual modulation approach [285], embedding the latent code z and scale s separately using two independent sub-networks (we use the same mapping network architecture for each). The two outputs are summed to obtain a set of modulation parameters $M(z, s)$, used to modulate the main generator features. Architectural details of the generator G and mapping network M, can be found in the supplemental.

Synthesizing large images. Our fully-convolutional generator can render image at arbitrary resolutions. But images larger than 1024×1024 require significant GPU memory. Equivalently, we can render non-overlapping tiles that we assemble into a larger image. Our patch-based multi-resolution training and the Fourier encoding of the spatial coordinates make the tile junctions seamless.

4.4 Experiments

We introduce a modified image quality metric that computes FID over multi-scale image patches without downsampling, which is largely correlated to the standard FID metric when ground truth high-resolution images are available (FFHQ), yet more sensitive to the quality of larger resolutions. We then compare our model to alternative approaches for variable scale synthesis and super-resolution on other natural image domains (§ 4.4.1). Finally, we investigate variations of our model and training procedure to validate our design decisions (§ 4.4.2).

Data. Our method is general and can work on collections of any-resolution data. As such, when targeting high-resolution generation, rather than starting over, we can add additional high-resolution (HR) images to existing, fixed-size, low-resolution (LR) datasets. Our datasets and their statistics are listed in Tab. 4.1. Figure 4-4 shows the resolution distrbution in each dataset.

We begin initial experiments with a controlled setting of FFHQ, which contains 70K images at 1024 resolution. From these, we construct a varied-size dataset by (1) using 256 resolution for all images (2) downsampling a 5K subset between 512-1024 (uniformly distributed) and (3) add 1k subset at full 1024. The last step enables us to judiciously compare to methods that are limited to synthesizing images at strict powers of 2. We refer to this mixture as FFHQ6k. We use the full 1024 dataset as ground-truth for evaluation metrics.

In the remaining domains, we push current generation results to higher resolution by scraping HR images from Flickr. In cases where a standard fixed-size dataset is available (LSUN Churches [271] and Mountains [173]), we select the additional HR images to approximately match the LR domain. Our final generators synthesize realistic details despite the majority of the training set being LR. For Birds and Churches, $> 92\%$ of the training set is at 256 resolution but our model maintains quality beyond 1024; for Mountains $> 98\%$ of training set at 1024 but our model can generate beyond 2048. These categories cover a range between objects (but without the strong alignment of FFHQ) and outdoor scenes.

4.4.1 Continuous multi-scale image synthesis

Qualitative examples. We show qualitative examples in Fig. 4-5. Our generated images preserve the fine details of HR structures, such as bricks, rocky slopes, feathers,

Table 4.1: **Any-resolution datasets and generator settings.** We build upon low-resolution (LR) datasets, and use it for fixed-size dataset pre-training. We add additional high-resolution (HR) images, of mixed resolutions. Note that the number of HR images is small (~2-8% of LR size). Patches of size p are sampled from both subsets during training, with average sampled scale $\mathbb{E}[s]$.

Domain	**Dataset**								**Generator**		
	Source	# Imgs		Resolutions					Config	Resolutions	
		LR	HR	LR	HR_{min}	HR_{med}	HR_{max}			p	$\mathbb{E}[s]$
Faces	FFHQ	70,000	6000	256	512	819	1024		R	256	458
Churches	LSUN & Flickr	126,227	6253	256	1024	2836	18,000		T	256	1061
Birds	Flickr	112,266	7625	256	512	1365	2048		T	256	585
Mountains	Flickr	507,495	9361	1024	2049	3168	12,431		T	1024	1823

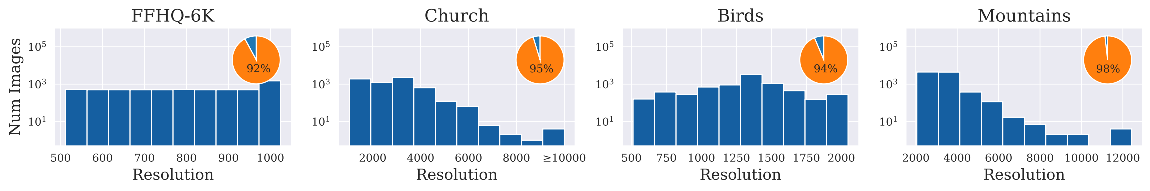


Figure 4-4: **Training set size distributions.** Histogram shows the size distribution of the HR images (y-axis in log scale); pie chart indicates the proportion of LR to HR images.

or hair. Pushing the inference resolution towards and beyond the higher resolutions of training images, we find that textured surfaces typically deteriorate first before edge boundaries deteriorate eventually (Fig. 4-6).

Patch-FID metric. Standard FID evaluates *global structure* by first downsampling all images to a common size of 299. By design, this ignores high-resolution details (and itself can cause artifacts [174]). Therefore, we propose a modification, which we dub 'patch-FID', to specifically evaluate *texture* synthesized at higher resolutions. Our patch-FID randomly resizes and crops patches from the HR dataset, and computes FID on real and generated patches, sampled at corresponding scales and locations. We use 50k patches, matching standard FID. By avoiding downsampling, our patch-FID is more sensitive to blurriness or artifacts at higher resolutions, resulting in larger absolute difference compared to standard FID at 1024 resolution. As a sanity check, when a full HR ground-truth is available, we find it is largely correlated to standard FID (see Table 4.2).

To summarize, to evaluate structure, we compute standard FID on images generated

Table 4.2: **Varied-size training and inference.** Random-resize MS-PE [37] performs varied-size synthesis, but assumes a fixed-size dataset. AnyCost-GAN handles varied training at powers of 2. Our method directly utilizes training images at any size, achieving better results by FID. († = copied from paper)

FFHQ6K	FID			pFID
	256	512	1024	random
MS-PE [37]†	6.75	30.41	–	–
Anycost [136]	4.24	5.94	6.47	18.39
Ours	3.34	3.71	4.06	2.96

Table 4.3: **Comparison to super-resolution using patch-FID (pFID).** For each domain, we compare our model to continuous-scale (LIIF) and fixed-scale super-resolution (Real-ESRGAN) models. Lower pFID suggests that our model can generate realistic details at high resolutions, not achievable with super-resolution alone.

	pFID (random)			
	FFHQ6K	Church	Bird	Mountain
LIIF [33]	22.93	83.88	30.19	23.10
Real-ESRGAN [254]	16.92	23.04	16.10	19.05
Ours	2.96	9.89	6.52	7.99

at specified resolutions, e.g., FID (256). To evaluate texture, we sample patches at random scales and locations and measure our patch-FID, which we denote as pFID (random). Lower numbers are better in both cases.

Alternative methods of multi-size training and generation. Our generator is encouraged to synthesize realistic high-resolution textures at training, even when the discriminator does not get to see the full image. While MS-PE [37] also enables continuous resolution synthesis, the discriminator learns only at a single resolution and the generator is not trained patch-wise. We find that this downsampling for the discriminator is detrimental to image quality at higher resolutions. Anycost-GAN[136] performs image synthesis at powers-of-two resolutions by adding additional synthesis blocks. For comparison, we modify it to handle a multi-size dataset by downsampling images to the nearest power of two and training each layer only on the valid image subset. Compared to Anycost-GAN, our model is more data-efficient, due to weight sharing for generation at multiple scales. Anycost-GAN learns a separate module for each increase in resolution, creating artifacts at higher resolutions when fewer HR training images are available and higher FID scores (Tab. 4.2). Additionally, Anycost-GAN increases the generator and discriminator size for synthesis at higher resolutions, whereas our model incurs a constant training cost, regardless of the inference scale.

Comparison to super-resolution. Most super-resolution methods require LR/HR image pairs, whereas there is no ground-truth HR counterpart to a LR image synthesized by our fixed-scale generator G_{fixed}. The teacher regularization encourages

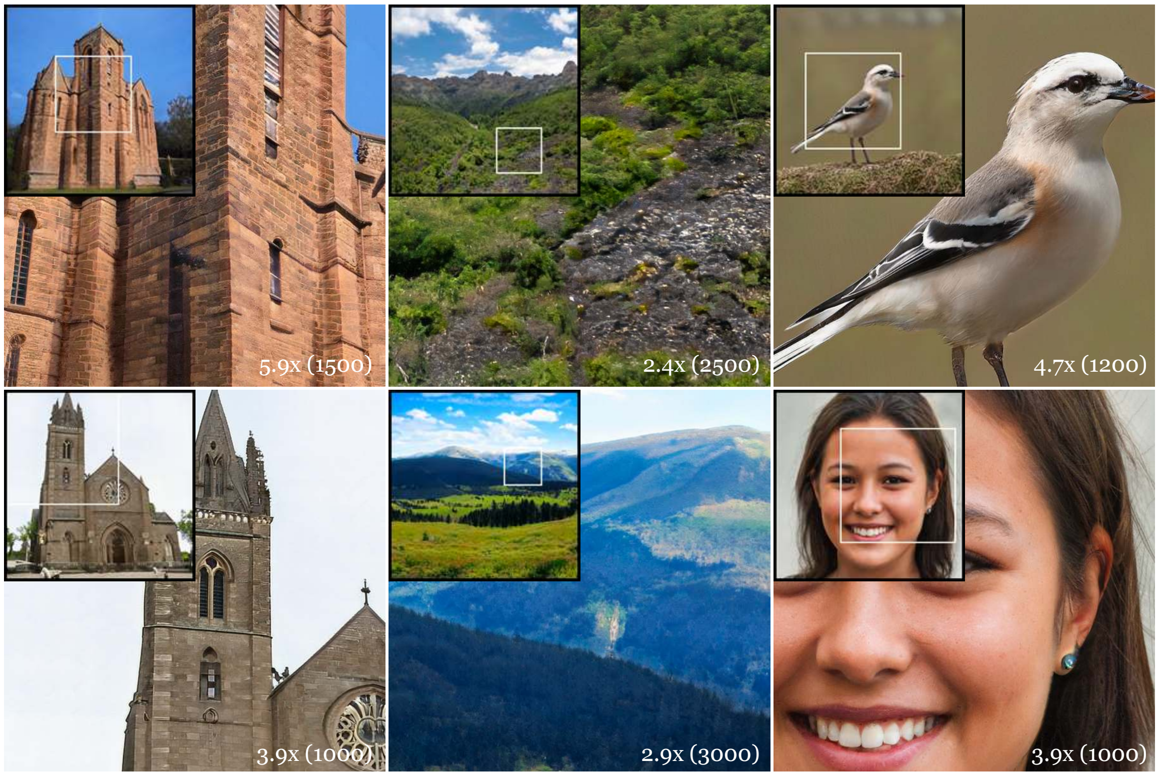


Figure 4-5: **Qualitative any-resolution generations.** The inset shows the entire generated, high-resolution images (between 1000-3000 resolution), with enlarged regions of interest outlined in the white box. Note that our model can render the image (or any sub-region) at any resolution.

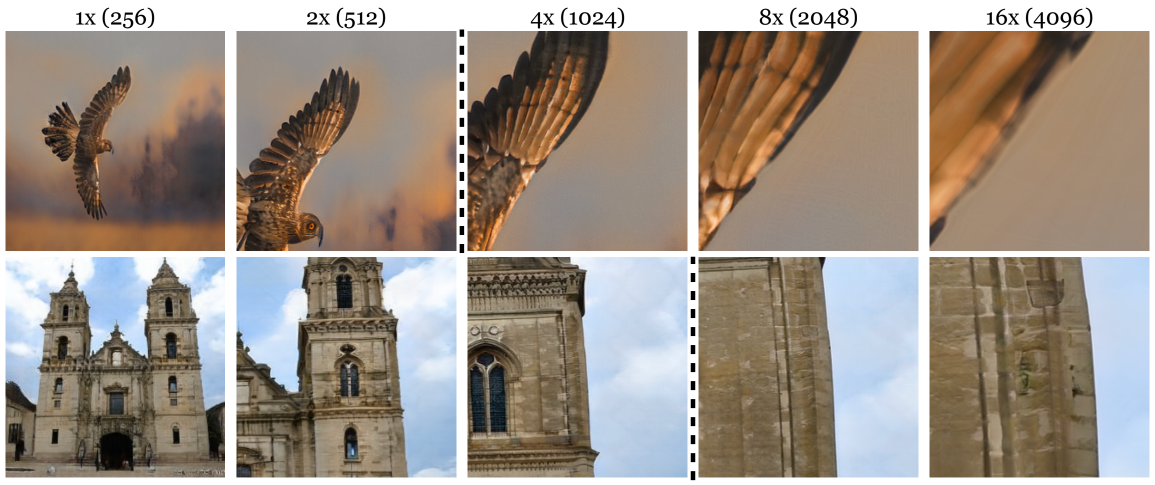


Figure 4-6: **Extrapolation limits.** We test the extrapolation capabilities of our model by specifying the inference scale s. Typically, textures such as bricks and feathers deteriorate first before edges degrade. The dotted line indicates when generation starts to exceed the average scale sampled in training $\mathbb{E}[s]$ (which is 585 for birds and 1061 for churches).

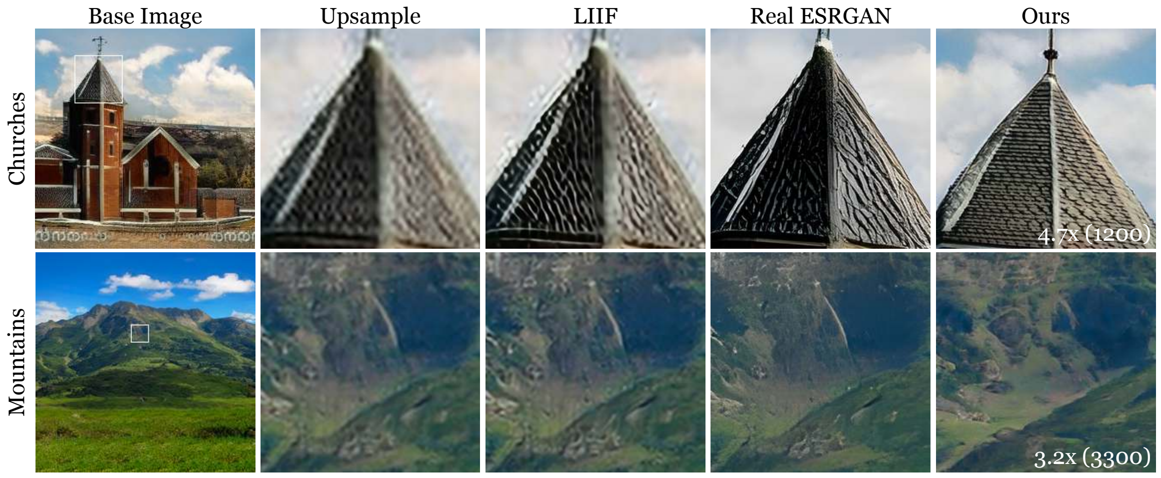


Figure 4-7: **Super-resolution comparisons.** Qualitative comparisons of Lanczos upsampling a patch from the base image (upsample), continuous (LIIF [33]) and fixed-factor (Real ESRGAN [254]) super-resolution models, and our trained model. LIIF tends to amplify artifacts from the base image (e.g. the JPEG artifacts around the church). While Real-ESRGAN is better at suppressing artifacts, it tends to overly smooth surfaces or synthesize grid-like textures (mountain). Our model is not a super-resolution model; it can add additional details to the low-resolution image but tolerates slight distortions in structure which are regularized with the teacher weight.

similarity between G_{fixed} and G's outputs, but unlike super-resolution, this supervision occurs at low-resolution, allowing variations in fine details. Figure 4-7 compares our model to super-resolution methods applied to the output of G_{fixed}. Our method produces much sharper details than LIIF [33], a recent continuous-scale super-resolution technique, and cleaner images than the state-of-the-art Real-ESRGAN [254]. The latter is a fixed-resolution model, so we run it iteratively until exceeding a target resolution, and then Lanczos downsample the result to the target size. Real-ESRGAN's outputs are either overly smooth, or exhibit grid-like artifacts. Our method generates realistic textures based on the low-resolution output of G_{fixed} and reaches a lower pFID (Tab. 4.3).

4.4.2 Model variations

Using the full high-resolution FFHQ dataset as a benchmark, we investigate individual components of our architecture and training process. We train each model variation for 5M images and record metrics from the best FID@1024 checkpoint. We only

Table 4.4: **Multisize training.** Downsampling or upsampling all images to a common size, or using only the subset of the largest images, worsens FID compared to our training strategy. (*) indicates our default setting.

	FID			pFID
	256	512	1024	random
Resize down to 512	3.31	4.11	19.18	26.83
Resize up to 1024	3.46	13.43	4.86	6.65
Train 1024 subset	3.46	12.41	4.67	5.43
Multisize training (*)	3.37	4.41	4.47	4.28

Table 4.5: **Number of HR images.** Our method is robust to a wide range of HR images, even when only 1K images at HR are available (<2% of the full ground-truth dataset).

	FID			pFID
	256	512	1024	random
1k (1.4%)	3.43	5.13	4.38	3.73
5k (7.1%) (*)	3.36	4.97	4.54	3.48
10k (14.2%)	3.46	4.96	4.65	3.54
70k (100%)	3.42	4.88	4.52	3.42

report quantitative metrics in the main paper and refer to the supplemental for further evaluations and visual comparisons.

Impact of teacher regularization. Our full model uses an "inverse" teacher regularizer to encourages a downsampled HR patch to match the low-resolution teacher as described in 4.3.2. We also explored a variant with a "forward" teacher loss, in which the generated patch is encouraged to match the *upsampled* teacher output. This variation is qualitatively inferior and blurs details; it has worse FID at higher resolutions (see supplemental for details and visuals). Removing the teacher altogether improves pFID but degrades FID. Qualitatively, the generated patches diverge significantly from the fixed-size global image. We hypothesize that the global change in structure negatively impacts overall image quality, causing global FID metrics to increase, but this cannot be captured from evaluating patches alone. We found $\lambda_{\text{teacher}} = 5$ to provide the best balance between global and local image quality, but we observe minimal differences in FID and pFID for other values, evidence that the model can tolerate a range of values for this parameter. See supplemental for a parameter sweep with full scores.

Removing scale conditioning degrades quality. We inject the scale information to intermediate layers of the generator through scale-conditioning. Adding this improves FID@1024 from 4.88 to 4.47, and pFID from 4.67 to 4.28.

Multi-size training improves fidelity at all scales. Our multi-size data pipeline lets our model learn to synthesize at continuous scales, which is a strictly more challenging than learning at a fixed scale. In Table 4.4, we investigate to what extent learning from images of varied sizes offers benefits over fixed-scale training on a smaller dataset. Visual comparisons can be found in supplemental. In a first alternative, using the same FFHQ-6K dataset, we resize all images down to 512 and train the model to generate patches at 512 × 512. In this case, the model performs well up to 512 scale, but does not generalize beyond (e.g., 1024) since it cannot exploit the information lost in downsampling. In two other variants, we train models for 1024 resolution in the first case by upsampling all images up to 1024, in the second by keeping only the 1K subset of images at 1024 resolution. Both variants are trained to output images specifically at 1024 resolution. The former approach (upsampling) increases blurriness. In the latter, FID@1024 remains worse than that of our multi-size training, which can take advantage of more data despite most of it being *smaller* than 1024.

Impact of number of HR images Due to the patch-based training procedure, we find that our model can be trained with a small fraction of HR images, compared to the 70k LR images in the dataset. In Table 4.5, we use progressively larger subsets of HR images: 1k, 5k, 10k. We found that the FID scores are largely similar (within 0.3) to using the entire 70k HR images. However, training with 1k or fewer HR images shows evidence of divergence during training (see supplemental), but stabilizes by 5k HR images, For the remaining domains, we collect roughly 5K-10K images to construct the HR dataset.

4.4.3 Properties of multi-scale generation

Correcting artifacts from low resolution. Because our model is not directly trained with corresponding LR and HR image pairs, we find that there can be small distortions between the upsampled base image and the HR generation from the same latent code. In some cases, this can be a desirable property (Fig. 4-8). For instance, the base generator on the birds dataset can struggle in synthesizing the eye of the bird, which is less apparent at low resolutions, but more salient at high resolution. Consequently, our HR generation will add the missing eye, and also synthesizes additional feather and beak details. In the churches domain, because the LR and HR

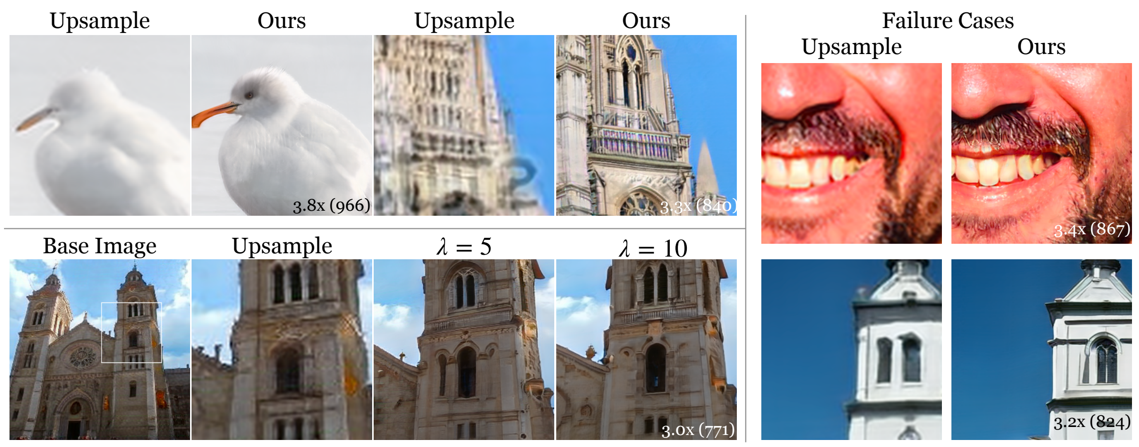


Figure 4-8: **Model properties and failure cases.** As fine details can be more difficult to learn at low resolution, our model is capable of adding corrections when generating at higher resolutions. In the case of inconsistencies between the LR and HR data sources, the model deletes patterns that are not present in the HR dataset (*e.g.* watermarks and compression artifacts), influenced by the teacher regularization weight. Failure cases include biases towards circular or ring-like structures.

datasets are collected separately, we find that the synthesized watermarks and JPEG artifacts at the base resolution disappear at higher resolution, because the HR dataset we used is of higher quality and does not have any watermark. The similarity between the LR and HR generations can be tuned using λ_{teacher} during training.

Failure Cases. Our model tends to inherit the artifacts from StyleGAN3, such as a centered front tooth in FFHQ. In instances in which the base resolution image contains uneven surfaces, the model may fail to fully mitigate them at higher resolutions. These artifacts are often subtle at the low resolution, but become more apparent when upsampling the base image or generating at a larger target scale. In some cases, our model also has a tendency to generate "watery" circular or ring-like artifacts (Fig. 4-8).

4.5 Conclusion

We propose an image synthesis approach that can train on images of varied resolution and perform inference at continuous resolutions. This lifts the fixed-resolution requirement of prior generative models, which discard higher-resolution details. To do this, we train a generator jointly on a low-resolution dataset to learn global structure, and on patches from the varied-size dataset to learn details. At inference time, we can synthesize an image at any resolution by supplying the appropriate coordinate grid and scale factor to the generator. By using training images at their native resolutions and a single model for continuous-resolution synthesis, our method can efficiently leverage information present in only a handful of high-resolution images to complement a large set of low-resolution images. This approach enables high-resolution synthesis without a larger generator or large dataset of fixed-size, high-resolution images.

Chapter 5

Unbounded Persistent Landscapes with 3D Camera Control

With minor modifications from:

Persistent Nature:
A Generative Model of Unbounded 3D Worlds

Lucy Chai, Richard Tucker, Zhengqi Li, Phillip Isola, Noah Snavely;

CVPR 2023.

The method presented in this chapter again builds coordinate transformations into our synthesis model, but now focuses on 3D spatial transformations that allow us to navigate within a 3D world representation, rather than the 2D transformations used in any-resolution image synthesis. We design a generator that creates unbounded landscape scenery, where the generated images are projections of the 3D world according to the camera's position and orientation. A key feature of our architecture is the concept of "persistence," which means that the underlying scene does not change or morph as the camera moves, which is a common challenge in video generation techniques. Trained on 2.5D supervision from single-view landscape datasets without any 3D ground truth, the resulting model enables arbitrary camera motion through large-scale natural scenery with circular flight patterns, combining the strengths of recent models for unbounded auto–regressive prediction and persistent 3D generation.

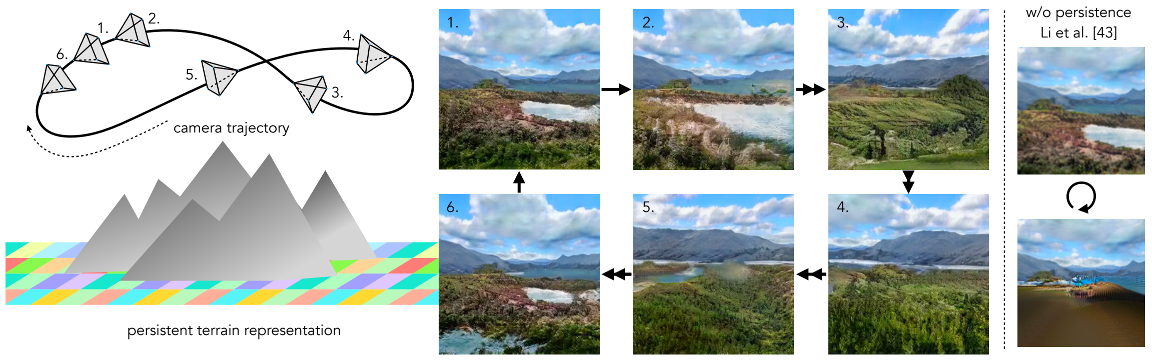


Figure 5-1: **Persistent, unbounded synthesis.** Our approach enables unconditional synthesis of unbounded 3D nature scenes with a persistent scene representation (**left**), using a scene layout grid representing a large-scale terrain model (depicted above as the checkered ground plane). This representation enables us to generate arbitrary camera trajectories, such as the six numbered views shown along a cyclic camera path (**center**). The *persistence* inherent to our representation stands in contrast to prior auto-regressive methods [131] that do not preserve consistency under circular camera trajectories (**right**); while the two images shown on the right are at the start and end of a cyclic path, the terrain depicted is completely different. Our method is trained solely from unposed, single-view landscape photos.

5.1 Introduction

Generative image and video models have achieved remarkable levels of realism, but are still far from presenting a convincing, explorable world. Moving a virtual camera through these models—either in their latent space [77, 98, 209, 20] or via explicit conditioning [112]—is not like walking about in the real world. Movement is either very limited (for example, in object-centric models [27]), or else camera motion is unlimited but quickly reveals the lack of a persistent world model. Auto-regressive 3D synthesis methods exemplify this lack of persistence [137, 131]; parts of the scene may change unexpectedly as the camera moves, and you may find that the scene is entirely different when returning to previous positions. The lack of spatial and temporal consistency can give the output of these models a strange, dream-like quality. In contrast, machines that can generate unbounded, persistent 3D worlds could be used to develop agents that plan within a world model [75], or to build virtual reality experiences that feel closer to the natural world, rather than appearing as ephemeral hallucinations [131].

We therefore aim to develop a unconditional generative model capable of generating unbounded 3D scenes with a persistent underlying world representation. We want synthesized content to move in a way that is consistent with camera motion, yet we should also be able to move arbitrarily far and still generate the same scene upon returning to a previous camera location, regardless of the camera trajectory.

To achieve this goal, we model a 3D world as a *terrain* plus a *skydome*. The terrain is represented by a *scene layout grid*—an extendable 2D array of feature vectors that acts as a map of the landscape. We 'lift' these features into 3D and decode them with an MLP into a radiance field for volume rendering. The rendered terrain images are super-resolved and composited with renderings from the skydome model to synthesize final images. We train using a layout grid of limited size, but can extend the scene layout grid by any desired amount during inference, enabling unbounded camera trajectories. Since our underlying representation is persistent over space and time, we can fly around 3D landscapes in a consistent manner. Our method does not require multiview data; each part of our system is trained from an unposed collection of single-view images using GAN objectives.

Our work builds upon two prior threads of research that tackle generating immersive worlds: 1) generative models of 3D data, and 2) generative models of infinite videos. Along the first direction are generators of meshes, volumes, radiance fields, etc (e.g., [162, 27, 180]). These models represent a consistent 3D world by construction, and excel at rendering isolated objects and bounded indoor scenes. Our work, in contrast, tackles the challenging problem of generating large-scale *unbounded* nature scenes. Along the second direction are methods like InfiniteNature [137, 131], which can indeed simulate visual worlds of infinite extent. These methods enable unbounded scene synthesis by predicting new viewpoints auto-regressively from a starting view. However, they lack a persistent world representation; content may change when revisited.

Our method aims to combine the best of both worlds, generating boundless scenes (unlike prior 3D generators) while still representing a persistent 3D world (unlike prior video generative models). In summary:

- We present an unconditional 3D generative model for unbounded nature scenes with a persistent world representation, consisting of a terrain map and skydome.

- We augment our generative pipeline to support camera extrapolation beyond the training camera distribution by extending the terrain features.
- Our model is learned entirely from single-view landscape photos with unknown camera poses.

5.2 Related Work

Image and view extrapolation. Pioneering work by Kaneva *et al.* [103] proposed the task of infinite image extrapolation by using a large image database to perform classical 2D image retrieval, stitching, and rendering. More recently, various learning-based 2D image inpainting [78, 272, 273, 139, 286, 228, 130, 195] and outpainting [255, 268, 232, 18, 134, 35] methods have been developed. These methods fill in missing image regions or expand the field of view by synthesizing realistic image content that is coherent with the partial input image. Beyond 2D, prior work has explored single-view 3D *view extrapolation*, often by applying 2D image synthesis techniques within a 3D representation [260, 212, 190, 88, 191, 128, 100]. However, these methods can only extrapolate content within a very limited range of viewpoints.

Video generation. Video generation aims to synthesize realistic videos from different types of input. Unconditional video generation produces long videos often from noise input [240, 160, 59, 222, 143, 20, 65], while conditional video generation generates sequences by conditioning on one or a few images [249, 248, 256, 247, 87, 125, 249, 57, 44, 269, 275, 117], or a text prompt [86, 219]. However, applying these ideas in 3D requires supervision from multi-view training data, and cannot achieve persistent 3D scene content at runtime, since there is no explicit 3D representation. Some recent work preserves global scene consistency via extra 3D geometry inputs such as point clouds [149] or voxel grids [76]. In contrast, our method synthesizes both the geometry and appearance of an entire world from scratch using a global feature representation to achieve consistent generated content.

Generative view synthesis. Novel view synthesis aims to produce new views of a scene from single [32, 239, 166, 238, 211, 260, 101, 212, 118, 191, 270] or multiple

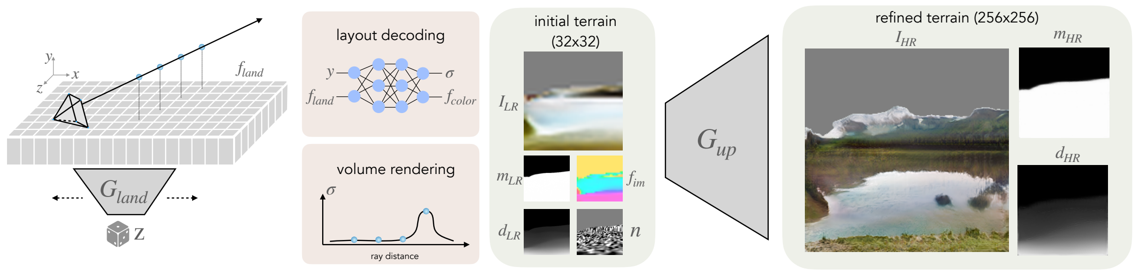


Figure 5-2: **Overview of scene layout decoding.** The layout generator G_{land} samples a random latent code to produce a 2D scene layout grid f_{land} representing the shape and appearance of a terrain map, and which can be spatially extended using a grid of latent codes (see § 5.3.2). To render an image from a given camera, sampled points along camera rays passing over the feature plane are decoded via an MLP into a color feature f_{color} and density σ, which are then volume rendered. This produces a low-resolution image, mask, depth, image features, and a projected noise pattern, which are provided to a refinement network G_{up} to produce final image, mask, and depth outputs.

image observations [127, 289, 155, 58, 36, 145, 189, 156, 250, 159, 9, 197, 208] by constructing a local or global 3D scene representation. However, most prior methods can only interpolate or extrapolate a limited distance from the input views, and do not possess a generative ability.

On the other hand, a number of generative view synthesis methods have been recently proposed utilizing neural volumetric representations [162, 203, 165, 47, 164, 71, 27, 186, 223]. These methods can learn to generate 3D representations from 2D supervision, and have demonstrated impressive results on generating novel objects [180], faces [71, 169, 27, 43], or indoor environments [187, 47]. However, none of these methods can generate unbounded outdoor scenes due to lack of multi-view data for supervision, and due to the larger and more complex scene geometry and appearance that is difficult to model with prior representations. In contrast, our approach can generate globally consistent, large-scale nature scenes by training solely from unstructured 2D photo collections.

Our work is particularly inspired by recent perpetual view generation methods, including InfiniteNature [137] and InfiniteNature-Zero [131], which can generate unbounded fly-through videos of natural scenes, and are trained on nature videos or photo collections. However, these methods generate video sequences in an auto-regressive manner, and therefore cannot achieve globally consistent 3D scene content. Our approach instead adopts a global scene representation that can be trained to generate consistent-by-construction and realistic novel views spanning large-scale scenes. Concurrent works for scene synthesis InfiniCity [135] and SceneDreamer[34] leverage birds-eye-view representations, while SceneScape [61] builds a mesh representation from text.

5.3 Method

Our scene representation for unbounded landscapes consists of two components, a *scene layout grid* and a *skydome*. The scene layout grid models the landscape terrain, and is a 2D grid of features defined on a "ground plane." These 2D features are intended to describe both the height and appearance content of the terrain, representing the full 3D scene — in fact, we decode these features to a 3D radiance field, which can then be rendered to an image (§5.3.1). To enable camera motion beyond the training volume, we spatially extend the 2D feature grid to arbitrary sizes (§5.3.2). Because it is computationally expensive to generate and volume render highly detailed 3D content at the scale we aim for, we use an image-space refinement network that adds additional texture detail to rendered images (§5.3.3).

The second scene component is a *skydome* (§5.3.4), which is a spherical (panoramic) image intended to model very remote content, such as the sun and sky, as well as distant mountains. The skydome is generated to harmonize with the terrain content described by the scene layout grid.

All the stages of our approach are trained with GAN losses (§5.3.5). In what follows, we use the 3D coordinate convention that the ground plane is the xz-plane, and the y-axis represents height above or below this plane. Generally, the camera used to view the scene will be positioned some height above the ground.

5.3.1 Scene layout generation and rendering

To represent a distribution over landscapes, we take a generative approach following the layout representation of GSN [47]. First, a 2D scene layout grid is synthesized from a sampled random noise code z passed to a StyleGAN2 [110] generator G_{land}. This creates a 2D feature grid f_{land}, which we bilinearly interpolate to obtain a 2D function over spatial coordinates x and z:

$$f_{\text{land}}(x, z) = \text{Interpolate}(G_{\text{land}}(z), (x, z)) \tag{5.1}$$

To define a full 3D scene, we need a way to compute the content at any 3D location (x, y, z). We define a multi-layer perceptron M that takes a scene grid feature, as well as the height y of the point at which we want to evaluate the scene content. The outputs of M are the 2D-to-3D lifted feature f_{color} and the density σ at point (x, y, z):

$$f_{\text{color}}, \sigma = M(f_{\text{land}}(x, z), y). \tag{5.2}$$

In this way, the 2D scene layout grid determines a radiance field over all 3D points within the bounds of the grid[270, 47, 206]. That is, feature vectors in the grid encode not just appearance information, but also the height (or possibly multiple heights) of the terrain at their ground location.

To render an image from a desired camera pose, we cast rays $\mathbf{r}$ from the camera origin through 3D space, sample points (x, y, z) along them, and compute f_{color} and σ at each point. We then use volume rendering to composite f_{color} along each ray into projected 2D image features f_{im}, a disparity image d_{LR}, and a sky segmentation mask m_{LR}. We form an initial RGB image of the terrain, I_{LR}, via a learned linear projection P of these image features. This process is depicted in the left half of Fig. 5-2, and is defined as:

$$\begin{aligned} f_{\text{im}}(\mathbf{r}) &= \sum_{i=1}^{N} w_i f_{\text{color},i}, \quad & d_{\text{LR}}(\mathbf{r}) &= \sum_{i=1}^{N} w_i d_i, \\ m_{\text{LR}}(\mathbf{r}) &= \sum_{i=1}^{N} w_i, & I_{\text{LR}} &= P f_{\text{im}}, \end{aligned} \tag{5.3}$$

where $i \in \{1..N\}$ refers to the index of each sampled point along ray $\mathbf{r}$ in order of increasing distance from the camera, d_i is the inverse-depth (disparity) of point i, and weights w_i are determined from the volume rendering equations used in NeRF [156] (see supplemental).

We intend the mask m_{LR} to distinguish sky regions (which will be empty and filled later using the skydome) from non-sky regions, and achieve this by training using segmented real images in which color and disparity for sky pixels are replaced with zero. Since to achieve zero disparity all weights along a ray must be zero (which also results in a zero-valued color feature), this approach encourages the generator to omit sky content. However, while we find that the model indeed learns to generate transparent sky regions, land geometry can also become partially transparent. To counter this, we penalize visible decreases in opacity along viewing rays using finite differences of opacity α:

$$\mathcal{L}_{\mathrm{transparent}}(\mathbf{r}) = \sum_{i=2}^{N} w_i \frac{\max(\alpha_{i-1} - \alpha_i, 0)}{\delta_i}. \tag{5.4}$$

5.3.2 Layout extension

While G_{land} creates a fixed-size feature grid, our objective is to generate geometry of arbitrary size, enabling long-distance camera motion at inference time. Hence, we devise a way to *extend* the feature grid in the x and z dimensions. We illustrate this process in Fig. 5-3, where we first sample noise codes z in a grid arrangement, where each z generates a 2D layout feature grid of size $H \times W$. To obtain a smooth transition between these independently sampled layout features, we generalize the image interpolation approach from SOAT (StyleGAN of all Trades) [40] to two dimensions. We operate on 2×2 sub-grids and blend intermediate features from each layer of the generator as follows:

$$\begin{aligned} f_{k,l+1} &= G_l(f_l, z_k); \quad k = \{00,\ 01,\ 10,\ 11\} \\ f_{l+1} &= \sum_{k=\{00,01,10,11\}} \beta_k(x,z) f_{k,l+1}. \end{aligned} \tag{5.5}$$

For each of the four corner anchors k, we construct the modulated feature $f_{k,l+1}$ by applying G_l (the l-th layer of G_{land}) in a fully convolutional manner over the

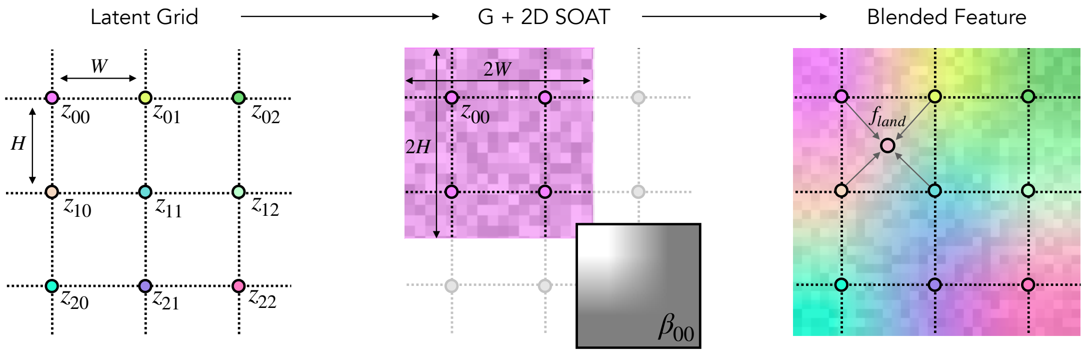


Figure 5-3: **Layout extension procedure.** To extend the layout at inference time, we sample noise codes z in a grid arrangement. To smoothly transition between adjacent feature grids, we use the SOAT (StyleGAN of All Trades) procedure [40] in 2D. Operating on a 2×2 sub-grid, we apply each generator layer four times in fully convolutional manner over the entire sub-grid, each time conditioned on a different corner latent code z, before multiplying by bilinear blending weights. This process is repeated for each layer of the generator and each sub-grid. Each 2×2 sub-grid produces a $2H \times 2W$ feature grid, and sub-grids are blended together in an overlapping fashion to obtain an extended feature grid f_{land} of arbitrary spatial size.

entire sub-grid. We then interpolate between the four feature grids using bilinear interpolation weights $\beta_k(x, z)$. By stitching these 2×2 sub-grids in an overlapping manner, we can obtain a scene layout feature grid of arbitrary size to use as f_{land}. Additional details are provided in the supplemental.

5.3.3 Image refinement

Due to the computational cost of volume rendering, training the layout generator at higher resolutions becomes impractical. We therefore use a refinement network G_{up} to upsample the initial generated image I_{LR} to a higher-resolution result I_{HR}, while adding textural details (Fig. 5-2-right). We use a StyleGAN2 backbone for G_{up}, replacing the earlier feature layers with feature output f_{im} and the RGB residual layers with a concatenation of I_{LR}, d_{LR}, and m_{LR}. To encourage the refined terrain image I_{HR} to be consistent with the sky mask, the network also predicts a refined disparity map and sky mask for compositing with the skydome (see §5.3.4):

$$I_{\text{HR}}, d_{\text{HR}}, m_{\text{HR}} = G_{\text{up}}(f_{\text{im}}, I_{\text{LR}}, d_{\text{LR}}, m_{\text{LR}}). \tag{5.6}$$

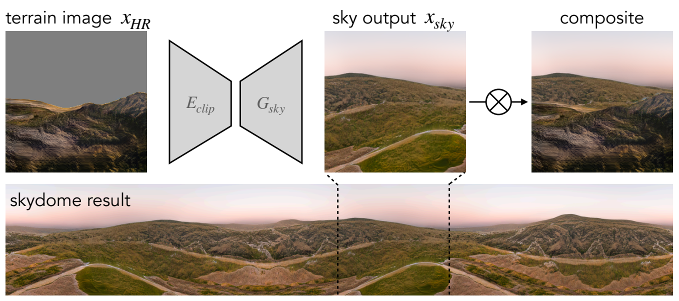


Figure 5-4: **Skydome generator.** Conditioned on the terrain image, the skydome generator G_{sky} synthesizes distant content (e.g., sky pixels and remote mountains) that is consistent with the generated terrain using encoder E_{clip}. G_{sky} uses cylindrical coordinates to produce a panoramic image.

We compute a reconstruction loss between the initial and refined disparity and mask outputs, and penalize G_{up} for producing gray sky pixels in I_{HR} outside the predicted mask m_{HR}. Please see the supplemental for more details.

For fine texture details, StyleGAN2 also uses layer-wise spatial noise in intermediate generator layers (in addition to the global latent z). Using a fixed 2D noise pattern results in texture 'sticking' as we move the camera [111], but resampling it every frame reduces spatial coherence and removing it entirely results in gridding artifacts. To avoid these issues and improve spatial consistency, we replace the 2D image-space noise with projected 3D world-space noise, where the noise input to G_{up} is the projection of samples from a grid of noise, n. This noise pattern is drawn from a standard Gaussian distribution defined on the ground plane at the same resolution of the layout features, which is then lifted into 3D and volume rendered along each ray $\mathbf{r}$:

$$n(\mathbf{r}) = \sum_{i=1}^{N} w_i n(x, z). \tag{5.7}$$

5.3.4 Skydome

We model remote content (sky and distant mountains) separately with a skydome generator G_{sky} (Fig.5-4). This generator follows the StyleGAN3 architecture [111], with a mapping network and synthesis network conditioned on cylindrical coordinates [25]. We adapt it by conditioning on the terrain output: we encode terrain images I_{HR} using the pretrained CLIP image encoder E_{clip} [183], and concatenate this to the style-code output of the mapping network as input into G_{sky}:

$$I_{\text{sky}} = G_{\text{sky}}(\text{concat}(E_{\text{clip}}(I_{\text{HR}}), \text{mapping}(z))). \tag{5.8}$$

Conditioning on the foreground terrain image encourages the skydome generator to generate a sky that is consistent with the terrain content. This model trains on single-view landscape images but can produce a full panorama at inference-time by passing in coordinates that correspond to a 360° cylinder. The skydome is rendered to an individual camera viewpoint using camera ray directions, giving the skydome image I_{dome} which is then composited with the terrain image using the sky mask:

$$I_{\text{full}} = I_{\text{HR}} \odot m_{\text{HR}} + I_{\text{dome}} \odot (1 - m_{\text{HR}}). \tag{5.9}$$

5.3.5 Training

We train the layout generator (rendering at 32x32), refinement network (upsampling to 256x256), and skydome generator separately. To train the refinement network, we operate on outputs of the layout generator, freezing the weights of that model. For the skydome generator, we train using real landscape images, and apply it only to the outputs of the refinement network at inference time. We follow the StyleGAN2 objective [110], with additional losses for each training stage, architecture, and hyperparameters provided in the supplemental.

Dataset and camera poses. We train on LHQ [221], a dataset of of 90K unposed, single-view images of natural landscapes. A number of LHQ images contain geometry that is not amenable to "flying", such as a landscape pictured through a window, or a closeup of trees. Therefore, we perform a filtering process on LHQ prior to training (see supplemental). We also obtain auxiliary outputs – disparity and sky segmentation – using the pretrained DPT [184] model. Disparity and sky segmentation are used to construct the real image distribution in the GAN training phases.

After filtering, we use 56,982 images for training, and augment with horizontal flipping. During training we also need to sample camera poses. Prior 3D generators[47, 26, 27, 203, 169, 71] either use ground-truth poses from a simulator, or assume an object-centric camera distribution in which the camera looks at a fixed origin from some radius. Because our dataset lacks ground truth poses, we first sample a bank of training poses uniformly across the layout feature grid with random small height offsets, and rotate such that the near half of the camera view frustum falls entirely within the layout grid. Since the aerial layout should not be specific to any given camera pose, we generate f_{land} without any camera pose information, and then adopt the sampling scheme from GSN[47] which samples a camera pose from the initial training pose bank proportional to the inverse terrain density at each camera position, to avoid placing the camera within occluding geometry.

5.4 Experiments

Given its persistent scene representation and the extensibility of the its layout grid, our model enables arbitrary motion through a synthesized landscape, including long camera trajectories. We show sample outputs from our model under a variety of camera movements (§ 5.4.1); present qualitative and quantitative comparisons with alternate scene representations, including auto-regressive prediction models and unconditional generators defined for bounded or object-centric scenes (§ 5.4.2); and investigate variations of our model to evaluate design decisions (§ 5.4.3).

5.4.1 Persistent, unbounded scene synthesis

Figure 5-5 shows example landscapes generated by our model with various camera motions. As the camera moves (by rotating and/or translating) the generated imagery changes in a way that is consistent with the underlying geometry, *e.g.* hills move across the image or become closer. Extending the generated aerial feature grid allows us to place the camera *outside* the distribution of training camera poses, while maintaining both geometric and stylistic consistency. As illustrated in Figure 5-1 and our project page, the persistent and extendable layout features enables synthetic ‘flights’ over large distances that can also return to a consistent starting point.

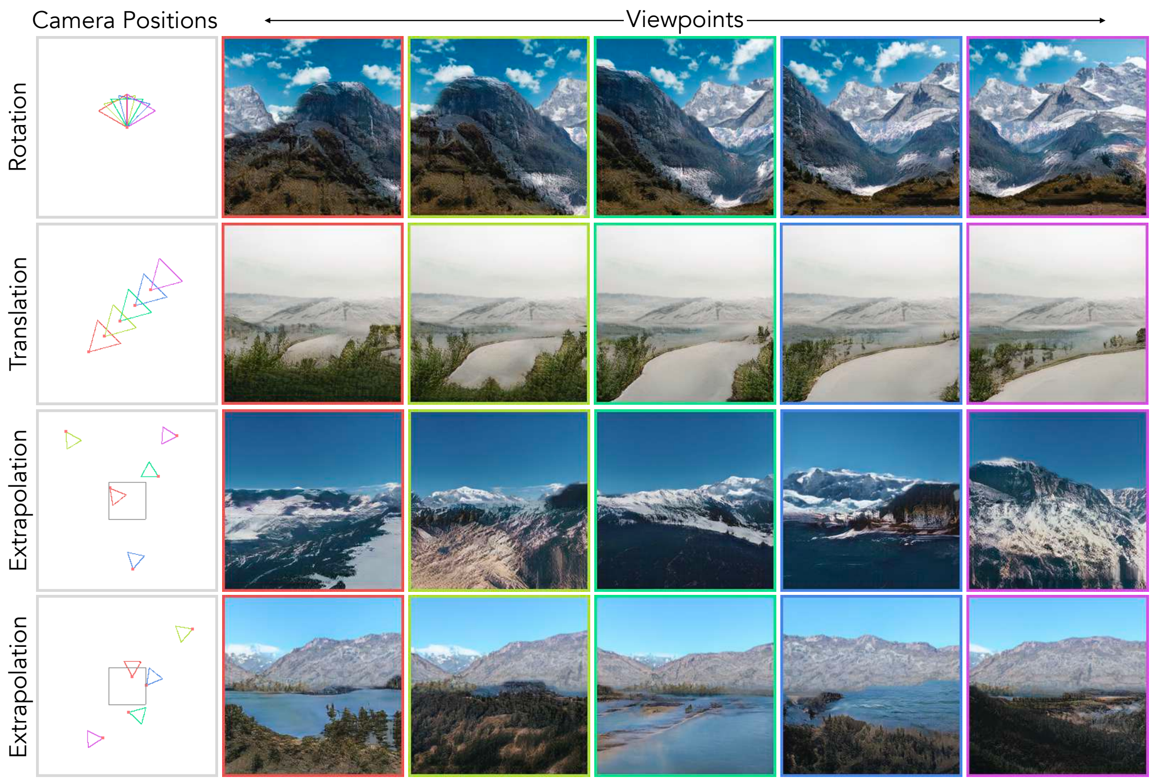


Figure 5-5: **Visualization of nearby and extrapolated camera motion.** Each row shows a set of sampled viewpoints, shown in an overhead view in the first column, and the corresponding rendered images in the other columns. Our model enables 3D-consistent view synthesis, visible under rotating or translating camera trajectories. We can also extrapolate the layout features at inference time, enabling camera motions outside of the training camera distribution (shown as a black square in the last two rows) with a consistent scene style.

5.4.2 Comparing scene representations

We compare our model with three state-of-the-art methods. InfiniteNature-Zero is an auto-regressive method that, given an initial frame, generates successive frames sequentially by warping each image to the next based on depth [131]. It allows for unbounded camera trajectories, but has no persistent world model. GSN [47] and EG3D [27] are unconditional generative models: GSN uses a layout feature grid which is also the basis of our model, but focuses on bounded indoor scenes with ground-truth camera pose trajectories, while EG3D uses a tri-plane representation and primarily focuses on objects and portraits. These methods have persistent world models (feature grid and tri-plane representation) but do not allow for unbounded trajectories.

Quantitative comparisons. We evaluate image quality using FID [82], and multi-

Model	**Persistent**	**Unbounded**	**FID**	**Consistency**	
			C_{forward}	1-step	cycle
Inf Nat Zero [131]	✗	✓	28.15	**1.84**	3.94
Ours (128px)	✓	✓	**26.09**	2.12	**0.00**

Table 5.1: **Comparison with InfiniteNature-Zero.** Using camera motions from InfiniteNature-Zero, we evaluate image quality as FID on 5K images after moving 100 steps forward (C_{forward}), one-step consistency as the L1 error when backwards warping one camera step, and cycle consistency as the L1 error between the original frame and the result after a pair of forward/backward steps. InfiniteNature-Zero is more consistent for a single step, but it has non-zero cyclic consistency error, and image quality degrades after repeated model applications. L1 values are multiplied by 100 throughout.

Model	**Persistent**	**Unbounded**	**FID**			**Consistency**
			C_{train}	C_{forward}	C_{random}	
GSN [47]	✓	✗	29.95	50.22	45.48	12.80
EG3D [27]	✓	✗	**9.85**	30.17	32.08	**3.01**
Ours	✓	✓	21.42	**26.67**	**23.39**	3.56

Table 5.2: **Quantitative comparison to unconditional GANs.** We evaluate image quality as FID on 5K images on (a) training camera poses C_{train}, (b) forward motion C_{forward} (See Table 5.1), (c) random camera poses C_{random}. One-step consistency error is measured as the L1 error when backwards warping the result after one camera step to the initial frame, multiplied by 100. Once outside the training pose distribution our model generates better images than other methods, with consistency close to that of EG3D.

view consistency using photometric error. To compare with InfiniteNature-Zero (Table 5.1), we initialize with an image and depth map from our model, move the camera forwards using a forward motion trajectory from InfiniteNature-Zero, and evaluate image quality at a distance of 100 forward steps. Our model attains better FID, showing that it does not suffer from image degradation due to successive applications of an auto-regressive model. To compute one-step consistency error, we generate a new frame at a position equivalent to one forward step of InfiniteNature-Zero, warp it back to the original camera position using depth, and compute L1 error with the original frame in the overlapping region. Because InfiniteNature-Zero uses explicit warping as part of its model, it can achieve better one-step consistency, whereas our 2D upsampling operation is more susceptible to geometric inconsistency. We measure cyclic consistency error as the L1 error between the initial frame to the result after a step forward and back. Because InfiniteNature-Zero lacks a persistent global representation, it has non-zero cyclic consistency error, whereas our model is fully consistent with zero cyclic consistency error.

To compare with the unconditional generative models GSN and EG3D, we compute FID on sets of output images corresponding to different distributions of camera positions: camera poses used in training which are intended to overlap with the layout, camera poses 100 steps forward from these mimicking InfiniteNature-Zero trajectories, and a uniform distribution of randomly oriented cameras over the layout grid. As seen in Table 5.2, GSN is the least successful method when applied to this domain. EG3D generates high-quality images at training camera poses, but tends to represent the scene as floating nearby clouds with planar mountains at the edges of the volume (incorrect geometry). Our method generalizes better to new camera positions. GSN has the highest one-step consistency error, while the consistency error of our model is close to that of EG3D (which relies less on 2D upsampling). In the supplemental, we experiment with an alternative architecture that builds on extendable triplane units with lower consistency error and faster rendering speed.

Qualitative comparisons. In Fig. 5-6 we show example outputs of each model over forward-moving and rotating trajectories. Due to its auto-regressive nature, the quality of InfiniteNature-Zero's output degrades somewhat as the camera trajectory becomes longer. A more serious limitation is that, trained only on forward movement, it is unable to synthesize plausible views under camera rotation. GSN and EG3D also struggle with long camera trajectories, producing unrealistic outputs as the cameras approach the spatial limits of the training camera distribution. In the case of GSN applied to our setting, the results contain flickering and grid-like artifacts, which our projected noise (§ 5.3.3) mitigates.

5.4.3 Model variations

To investigate individual components of our model, we separately evaluate variations of the layout generator and refinement network.

Layout generator. The resolution of the scene layout grid and the number of

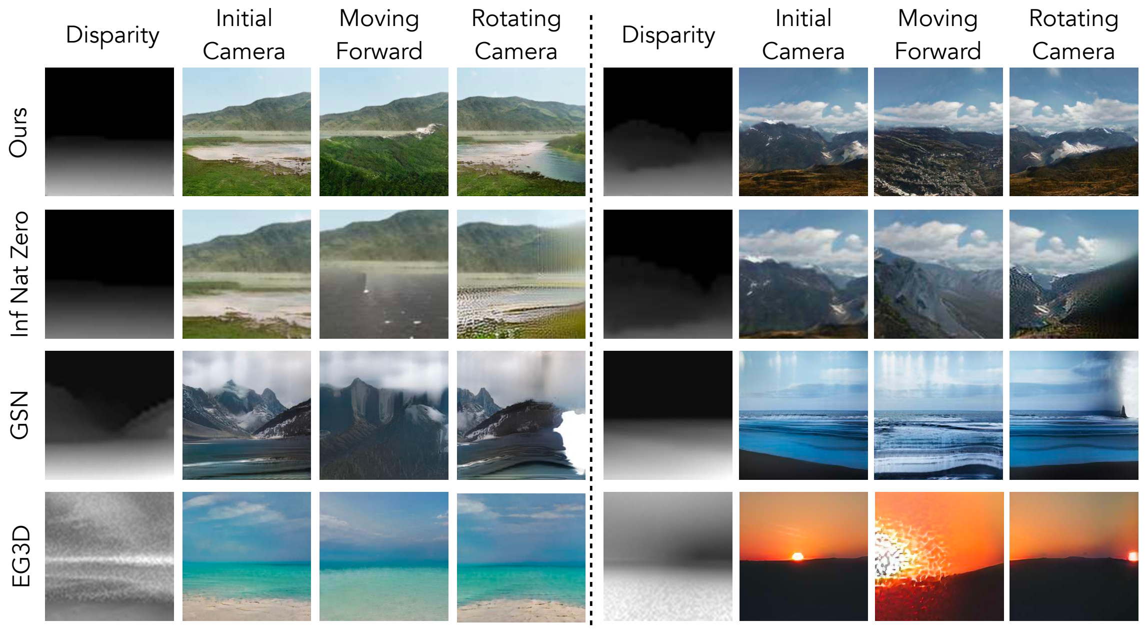


Figure 5-6: **Comparison to auto-regressive and bounded-volume 3D generative models.** Each row shows results for a given method on two generated scenes under different camera motion, along with a disparity map. Compared to InfiniteNature-Zero, our model enables long-range view synthesis by rendering a global scene description from different viewpoints, rather than auto-regressively predicting successive frames. 3D generative models like EG3D and GSN do not support view extrapolation on unbounded scenes. See our webpage for animated results.

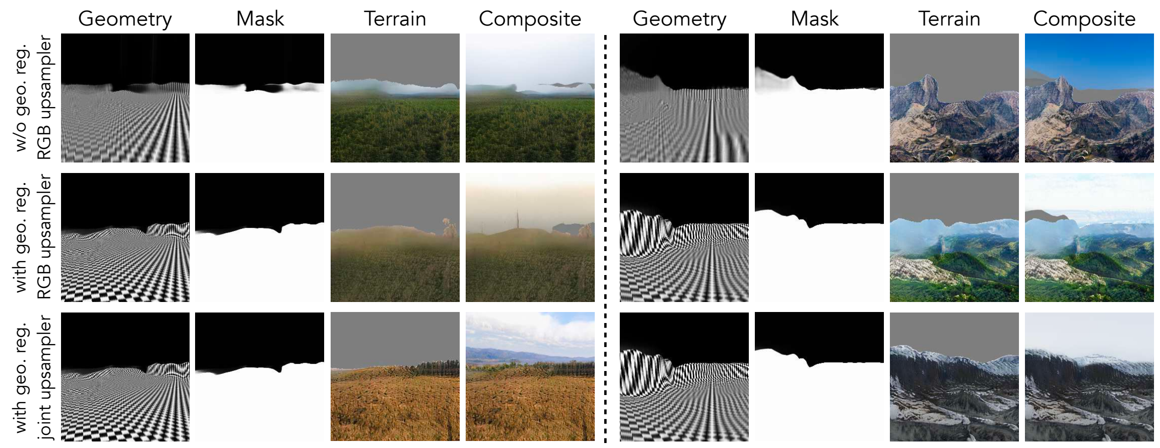

Geometry
Mask
Terrain
Composite
Geometry
Mask
Terrain
Composite
w/o geo. reg. RGB upsampler
with geo. reg. RGB upsampler
with geo. reg. joint upsampler

Figure 5-7: **Qualitative comparison of model variations.** Each row shows a model variant, visualizing generated geometry (as a rendered scene filled with a checkerboard pattern), sky mask, rendered terrain, and final image composite. (Top) Without geometry regularization, the model produces semi-transparent terrain. (Middle) Adding geometry regularization (Eqn. 5.4) makes the terrain more solid, but there are inconsistencies between the terrain and mask prediction. (Bottom) Our full model uses geometry regularization and also adds a upsampler that operates on inverse-depth and sky mask inputs in addition to RGB (Eqn. 5.6) to discourage boundary effects between the terrain and sky.

Model	**Samples Per Ray**	**Layout Resolution**	**FID (I_{LR})**
Low	64	32	33.66
Medium	128	32	32.02
High	128	64	22.62
Full	128	256	**16.06**

Table 5.3: **Variations on layout generation.** Higher feature grid resolution and more samples per ray yield the best results, bounded by computational limits. For speed, FID is computed on 5K samples rendered at 32×32.

Refinement Output	**Projected Noise**	**FID (I_{HR})**		**Consistency**
		C_{train}	C_{random}	
I_{HR}	✗	26.30	27.08	5.08
$I_{\text{HR}}, d_{\text{HR}}, m_{\text{HR}}$	✗	23.75	27.25	5.81
$I_{\text{HR}}, d_{\text{HR}}, m_{\text{HR}}$	✓	**21.42**	**23.39**	**3.91**

Table 5.4: **Variations on the refinement network.** We find refining not only the low-resolution image but also the depth and sky-mask improves image quality, but can lead to jittery results. The addition of projected noise into the upsampler results in smoother frames with lower consistency error.

samples per ray affect the quality of the volume-rendered output I_{LR}. As shown in Table 5.3, higher resolution and more samples lead to the best image quality (FID computed on 32×32 pixel images for speed, compared to segmented real images with gray sky pixels). To maximize the capacity of layout generation and rendering within computational limits we opt for a 256×256 feature grid with 128 samples per ray.

Refinement network. Next, we investigate the refinement stage, which upsamples and refines the layout generator output. In our full model, the refinement network operates not only on RGB images but also on inverse-depth and sky mask (Eqn. 5.6), and uses projected noise for spatial consistency of texture detail (Eqn. 5.7). As shown in Table 5.4, both help to improve our model's FID and consistency error.

As shown in Fig. 5-7 (second row), upsampling only the RGB image I_{LR} can lead to output that is inconsistent with the generated sky mask, leading to temporally unstable gaps in the final composited image. This figure also shows the effect of our geometric regularization (Eqn. 5.4) in reducing unwanted transparency, especially in distant terrain.

5.5 Discussion and conclusion

Limitations. A few drawbacks of our model include costly volume rendering limiting the resolution of I_{LR}, imperfect 3D consistency due to image-space refinement, and imperfect or repeating geometry decoded from the scene layout features. We elaborate in the supplemental.

Conclusion. We present an unconditional world generator for unbounded synthesis of persistent 3D nature scenes. We build persistent world representation by modeling scene content with a spatially extendable layout feature grid which can be decoded via volume rendering to form a terrain image. This rendered terrain is combined with a separate skydome, representing infinitely far content, to synthesize novel viewpoints supporting nearby and distant camera motions. Altogether, our model enables 3D consistent image generation and view synthesis of unbounded scenes learned from single-view, unposed landscape photos.

Chapter 6

Epilogue

Driven by the collection of these large datasets and improvements in training techniques, our ability to synthesize photorealistic images as improved rapidly over the past several years. However, the practical use case of creating fully random imagery is limited, thus we turn to methods that impart some manner of controlling the output during the synthesis process. With these forms of control, we can better specify the content we want the the image generator to produce.

In this dissertation, we explore two paths towards controllable content creation. Using the *emergent* priors from pre-trained synthesis models, we are able to adjust generated content according to learned perturbations to the latent codes or image exemplars which are encoded into latent space. The key idea of emergent control is that a model trained solely for the synthesis task functions as an effective image prior, and we add control capabilities by holding the model fixed while manipulating its inputs. The frozen generator allows us to maintain realistic outputs while adjusting the latent code, so that we can achieve semantically meaningful changes that are consistent with variations that we might observe in real images. On the other hand, *designed* control trains the model jointly to synthesize images while obeying additional control inputs via adjusted model architectures and training objectives. Here, we focus on designing spatial constraints, which use coordinate conditioning and geometric transformations within the model, allowing us to demonstrate variable-rate synthesis and synthesis of unbounded 3D landscapes with camera control.

Synthesis for visual analysis. The ability to create and edit synthetic images not only allows us to give users creative control over generated content, but also opens the possibility of using them for downstream visual analysis applications. Using synthesis models as infinite data generators has been explored in the context of learning image representations for classification tasks [185, 24, 99, 150, 199, 283], with recent improvements driven by the development of text-to-image synthesis models [236, 6, 198]. Additionally, these synthetic outputs can serve as stimuli for human perceptual experiments [68, 62]. With the rise of synthetic content creation, it is also important to develop mechanisms for detecting such material, using learned detection models [251, 274, 23, 282, 294] for entirely synthesized content, or consistency methods when parts of the input have been manipulated [91, 147].

Recent advances in text-to-image synthesis. The image synthesis pipeline has undergone dramatic improvements within the past year (2022-2023) with the rise of massive text-to-image datasets [202] and the development of diffusion and transformer-based synthesis methods [29, 55, 48, 224, 226, 84] leading to massive foundation models that generate high quality images by combining stochastic noise sampling with text inputs [192, 30]. In contrast to adversarial objectives in which two competing models must be carefully balanced, these alternatives offer more stable training trajectories via an iterative synthesis procedure with ground truth targets, in which the task is to gradually recover the training image starting from a randomly sampled latent in multiple iterative steps. While GAN models have largely focused on single domains or curated datasets like ImageNet, recent work has also developed techniques for scaling up GAN frameworks to handle text-to-image synthesis on arbitrary domains [201, 104].

However, even within the scope of these newly developed models, similar editing goals remain – with just a text caption it is impossible to provide a precise description of everything one might want to see in an image, so we often require additional techniques to gain increased control over resulting output. Using these recent text-to-image models as general-purpose image priors and adjusting the inputs and inference procedure allows for properties of emergent control over image editing [122, 241, 81, 54, 152, 63]. One particular application also enables 3D camera control from pretrained 2D diffusion models by optimizing a neural field or mesh using a text-to-image model as a prior [180, 132, 61]. Alternatively, 3D controls can be incorporated into directly into diffusion models with aspects of designed control, by reformulating the 3D domain into 2.5D, 2D, or latent feature representations during model training [129, 216, 28, 264, 257]. As large vision models become increasingly difficult to train from scratch, an increasingly common approach is a hybrid setup which leverages the emergent priors of a pretrained model and further finetunes the model with additional designed objectives for downstream editing [193, 121, 21, 278, 120, 64, 141].

Future directions of image synthesis. Image synthesis as a field is rapidly progressing, scaling up with larger models and increased diversity of the context we can create. However, synthesis today is not fully capable of representing everything that we might observe in the world. Models today tend to be better at producing object-centric results, but large-scale scenes and complex, detailed layouts remain difficult to capture in a fully realistic manner. In addition, there are several ways that we might imagine interacting with our synthesized content.

One possibility is framing synthesis as an iterative procedure, in which the model can incrementally adjust its outputs based on intermediate user feedback. These feedback systems have been deployed in text generation domains [56, 168], but image synthesis today tends to be a single-shot process, in which the user provides some input and the model produces an output, rather than an interactive, cyclical process. Doing so brings about additional challenges, such as balancing between preserving content from previous editing operations while making the adequate changes at subsequent iterations.

The world itself is also does not exist in a static state; things can move and change over time. While representing a moving camera within a generated scene is one way of creating dynamic content, it does not address how the scene itself can change. Currently, representing dynamic scenes has been handled via video generation techniques [222, 20, 219, 16], but these do not address the topic of interactive generation. For example, we might imagine that if we interactively reconfigure objects in the scene, the output might change as the result of the sequence of individual actions taken. Representing dynamic, interactive content creation will require us to reevaluate how we represent scenes, objects, and motion within our synthesis pipelines.

As our techniques for creating synthetic imagery improve, the fundamental problem of how we can control aspects of the synthesized content continues to reappear. Improved control can allow us to efficiently convert anything we imagine in our minds directly into images, without needing to manually draw the individual pixel values. Through methods of emergent control added to pretrained models or designed control built into the model training process, we can develop techniques that allow image generators to serve as even better creative assistants and increasingly realistic world models, allowing synthesis to be a more useful tool in our everyday lives.

Appendix A

Supplementary: GAN Steering

A.1 Additional Methodological Details

A.1.1 Optimization for the linear walk

We learn the walk vector using mini-batch stochastic gradient descent with the Adam optimizer [114] in tensorflow, trained on 20000 unique samples from the latent space z. We share the vector w across all ImageNet categories for the BigGAN model.

A.1.2 Implementation details for linear walk

We experiment with a number of different transformations learned in the latent space, each corresponding to a different walk vector. Each of these transformations can be learned without any direct supervision, simply by applying our desired edit to the source image. Furthermore, the parameter α allows us to vary the extent of the transformation. We found that a slight modification to each transformation improved the degree to which we were able to steer the output space: we scale α differently for the learned transformation $G(z + \alpha_g w)$, and the target edit $\texttt{edit}(G(z), \alpha_t)$. We detail each transformation below:

Shift. We learn transformations corresponding to shifting an image in the horizontal X direction and the vertical Y direction. We train on source images that are shifted

$-\alpha_t$ pixels to the left and α_t pixels to the right, where we set α_t to be between zero and one-half of the source image width or height D. When training the walk, we enforce that the α_g parameter ranges between -1 and 1; thus for a random shift by t pixels, we use the value $\alpha_g = \alpha_t/D$. We apply a mask to the shifted image, so that we only apply the loss function on the visible portion of the source image. This forces the generator to extrapolate on the obscured region of the target image.

Zoom. We learn a walk which is optimized to zoom in and out up to four times the original image. For zooming in, we crop the central portion of the source image by some α_t amount, where $0.25 < \alpha_t < 1$ and resize it back to its original size. To zoom out, we downsample the image by α_t where $1 < \alpha_t < 4$. To allow for both a positive and negative walk direction, we set $\alpha_g = \log(\alpha_t)$. Similar to shift, a mask applied during training allows the generator to inpaint the background scene.

Color. We implement color as a continuous RGB slider, e.g., a 3-tuple $\alpha_t = (\alpha_R$, α_G, $\alpha_B)$, where each α_R, α_G, α_B can take values between $[-0.5, 0.5]$ in training. To edit the source image, we simply add the corresponding α_t values to each of the image channels. Our latent space walk is parameterized as $z + \alpha_g w = z + \alpha_R w_R + \alpha_G w_G + \alpha_B w_B$ where we jointly learn the three walk directions w_R, w_G, and w_B.

Rotate in 2D. Rotation in 2D is trained in a similar manner as the shift operations, where we train with $-45 \leq \alpha_t \leq 45$ degree rotation. Using $R = 45$, scale $\alpha_g = \alpha_t/R$. We use a mask to enforce the loss only on visible regions of the target.

Rotate in 3D. We simulate a 3D rotation using a perspective transformation along the Z-axis, essentially treating the image as a rotating billboard. Similar to the 2D rotation, we train with $-45 \leq \alpha_t \leq 45$ degree rotation, we scale $\alpha_g = \alpha_t/R$ where $R = 45$, and apply a mask during training.

A.1.3 Linear NN(z) walk

Rather than defining w as a vector in z space (Eq. 2.1), one could define it as a function that takes a z as input and maps it to the desired z' after taking a variable-sized step α in latent space. In this case, we may parametrize the walk with a neural network

$w = \mathrm{NN}(z)$, and transform the image using $G(z + \alpha \mathrm{NN}(z))$. However, as we show in the following proof, this idea will not learn to let w be a function of z.

Proof. For simplicity, let $w = F(z)$. We optimize for $J(w, \alpha) = \mathbb{E}_z \left[\mathcal{L}(G(z + \alpha w), \texttt{edit}(G(z), \alpha))\right]$ where α is an arbitrary scalar value. Note that for the target image, two equal edit operations is equivalent to performing a single edit of twice the size (e.g., shifting by 10px the same as shifting by 5px twice; zooming by 4x is the same as zooming by 2x twice). That is,

$$\texttt{edit}(G(z), 2\alpha) = \texttt{edit}(\texttt{edit}(G(z), \alpha), \alpha).$$

To achieve this target, starting from an initial z, we can take two steps of size α in latent space as follows:

$$\begin{aligned} z_1 &= z + \alpha F(z) \\ z_2 &= z_1 + \alpha F(z_1) \end{aligned}$$

However, because we let α take on any scalar value during optimization, our objective function enforces that starting from z and taking a step of size 2α equals taking two steps of size α:

$$z + 2\alpha F(z) = z_1 + \alpha F(z_1) \tag{A.1}$$

Therefore:

$$\begin{aligned} z + 2\alpha F(z) = z + \alpha F(z) + \alpha F(z_1) &\Rightarrow \\ \alpha F(z) = \alpha F(z_1) &\Rightarrow \\ F(z) = F(z_1). & \end{aligned}$$

Thus $F(\cdot)$ simply becomes a linear trajectory that is independent of the input z. □

A.1.4 Optimization for the non-linear walk

Given the limitations of the previous walk, we define our nonlinear walk $F(z)$ using discrete step sizes ϵ. We define $F(z)$ as $z + \mathrm{NN}(z)$, where the neural network NN learns a fixed ϵ step transformation, rather than a variable α step. We then renormalize the magnitude z. This approach mimics the Euler method for solving ODEs with a discrete step size, where we assume that the gradient of the transformation in latent space is of the form $\epsilon\frac{dz}{dt} = \mathrm{NN}(z)$ and we approximate $z_{i+1} = z_i + \epsilon\frac{dz}{dt}|_{z_i}$. The key difference from A.1.3 is the fixed step size, which avoids optimizing for the equality in (A.1).

We use a two-layer neural network to parametrize the walk, and optimize over 20000 samples using the Adam optimizer as before. Positive and negative transformation directions are handled with two neural networks having identical architecture but independent weights. We set ϵ to achieve the same transformation ranges as the linear trajectory within 4-5 steps.

A.2 Additional Experiments

A.2.1 Model and data distributions

How well does the model distribution of each property match the dataset distribution? If the generated images do not form a good approximation of the dataset variability, we expect that this would also impact our ability to transform generated images. In Fig. A-1 we show the attribute distributions of the BigGAN model $G(z)$ compared to samples from the ImageNet dataset. We show corresponding results for StyleGAN and its respective datasets in Appendix A.2.5. While there is some bias in how well model-generated images approximate the dataset distribution, we hypothesize that additional biases in our transformations come from variability in the training data.

A.2.2 Quantifying transformation limits

We observe that when we increase the transformation magnitude α in latent space, the generated images become unrealistic and the transformation ceases to have further effect. We show this qualitatively in Fig. 2-3. To quantitatively verify this trends, we can compute the LPIPS perceptual distance of images generated using consecutive pairs of α_i and α_{i+1}. For shift and zoom transformations, perceptual distance is larger when α (or $\log(\alpha)$ for zoom) is near zero, and decreases as the the magnitude of α increases, which indicates that large α magnitudes have a smaller transformation effect, and the transformed images appear more similar. On the other hand, color and rotate in 2D/3D exhibit a steady transformation rate as the magnitude of α increases.

Note that this analysis does not tell us how well we achieve the specific transformation, nor whether the latent trajectory deviates from natural-looking images. Rather, it tells us how much we manage to change the image, regardless of the transformation target. To quantify how well each transformation is achieved, we rely on attribute detectors such as object bounding boxes (see A.2.3).

A.2.3 Detected bounding boxes

To quantify the degree to which we are able to achieve the zoom and shift transformations, we rely on a pre-trained *MobileNet-SSD v1*[*] object detection model. In Fig. A-3 and A-4 we show the results of applying the object detection model to images from the dataset, and images generated by the model under the zoom, horizontal shift, and vertical shift transformations for randomly selected values of α, to qualitatively verify that the object detection boundaries are reasonable. Not all ImageNet images contain recognizable objects, so we only use ImageNet classes containing objects recognizable by the detector for this analysis.

A.2.4 Alternative walks in BigGAN

LPIPS objective

In the main text, we learn the latent space walk w by minimizing the objective function:

$$J(w, \alpha) = \mathbb{E}_z \left[\mathcal{L}(G(z + \alpha w), \texttt{edit}(G(z), \alpha)) \right]. \tag{A.2}$$

[*]https://github.com/opencv/opencv/wiki/TensorFlow-Object-Detection-API

using a Euclidean loss for $\mathcal{L}$. In Fig. A-5 we show qualitative results using the LPIPS perceptual similarity metric [280] instead of Euclidean loss. Walks were trained using the same parameters as those in the linear-L2 walk shown in the main text: we use 20k samples for training, with Adam optimizer and learning rate 0.001 for zoom and color, 0.0001 for the remaining edit operations (due to scaling of α).

Non-linear Walks

Following A.2.4, we modify our objective to use discrete step sizes ϵ rather than continuous steps. We learn a function $F(z)$ to perform this ϵ-step transformation on given latent code z, where $F(z)$ is parametrized with a neural network. We show qualitative results in Fig. A-6. We perform the same set of experiments shown in the main text using this nonlinear walk in Fig. A-7. These experiments exhibit similar trends as we observed in the main text – we are able to modify the generated distribution of images using latent space walks, and the amount to which we can transform is related to the variability in the dataset. However, there are greater increases in FID when we apply the non-linear transformation, suggesting that these generated images deviate more from natural images and look less realistic.

Additional Qualitative Examples

We show qualitative examples for randomly generated categories for BigGAN linear-L2, linear LPIPS, and nonlinear trajectories in Figs. A-8, A-9, A-10 respectively.

A.2.5 Walks in StyleGAN

We perform similar experiments for linear latent space walks using StyleGAN models trained on the LSUN cat, LSUN car, and FFHQ face datasets. As suggested by Karras et al. [107], we learn the walk vector in the intermediate W latent space due to improved attribute disentanglement in W. We show qualitative results for color, shift, and zoom transformations in Figs. A-11, A-13, A-15 and corresponding quantitative analyses in Figs. A-12, A-14, A-16. We show qualitative examples for the comparison of optimizing in the W and z latent spaces in Stylegan in A-19.

A.2.6 Walks in Progressive GAN

We also experiment with the linear walk objective in the latent space of Progressive GAN [106]. One interesting property of the Progressive GAN interpolations is that they take much longer to train to have a visual effect – for example for color, we could obtain drastic color changes in Stylegan W latent space using as few as 2k samples, but with progressive gan, we used 60k samples and still did not obtain as strong of an effect. This points to the Stylegan w latent space being more "flexible" and generalizable for transformation, compared to the latent space of progressive GAN. Moreover, we qualitatively observe some entanglement in the progressive gan transformations – for example, changing the level of zoom also changes the lighting. We did not observe big effects in the horizontal and vertical shift transformations. Qualitative examples and quantitative results are shown in Figs. A-17, A-18.

A.2.7 Qualitative examples for additional transformations

Since the color transformation operates on individual pixels, we can optimize the walk using a segmented target – for example when learning a walk for cars, we only modify pixels in segmented car region when generating $\texttt{edit}(G(z), \alpha)$. StyleGAN is able to roughly localize the color transformation to this region, suggesting disentanglement of different objects within the W latent space (Fig. A-20 left) as also noted in Karras et al. [107], Shen et al. [210]. We also show qualitative results for adjust image contrast (Fig. A-20 right), and for combining zoom, shift X, and shift Y transformations (Fig. A-21).

A.2.8 Additional results for improving model steerability

We further test the hypothesis that dataset variability impacts the amount we are able to transform by comparing DCGAN models trained with and without data augmentation. Namely, with data augmentation, the discriminator is able to see edited versions of the real images. We also jointly train the model and the walk trajectory which encourages the model to learn linear walks. For zoom, horizontal shift, and 2D rotate transformations, additional samples for three training approaches – without data augmentation, with data augmentation, and joint optimization – appear in Fig. A-22-A-24. Qualitatively, transformations using the model trained without data augmentation degrade the digit structure as α magnitude increases, and may even change one digit to another. Training with data augmentation and joint optimization better preserves digit structure and identity.

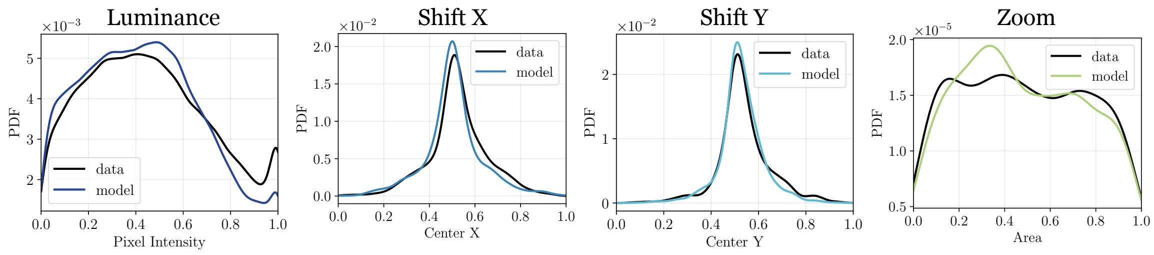


Figure A-1: **Comparing model versus dataset distribution.** We plot statistics of the generated under the color (luminance), zoom (object bounding box size), and shift operations (bounding box center), and compare them to the statistics of images in the training dataset.

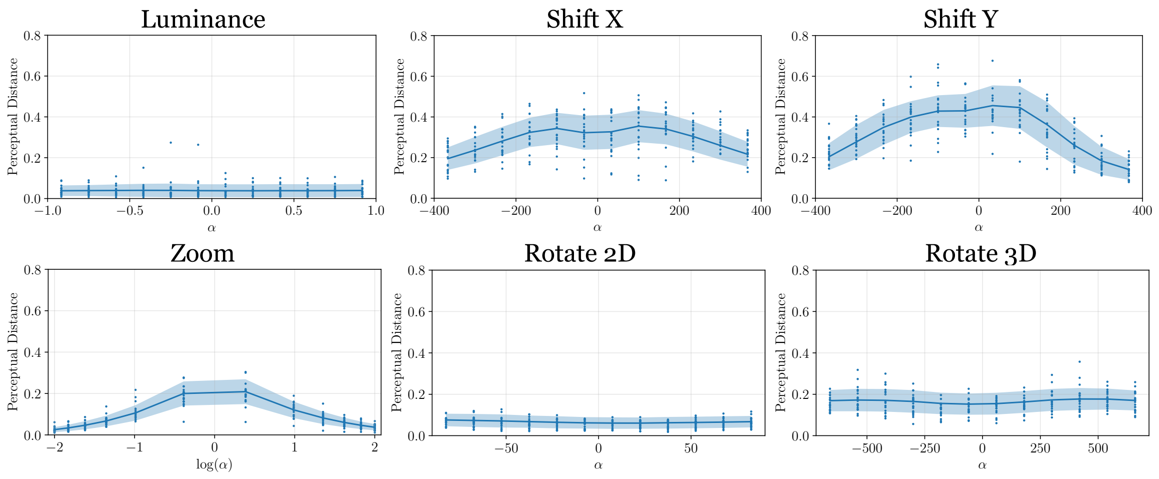


Figure A-2: **Perceptual distances along the transformation path.** LPIPS Perceptual distances between images generated from pairs of consecutive α_i and α_{i+1}. We sample 1000 images from randomly selected categories using BigGAN, transform them according to the learned linear trajectory for each transformation. We plot the mean perceptual distance and one standard deviation across the 1000 samples (shaded area), as well as 20 individual samples (scatterplot). Because the Rotate 3D operation undershoots the targeted transformation, we observe more visible effects when we increase the α magnitude.

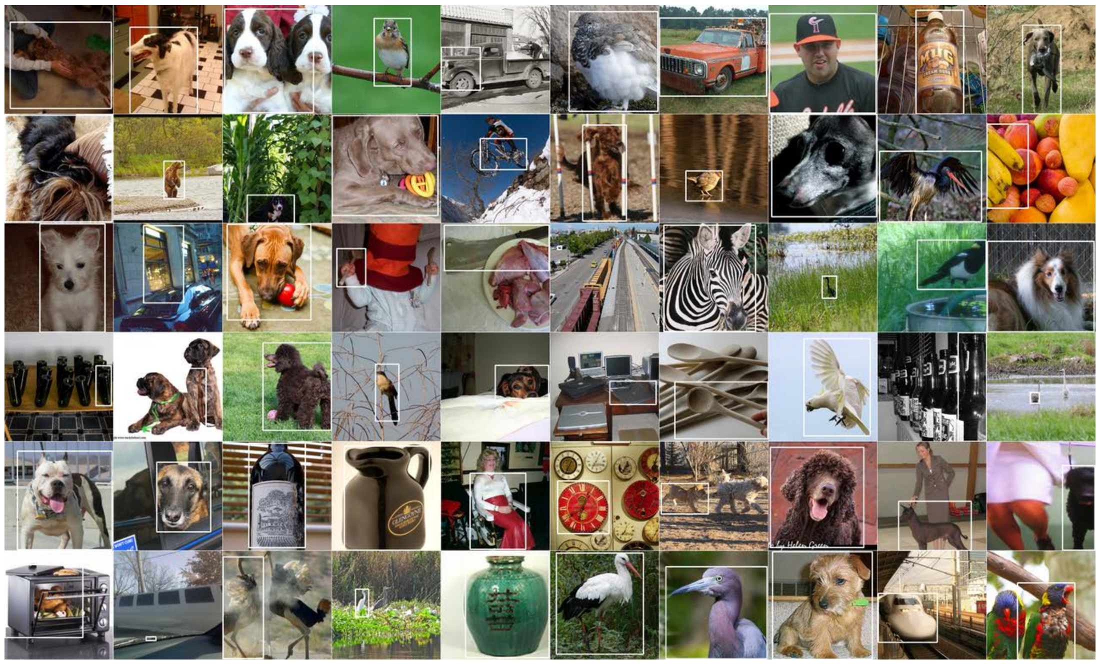


Figure A-3: **Visualization of object boxes.** Bounding boxes for random selected classes using ImageNet training images.

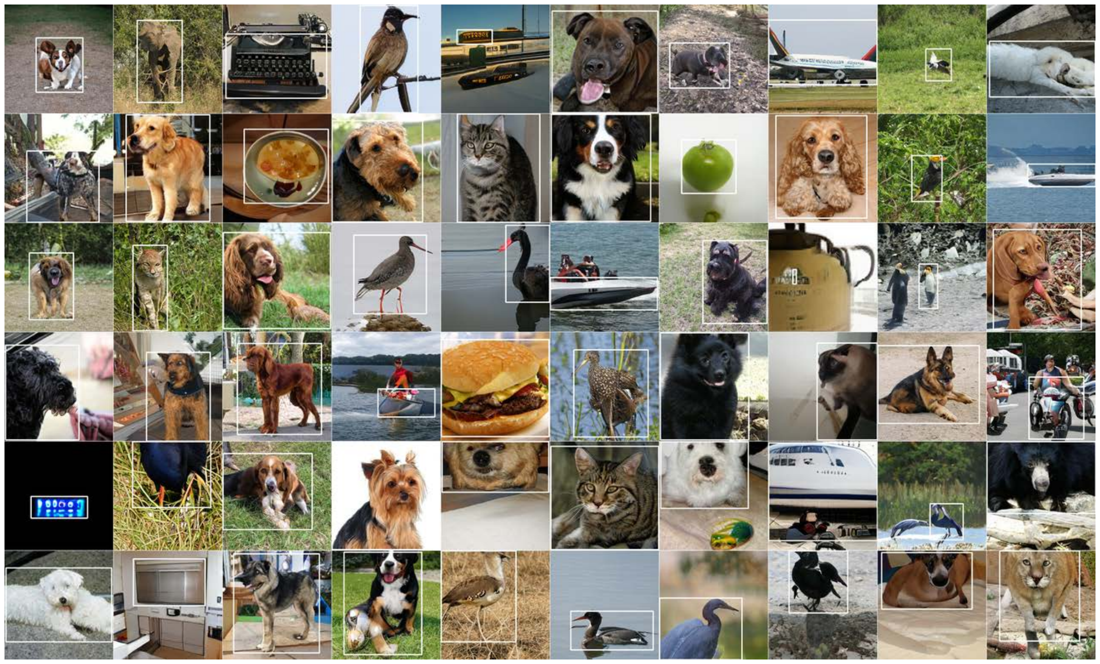


Figure A-4: **Object boxes for generated images.** Bounding boxes for random selected classes using model-generated images for zoom and horizontal and vertical shift transformations under random values of α.

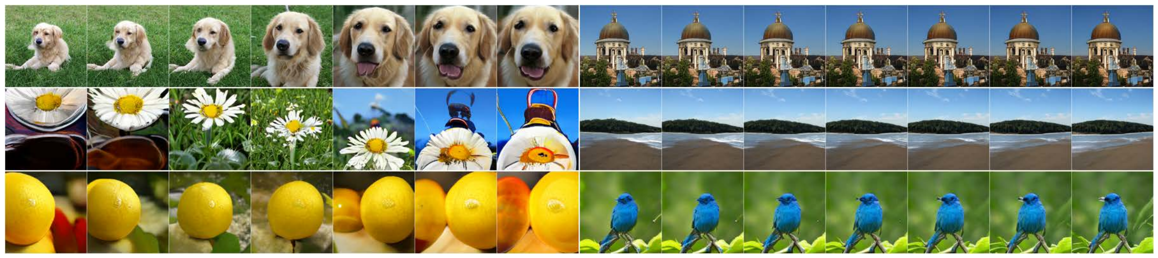


Figure A-5: **Optimization using perceptual losses.** Linear walks in BigGAN, trained to minimize LPIPS loss. For comparison, we show the same samples as in Fig. 2-1 (which used a linear walk with L2 loss).

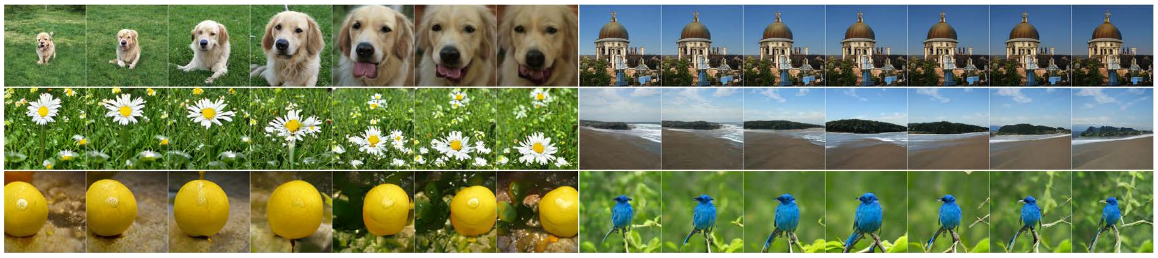


Figure A-6: **Nonlinear walks in BigGAN.** These are trained to minimize L2 loss for color and LPIPS loss for the remaining transformations. For comparison, we show the same samples in Fig. 2-1 (which used a linear walk with L2 loss), replacing the linear walk vector w with a nonlinear walk.

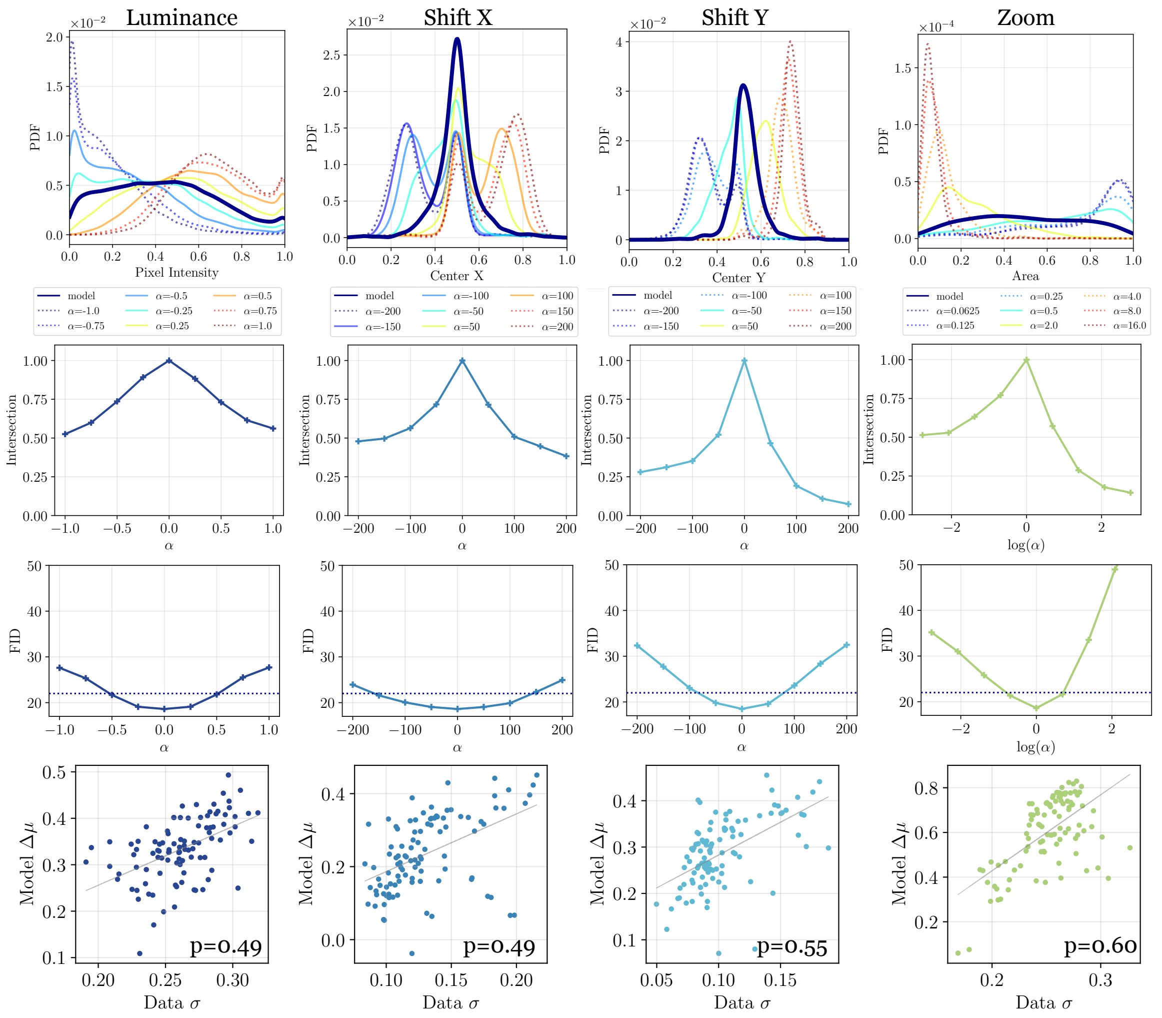


Figure A-7: **Quantitative experiments for nonlinear walks in BigGAN.** We show the attributes of generated images under the raw model output $G(z)$, compared to the distribution under a learned transformation $\texttt{model}(\alpha)$, the intersection area between $G(z)$ and $\texttt{model}(\alpha)$, FID score on transformed images, and scatterplots relating dataset variability to the extent of model transformation.

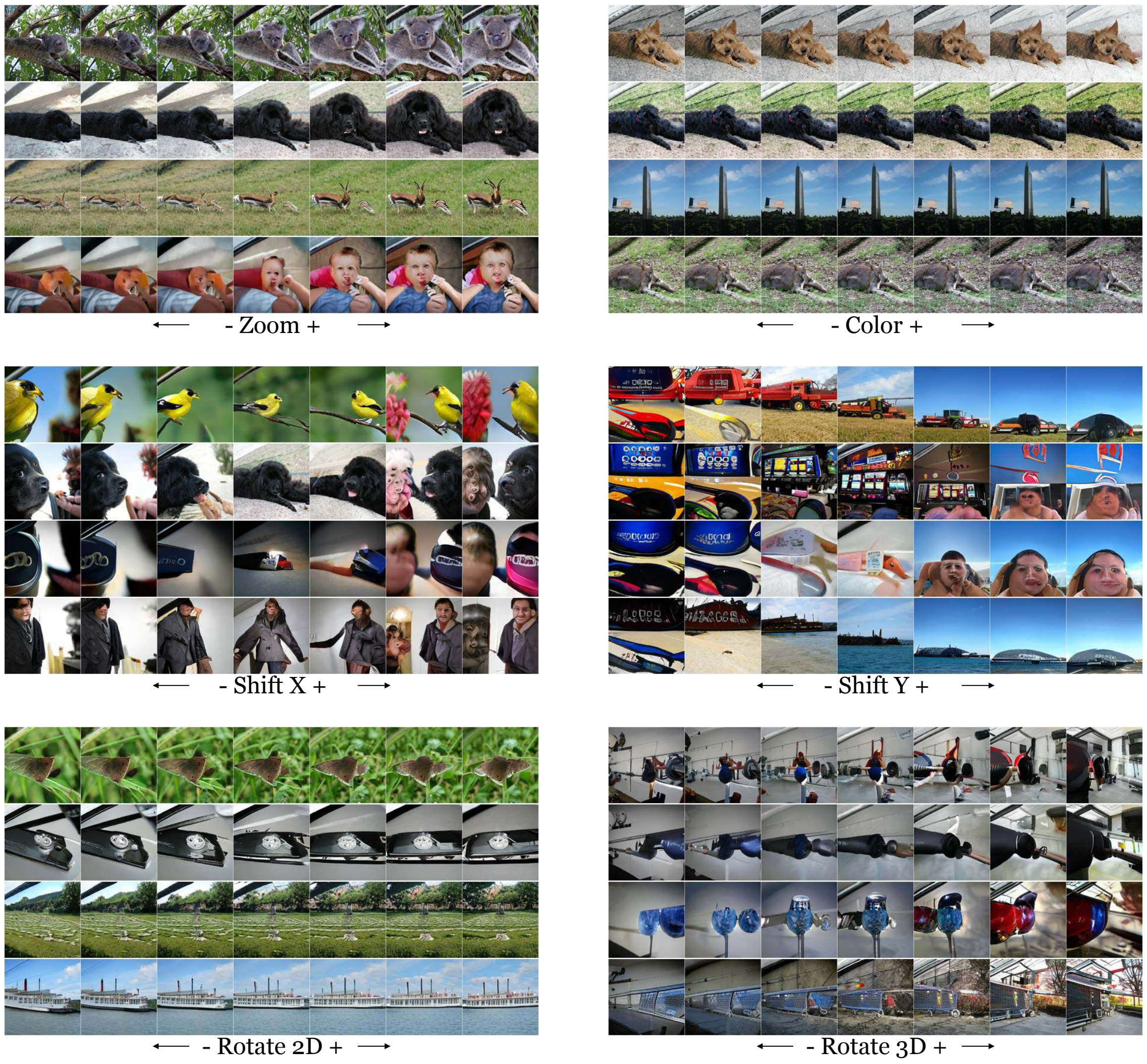


Figure A-8: **Samples with L2 loss.** Qualitative examples for randomly selected categories in BigGAN, using the linear trajectory and L2 objective.

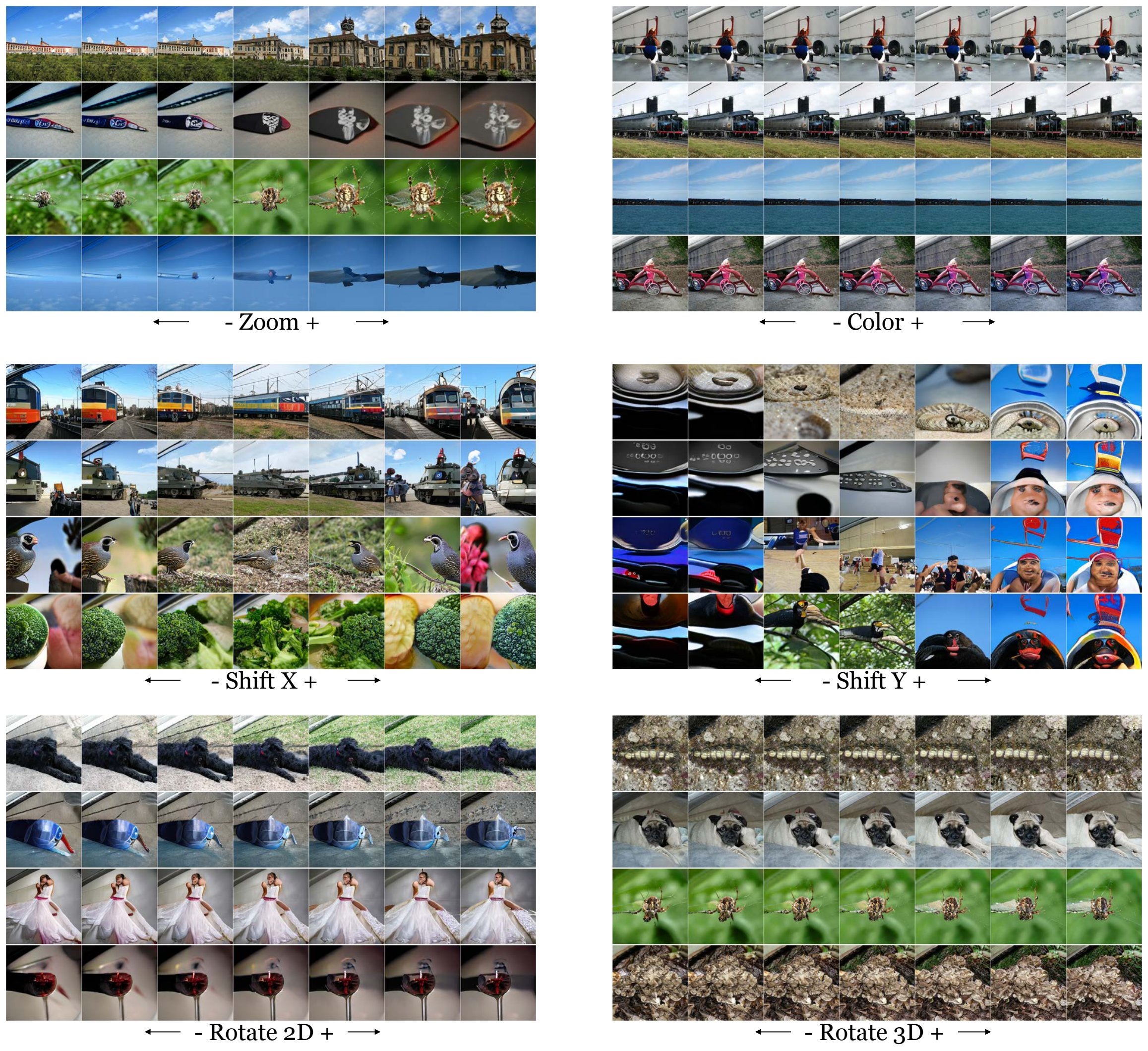


Figure A-9: **Samples with perceptual loss.** Qualitative examples for randomly selected categories in BigGAN, using the linear trajectory and LPIPS objective.

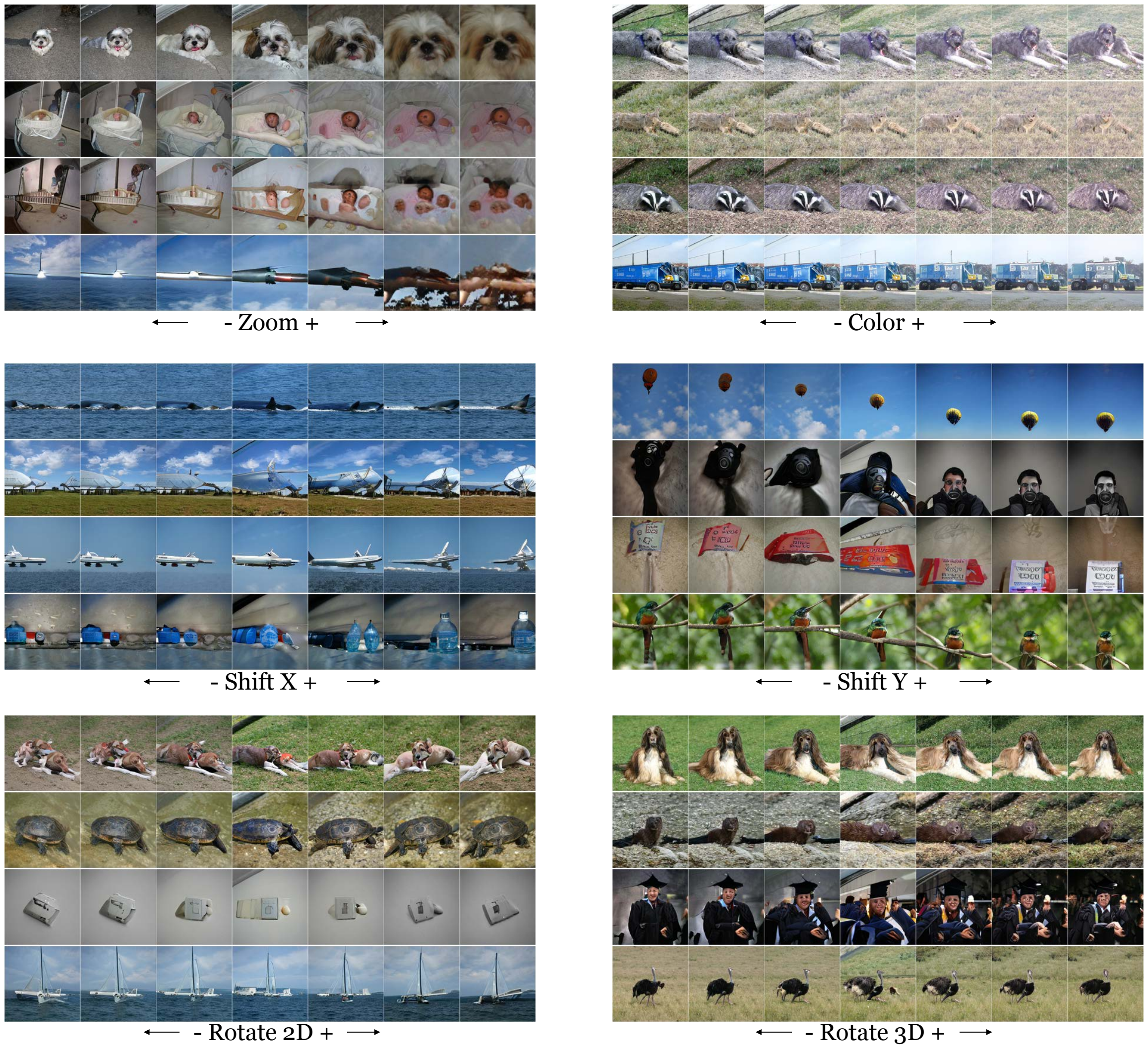


Figure A-10: **Samples with nonlinear trajectory.** Qualitative examples for randomly selected categories in BigGAN, using a nonlinear trajectory.

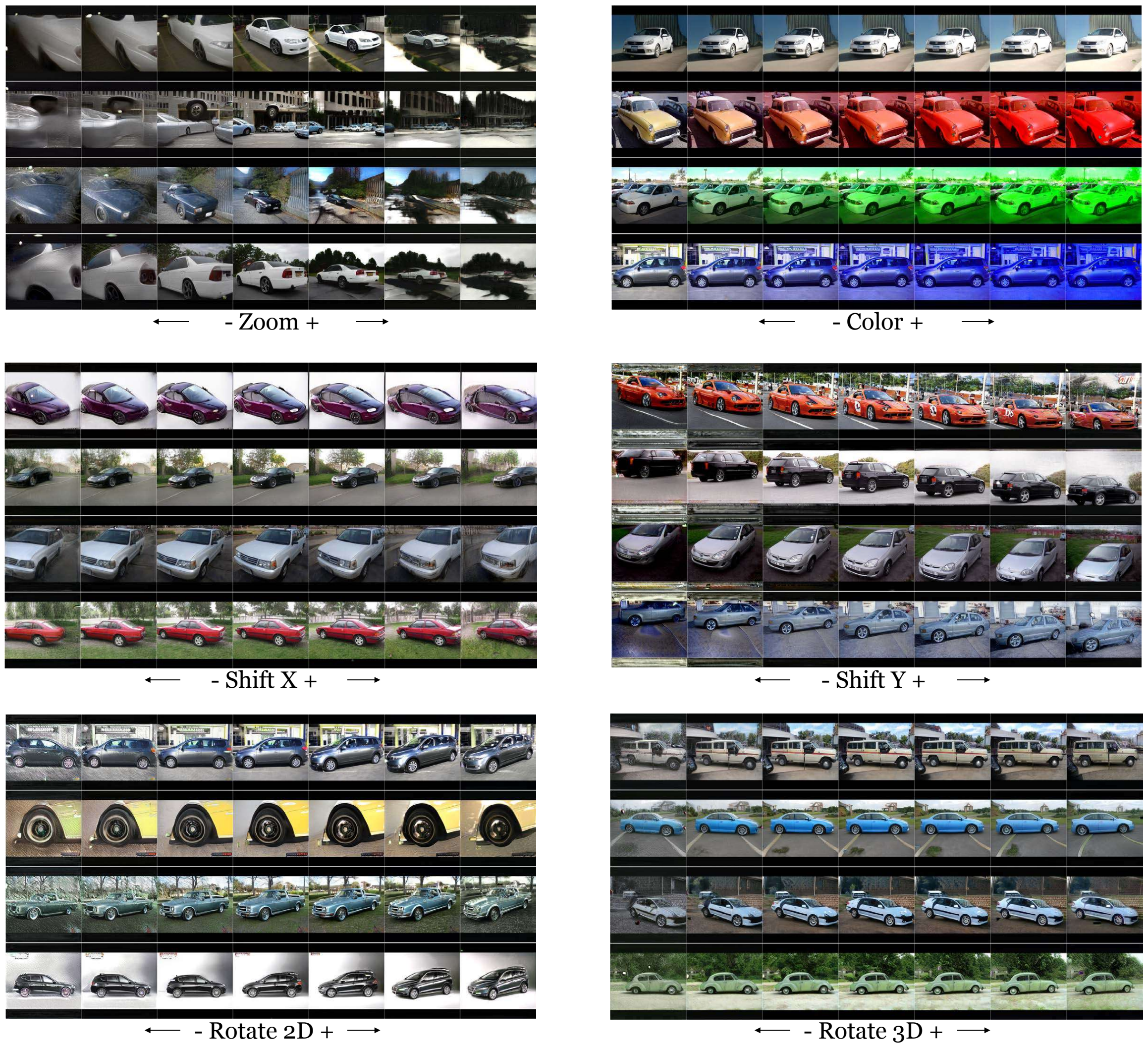


Figure A-11: **Samples using StyleGAN car generator.** Qualitative examples for learned transformations using the StyleGAN car generator.

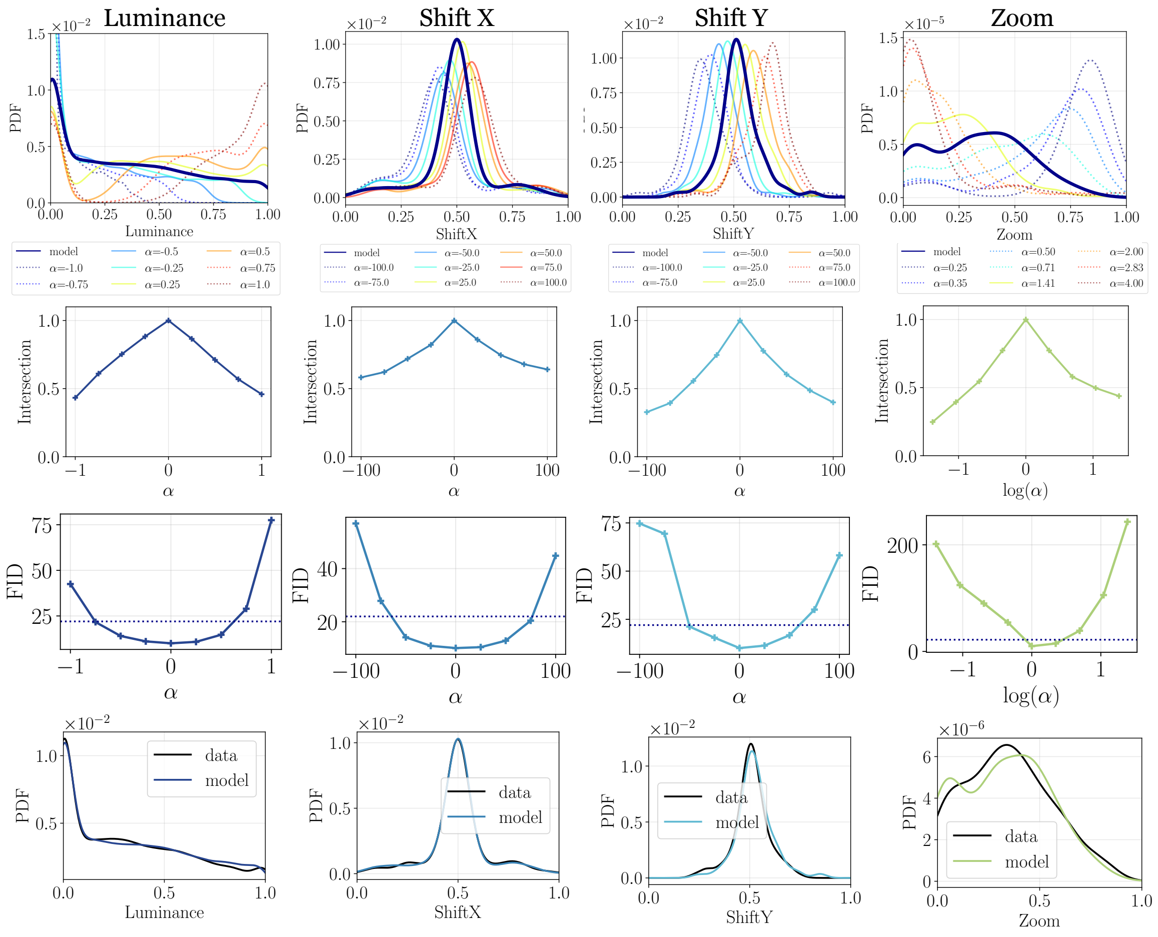


Figure A-12: **StyleGAN car model transformation metrics.** Quantitative experiments for learned transformations using the StyleGAN car generator.

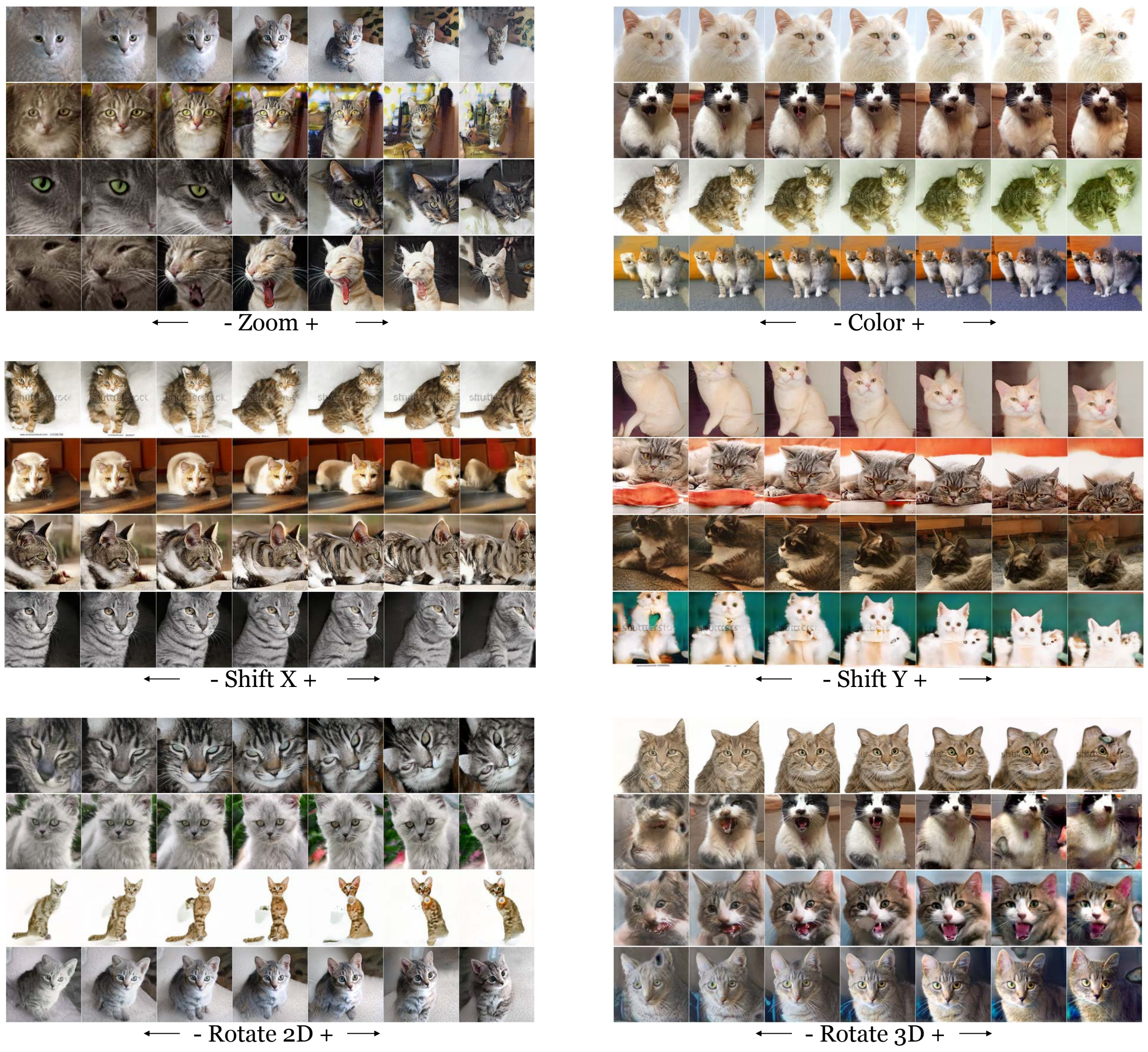


Figure A-13: **Samples using StyleGAN cat generator.** Qualitative examples for learned transformations using the StyleGAN cat generator.

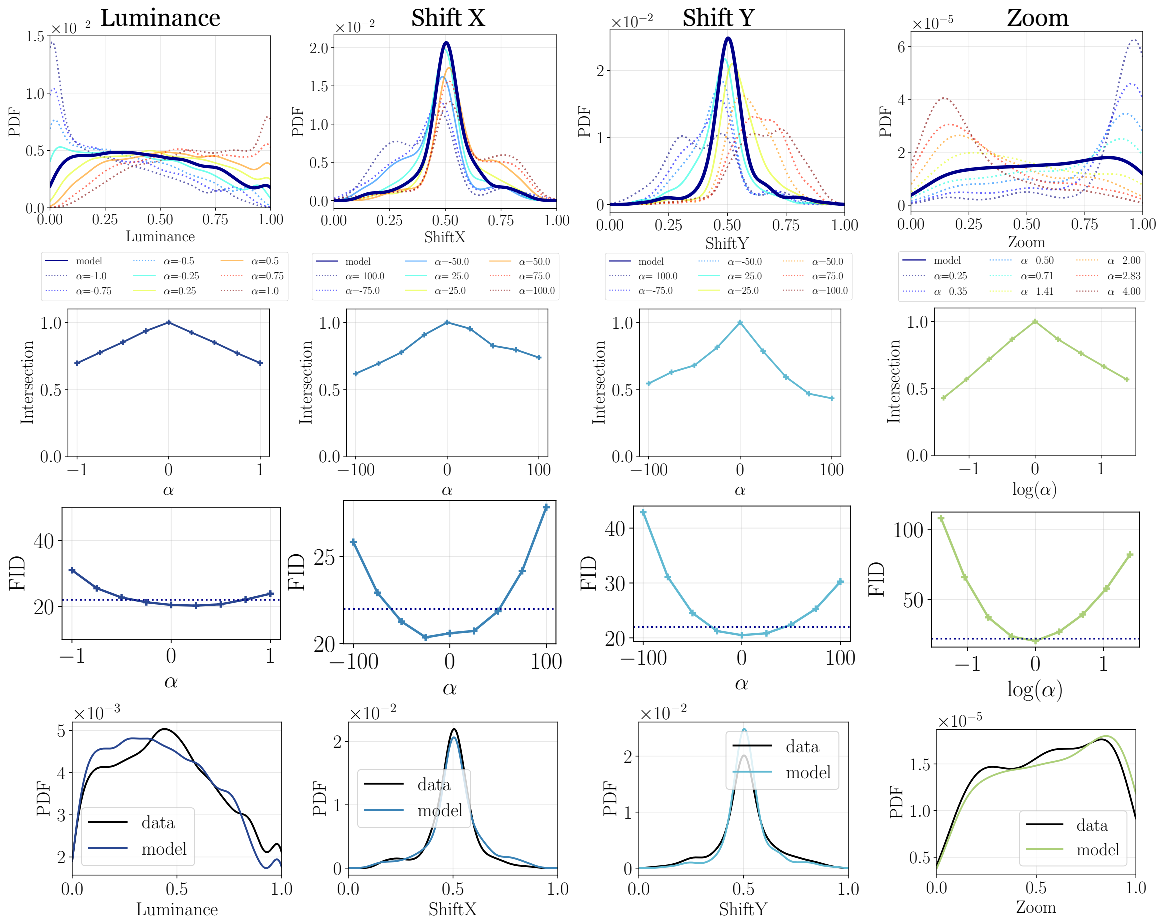


Figure A-14: **StyleGAN cat model transformation metrics.** Quantitative experiments for learned transformations using the StyleGAN cat generator.

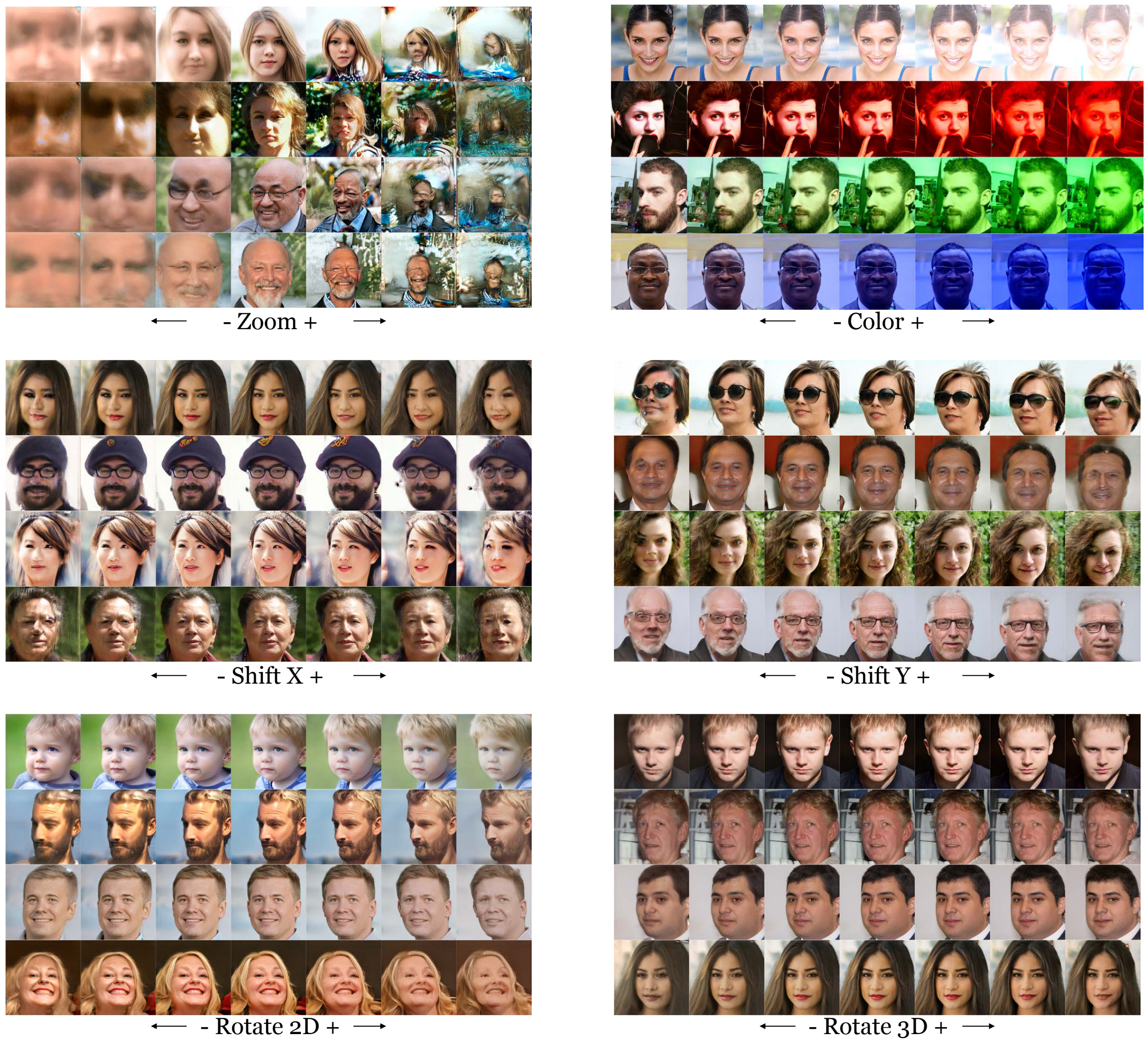


Figure A-15: **Samples using StyleGAN face generator.** Qualitative examples for learned transformations using the StyleGAN FFHQ face generator.

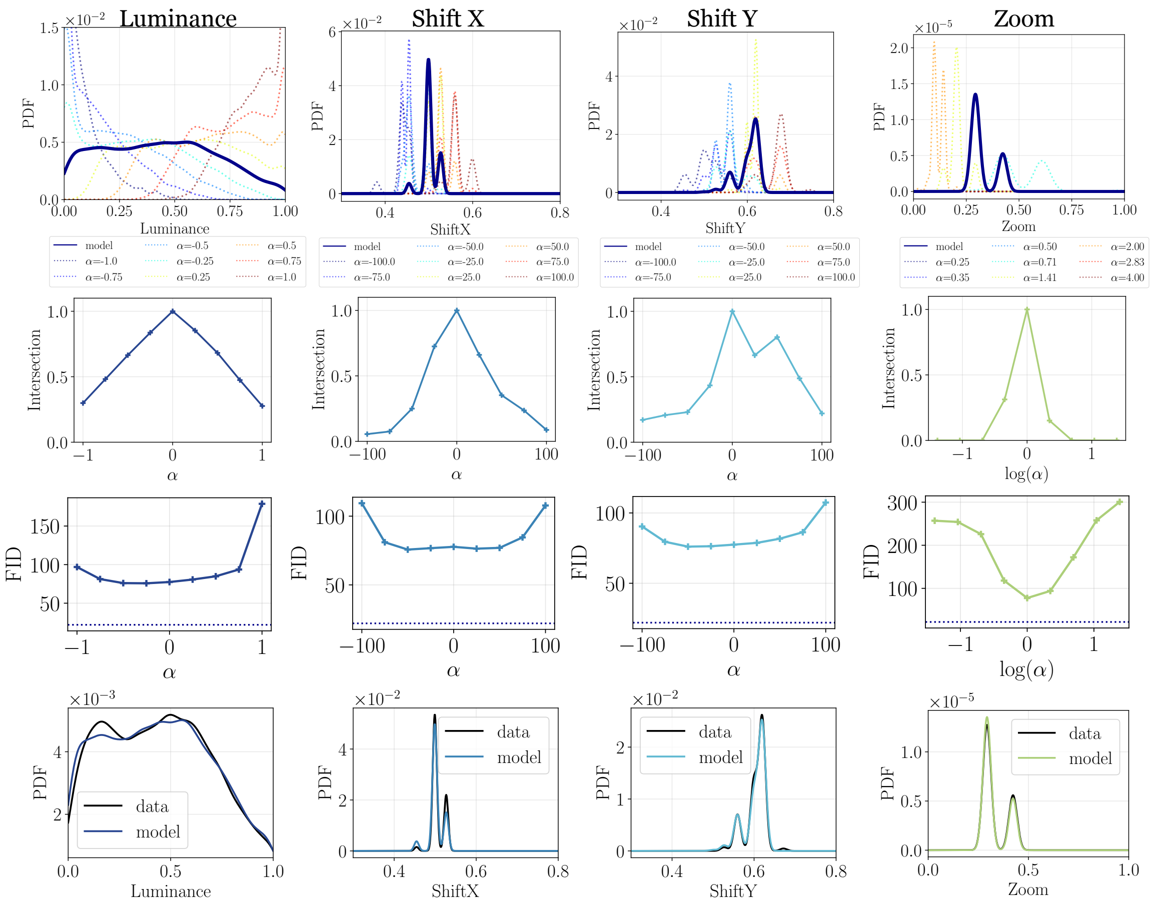


Figure A-16: **StyleGAN face model transformation metrics.** Quantitative experiments for learned transformations using the StyleGAN FFHQ face generator. For the zoom operation not all faces are detectable; we plot the distribution as zeros for α values in which no face is detected. We use the dlib face detector [113] for bounding box coordinates.

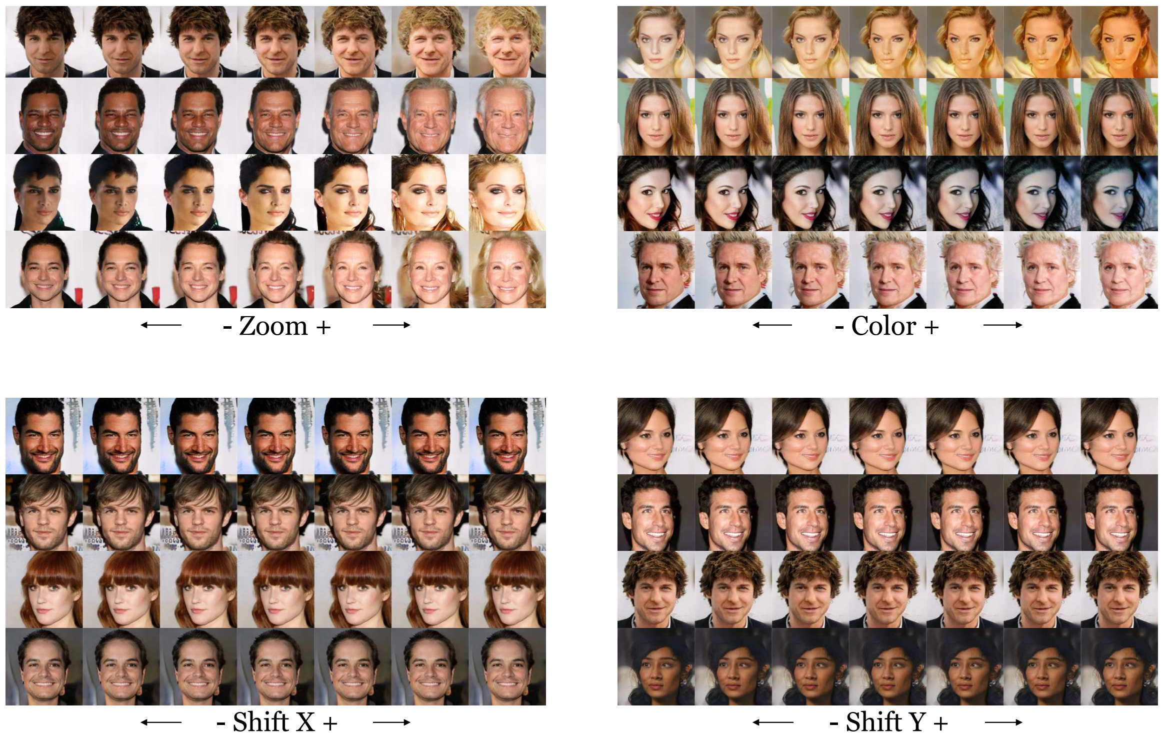


Figure A-17: **Samples using ProGAN face generator.** Qualitative examples for learned transformations using the Progressive GAN CelebaA-HQ face generator.

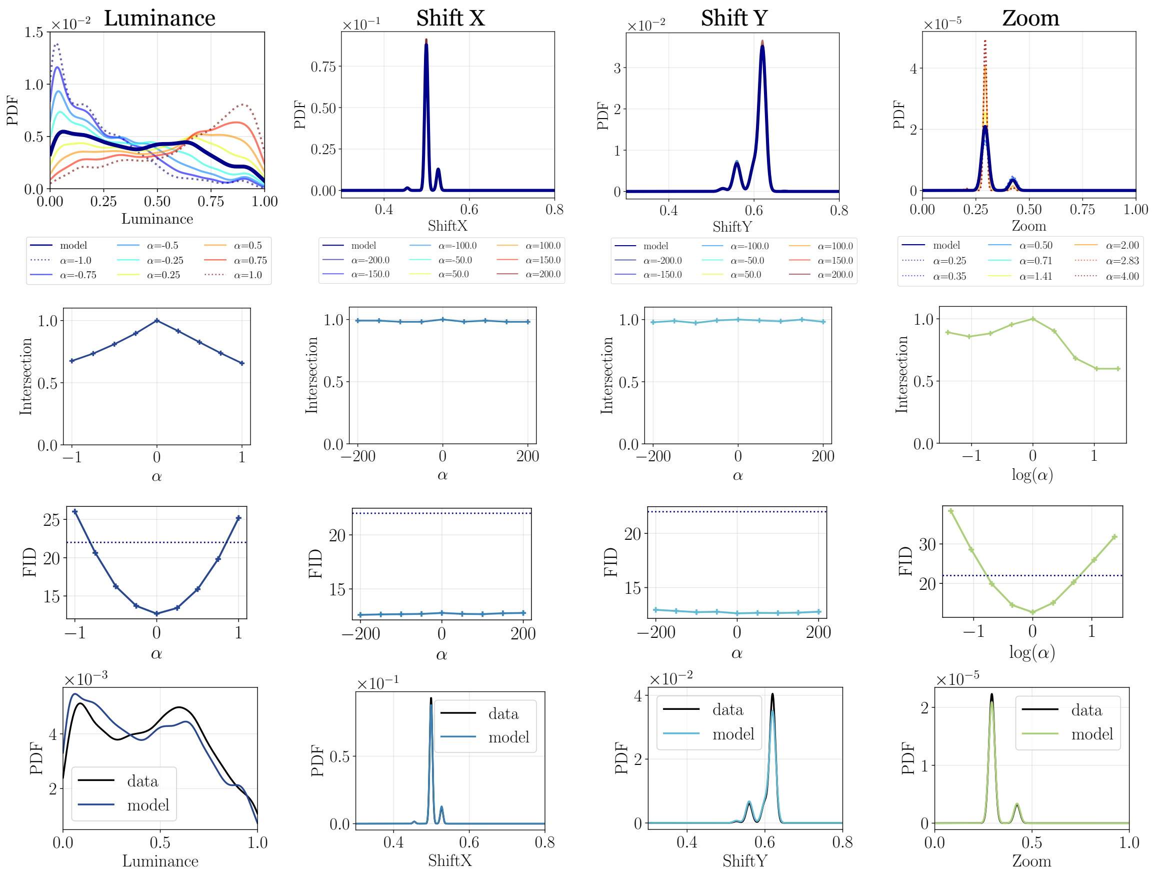


Figure A-18: **ProGAN face model transformation metrics.** Quantitative experiments for learned transformations using the Progressive GAN CelebA-HQ face generator.

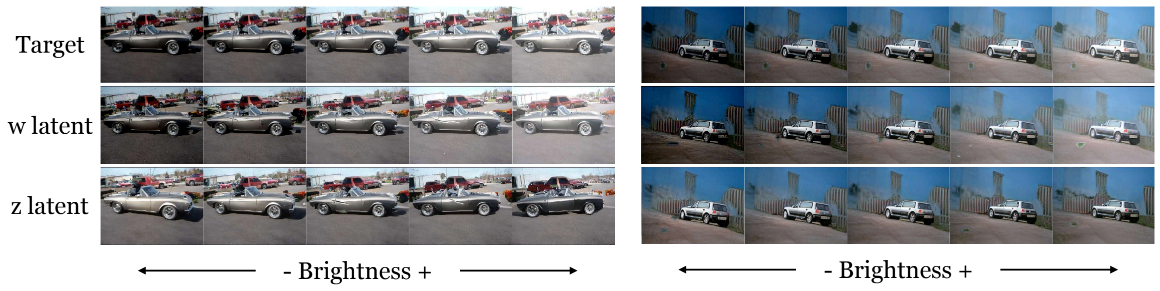


Figure A-19: **StyleGAN latent space comparisons.** Comparison of optimizing for color transformations in the Stylegan w and z latent spaces.

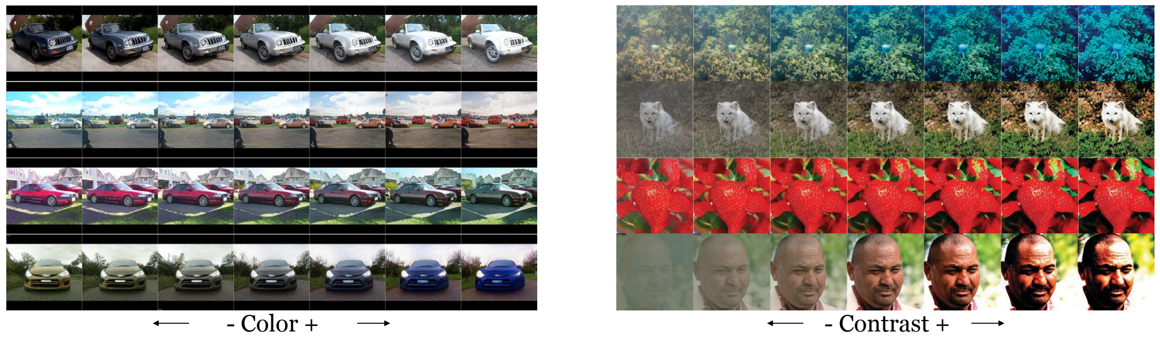


Figure A-20: **Latent walk variations.** Qualitative examples of optimizing for a color walk with a segmented target using StyleGAN in left column and a contrast walk for both BigGAN and StyleGAN in the right column.

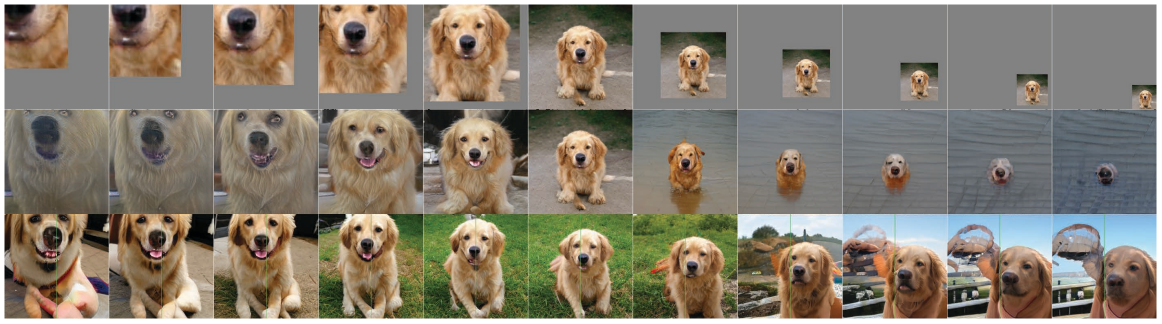


Figure A-21: **Combination of walk vectors.** Qualitative examples of a linear walk combining the zoom, shift X, and shift Y transformations. First row shows the target image, second row shows the result of learning a walk for the three transformations jointly, and the third row shows results for combining the separately trained walks. Green vertical line denotes image center.

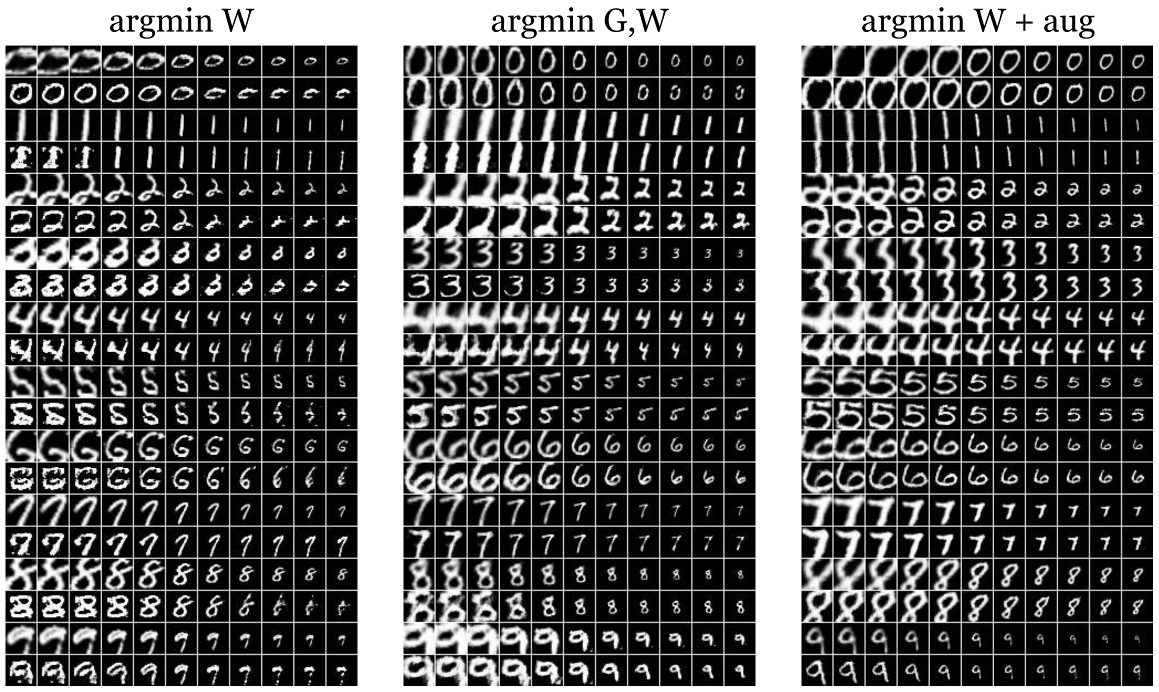


Figure A-22: **DCGAN zoom visualization.** Qualitative experiments on steerability with an MNIST DCGAN for the Zoom transformation. Odd rows are the target images and even rows are the learned transformations.

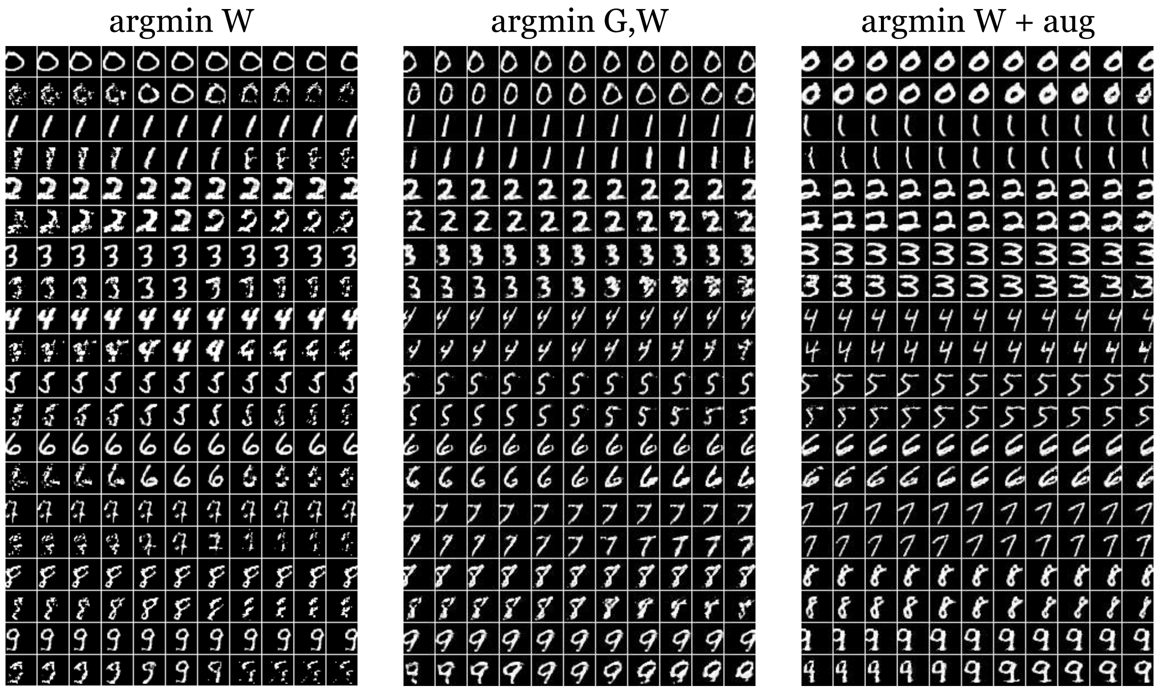


Figure A-23: **DCGAN shift visualization.** Qualitative experiments on steerability with an MNIST DCGAN for the Shift X transformation. Odd rows are the target images and even rows are the learned transformations.

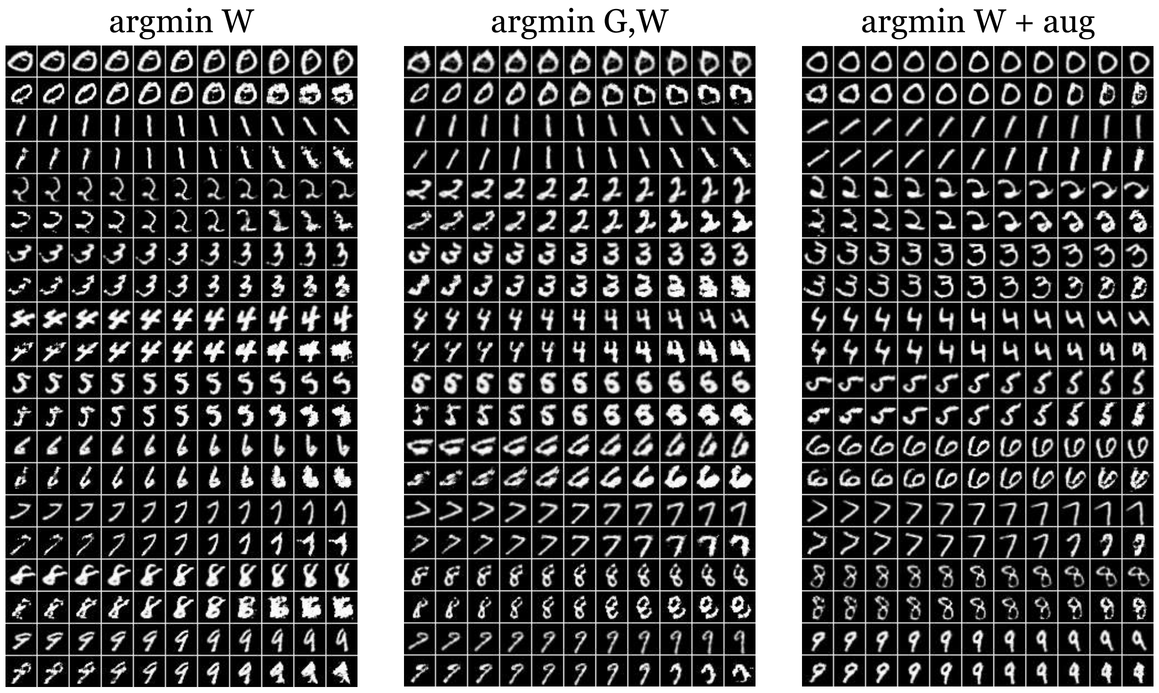


Figure A-24: **DCGAN rotate visualization.** Qualitative experiments on steerability with an MNIST DCGAN for the Rotate 2D transformation. Odd rows are the target images and even rows are the learned transformations.

Appendix B

Supplementary: Latent Composition

B.1 Supplementary Methods

B.1.1 Additional training details

The loss function of the encoder contains image loss terms to ensure that the output of the generator approximates the target image, and a latent recovery loss term to ensure that the predicted latent code matches the original latent code. On the image side, we use mean square error loss in conjunction with LPIPS perceptual loss [281]. The latent recovery loss depends on the type of GAN. Due to pixel normalization in ProGAN, we use a latent recovery loss based on cosine similarity, as the exact magnitude of the recovered latent code does not matter after normalization:

$$L_z = 1 - \frac{z}{||z||_2} \cdot \frac{E(x)}{||E(x)||_2}. \tag{B.1}$$

For StyleGAN, we invert to an intermediate latent space, as it is known that in this space semantic properties are better disentangled than in $\mathcal{Z}$ [107]. Furthermore, allowing the latents to differ on different scales has been shown to better capture the variability of real images [1]. During training, we therefore generate different latents for different scales, and train the encoder to estimate different styles, i.e. estimate $w \in \mathcal{W}^+$. Unlike the latent space of ProGAN, however, $w \in \mathcal{W}^+$ is not normalized to the hypersphere. Instead of a cosine loss, we therefore use a mean square error loss as the latent recovery loss:

$$L_w = ||w - E(x)||_2. \tag{B.2}$$

We train the encoders using a ResNet backbone (ResNet-18 for ProGAN, and ResNet-34 for Stylegan; He et al. [79]), modifying the output dimensionality to match the number of latent dimensions for each GAN. The encoders are trained with the Adam optimizer [114] with learning rate $lr = 0.0001$. Training takes from two days to about a week on a single GPU, depending on the resolution of the GAN. For ProGAN encoders, we use batch size 16 for the 256 resolution generators, and train for 500K batches. For the 1024 resolution generator, we use batch size 4 and 400K batches. We train the StyleGAN encoders for 680k batches (256 and 512 resolution) or 580k batches (1024 resolution), and add identity loss [188] with weight $\lambda = 1.0$ on the FFHQ encoder.

When training with masks, we take a small 6x6 patch of random uniform noise u, upsample to the generator's resolution, and sample a threshold t from the uniform distribution in range $[0.3, 1.0]$ to create the binary mask:

$$\begin{aligned} m &= \mathbb{1}\left[\text{Upsample}(u) > t\right] \\ x_m &= m \otimes x \end{aligned} \tag{B.3}$$

We also experimented with masks comprised of random rectangular patches [279], but obtained qualitatively similar results. At inference time, the exact shape of the mask does not matter: we can use hard coded rectangles, hand-drawn masks, or masks based on other pretrained networks. Note that the mask does not distinguish between input image parts – it is a binary mask with value 1 where the generator should try to reconstruct, and 0 where the generator should fill in the missing region.

B.1.2 Additional details on composition

When creating automated collages from image parts, we use a pretrained segmentation network [265] to extract parts from randomly sampled individual images. We manually define a set of segmentation class for a given image class, and, to handle overlap between parts, specify an order to these segmentation classes. To generate a collage, we then sample one random image per class. For church scenes, we use an ordering of (from back to front) sky, building, tree, and foreground layers – this ensures that a randomly sampled building patch will appear in front of the randomly sampled sky patch. For living rooms, the ordering we use is floor, ceiling, wall, painting, window, fireplace, sofa, and coffee table – again ensuring that the more foreground elements are layered on top. For cars, we use sky, building, tree, foreground, and car. For faces we order by background, skin, eye, mouth, nose, and hair. In Sec. B.2.4 we investigate using other methods to extract patches from images, rather than image parts derived from segmentation classes.

B.2 Supplementary Results

B.2.1 Additional applications

In the main text, we primarily focus on using the regression network as a tool to understand how the generator composes scenes from missing regions and disparate image parts. However, because the regressor allows for fast inference, it enables a number of other real-time image synthesis applications.

Image Completion From Masked Inputs. Using the masked latent space regressor in Eqn. 3.3, we investigate the GAN's ability to automatically fill in unknown portions of a scene given incomplete context. These reconstructions are done on parts of real images to ensure that the regressor is not simply memorizing the input, as could be the case with a generated input (the regressor is never trained on real images). For example, when a headless horse is shown to the masked regressor, the rest of the horse can be filled in by the GAN. In contrast, a regressor that is unaware of missing pixels (Eqn. 3.2; RGB) is unable to produce a realistic output. We show qualitative examples in Fig. B-1.

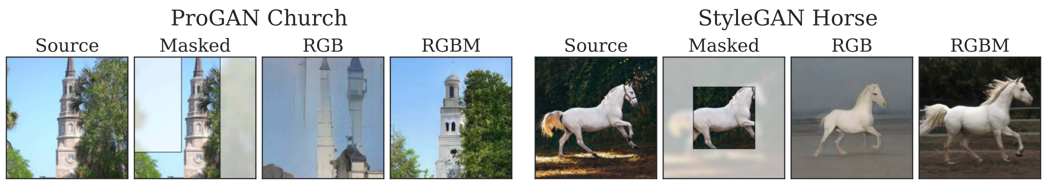


Figure B-1: **Image completion from masked inputs.** Given a masked real image, a regressor without knowledge of masked images (RGB) is unable to realistically reconstruct the scene, while the regressor trained on masks inputs inpaints in the unknown region in a way that is consistent with the given context (RGBM).

Multimodal Editing. Because the regressor only requires a single example of the property we want to reconstruct, it is possible to achieve multimodal manipulations simply by overlaying different desired properties on a given context image. Here, we demonstrate an example of adding different styles of trees to a church scene. In each of the context images, there is originally no tree on the right-hand side. We can add different types of trees to each context image simply by replacing some pixels with tree pixels from a different image, and performing image inversion to create the composite. Here, we use a rectangular region as a mask, rather than following the boundary of the tree precisely. However, note that, after inversion, the color of the sky remains consistent in the composite image, and so does the building color in second tree example. This image editing approach does not require learning separate clusters or having labelled attributes, and therefore can be done with a single pair of images without known cluster definitions, unlike the methods of Bau et al. [10] and Collins et al. [42]. Furthermore, unlike methods based on segmentation maps [172, 73], styles within each individual semantic category, e.g. the color of the sky, is also changeable based on the provided input to the encoder.

Attribute Editing With A Single Unlabeled Example. Typically in attribute editing using generative models, one requires a collection of labeled examples of the attribute to modify; for example, we take a collection of generated images with and without the attribute based on an attribute classifier, and find the average difference in their latent codes [98, 182, 68, 210]. Here, we demonstrate an approach towards attribute editing without using attribute labels (Fig. B-3). We overlay a single example

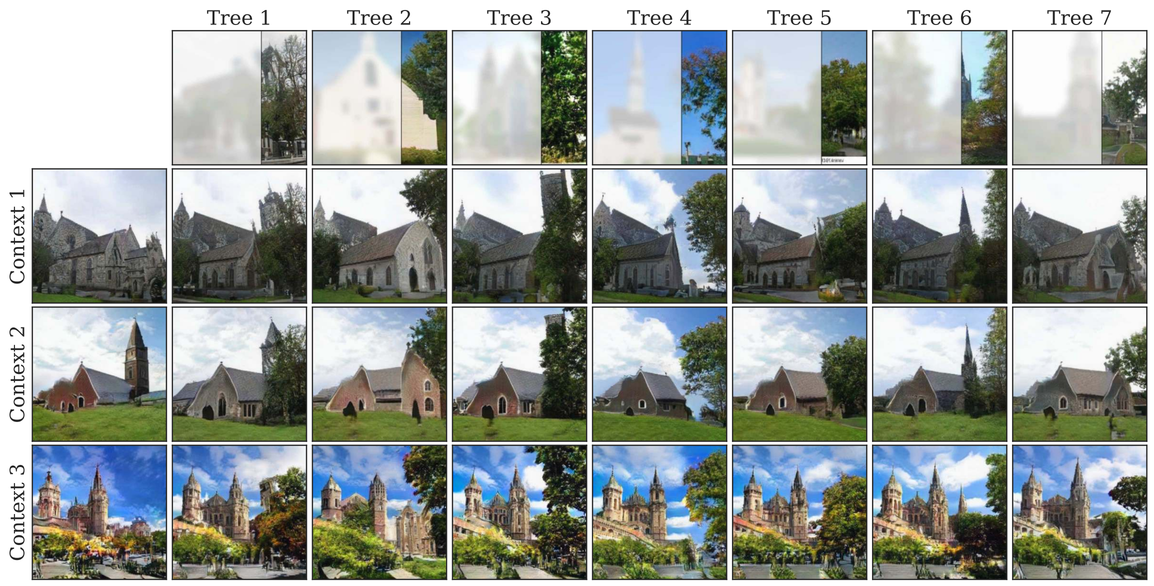


Figure B-2: **Latent regression for multimodal editing.** Using latent regression, we can perform multimodal editing to a context image. Each editing operation uses a single pair of images. Here, our editing mask is a rough rectangular box. Note how in case of *Tree 2*, the pasted region includes part of the church; the regressor attempts to include this part, and the resulting re-generated building therefore looks different from the building in the other examples.

of our target attribute (e.g. a smiling mouth or a round church tower), on top of the context image we want to modify, and encode it to get the corresponding latent code. We then subtract the difference between the original and modified latent codes, and add it to the latent code of the secondary context image: $z_{1,\text{modified}} - z_1 + z_2$. This bypasses the need for an attribute classifier, such as one that identifies round or square church towers.

Dataset Rebalancing. Because the latent regression inverter only requires a forward pass through the network, we can quickly generate a large number of reconstructions. Here, we use this property to investigate rebalancing a dataset (Fig. B-4). Using pretrained attribute detectors from Karras et al. [107], we first note that there is a smaller fraction of smiling males than smiling females in CelebA-HQ (first panel), although the detector itself is biased and tends to overestimate smiling in general compared to ground truth labels (second panel). The detections in the GAN generated images mimic this imbalance (third panel). Next, using a fixed set of generated

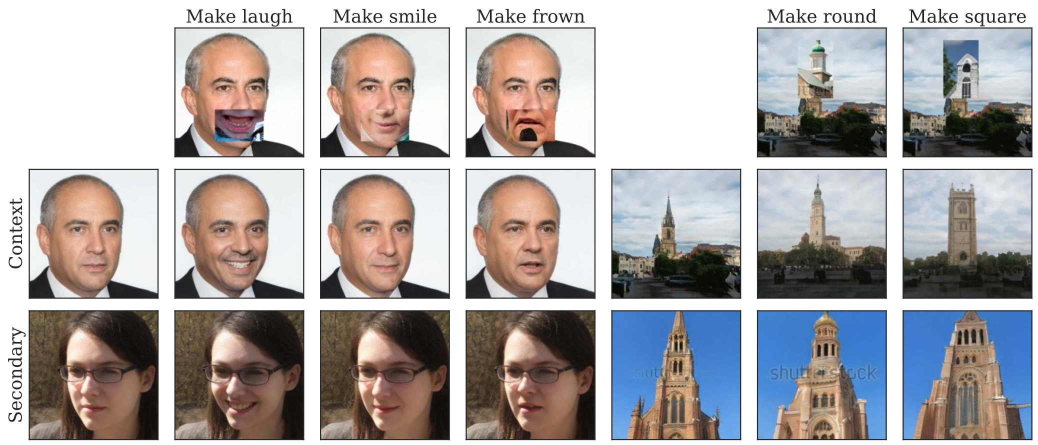


Figure B-3: **Transferring manipulations to unseen images.** We demonstrate the transfer capability of the latent code manipulations. From a single pair of images, we can compute a manipulation vector and apply it to a secondary image, which edits the secondary image accordingly.

images, we randomly swap the mouth regions among them in accordance to our desired proportion of smiling rates – we give each face a batch of 16 possible swaps and taking the one that yields the strongest detection as smiling/not smiling based on our desired characteristic (if no swaps are successful, the face is just used as is). We use a hardcoded rectangular box region around the mouth, and encode the image through the generator to blend the modified mouth to the face. After performing swapping on generated images, this allows us rebalance the smiling rates, better equalizing smiling among males and females (fourth panel). Finally, we use the latent regression to perform mouth swapping on the dataset, and train a secondary GAN on this modified dataset – this also improves the smiling imbalance, although the effect is stronger in males than females (fifth panel). We note that a limitation of this method is that the rebalanced proportion uses the attribute detector as supervision, and therefore a biased or inaccurate attribute detector will still result in biases in the rebalanced dataset.

B.2.2 Comparing composition with latent space interpolation

In the main text, we compare our composition approach to two types of interpolations – latent α-blending and pixel α-blending – on a living room generator. Here, we show

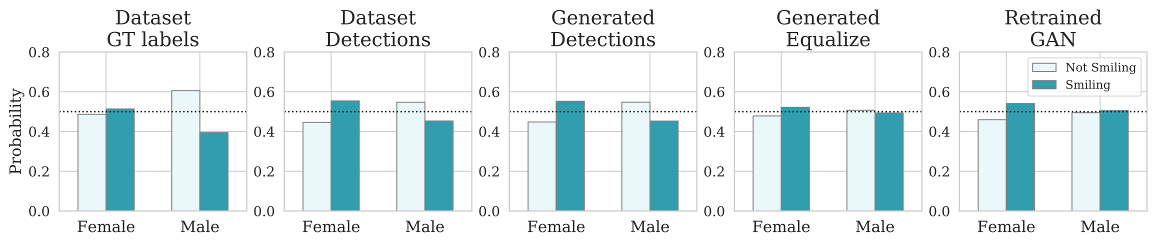


Figure B-4: **Dataset rebalancing using latent space regression.** From pretrained attribute detectors, a smaller fraction of males smile than females in CelebA-HQ, and the GAN samples mimic this trend. By swapping mouths in the GAN samples, we can equalize male and female smiling rates. Next, we do this swap and invert operation on the dataset and retrain a GAN on the modified dataset, which also improves balance in smiling rates although the effect is stronger for the male category.

equivalent examples in church scenes. In Fig. B-5, we demonstrate a "tree" edit, in which we want the background church to remain the same but trees to be added in the foreground. Compared to latent and pixel α-blending, the composition approach better preserves the church while adding trees, which we quantify using masked L1 distance. Similarly, we can change the sky of context scene – e.g. by turning a clear sky into a cloudy one in Fig. B-6, where again, using composition better preserves the details of the church as the sky changes, compared to α-blending.

In Fig B-7, we show the result of changing the smile on a generated face image. For the smile attribute, we also compare to a learned smile edit vector using labelled images from a pretrained smile classifier. We additionally measure the the facial embedding distance using a pretrained face-identification network* – the goal of the interpolation is to change the smile of the image while reducing changes to the rest of the face identity, and thus minimize embedding distance. Because the mouth region is small, choosing the interpolation weight α by the target area minimally changes the interpolated image ($\alpha > 0.99$), so instead we use an $\alpha = 0.7$ weight so that all methods have similar distance to the target. While the composition approach and applying learned attribute vector perform similarly, learning the attribute vector requires labelled examples, and cannot perform multimodal modifications such as applying different smiles to a given face.

*`https://github.com/timesler/facenet-pytorch`

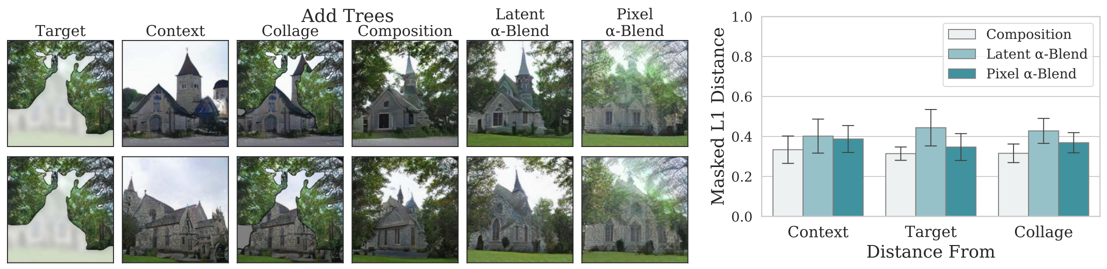


Figure B-5: **Quantifying reconstruction fidelity: adding trees.** Comparison of image editing outcomes when adding trees (Target) to different scenes (Context). The overlay of these two components is shown in Collage, where we use a mask to indicate missing regions. We compare (1) Composition: using the encoder and generator to blend the images together to (2) Latent α-blend: interpolation in the latent codes of the encoded images and (3) Pixel α-blend: interpolation in pixel space, which then passes through the encoder to pull the image closer to the image manifold. (Right) Over 500 randomly sampled images, we find that the composition approach is better able to match properties of both the context and target images compared to the two alpha-blending techniques.

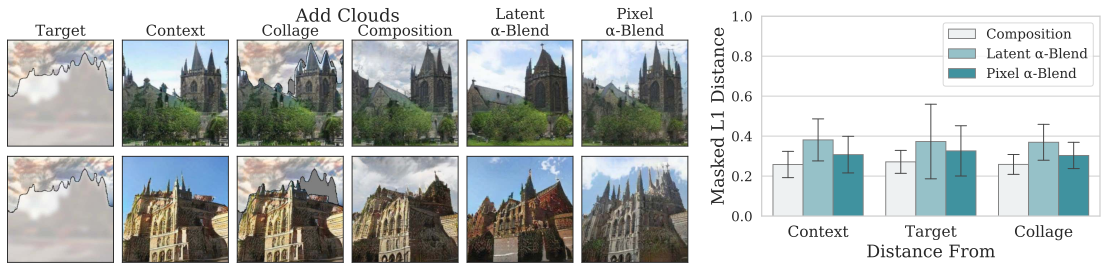


Figure B-6: **Quantifying reconstruction fidelity: changing sky.** Comparison of image editing outcomes when changing the sky of a scene, using similar encoder-based Composition, Latent α-blend, and Pixel α-blend methods as above.

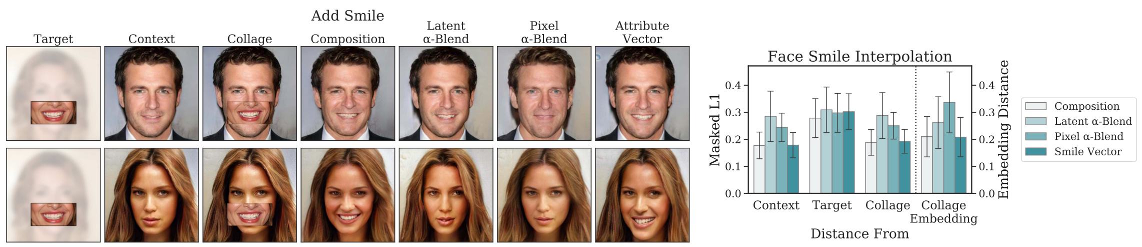


Figure B-7: **Quantifying reconstruction fidelity: add smile.** Comparison of image editing outcomes using the same encoder-based Composition, Latent α-blend, and Pixel α-blend methods as above when adding a smile to a face. We also compare these approaches to the modifications produced by edit vector learned from attribute labels.

B.2.3 Loss ablations

When training the regression network, we use a combination of three losses: pixel loss, perceptual loss [281], and a latent recovery loss. In this section, we investigate the reconstruction result using different combinations of losses on the ProGAN church generator. In the first case, we do not enforce latent recovery, so the encoded latent code does not have to be close to the ground truth latent code but the encoder and generator just need to reconstruct the masked input image. In the second case, we investigate omitting the perceptual loss, and simply use an L2 loss in image space. Since the encoders are trained with L2 loss and the VGG variant of perceptual loss, we evaluate with L1 reconstruction and the AlexNet variant of perceptual loss. Training with all losses leads to reconstructions that are more perceptually similar (lower LPIPS loss) compared to the two other variants, while per-pixel L1 reconstruction is not greatly affected (Tab. B.1). We show qualitative examples of the reconstructions on masked input in Fig. B-8.

Table B.1: **Encoder network ablations.** Table of masked L1 and LPIPS-Alexnet reconstruction errors for encoder networks trained with all losses, no latent loss, and no perceptual loss.

	All Losses	**No Latent**	**No LPIPS**
L1	0.194	0.194	0.197
LPIPS(alex)	0.291	0.305	0.318

B.2.4 Additional composition results

Comparing different composition approaches. In Fig B-9, we show examples of extracted image parts, the collage formed by overlaying the image parts, and the result of poisson blending the images according to the outline of each extracted part. We further compare to variations of the encoder setup, where (1) RGB: the encoder is not aware of missing pixels in the collaged input, (2) RGB Fill: we fill the missing pixels with the average color of the background layer, and (3) RGBM: we allow the encoder and generator to inpaint on missing regions. Table B.2 shows image quality metrics of FID (lower is better; Heusel et al. [82]) and density and coverage (higher

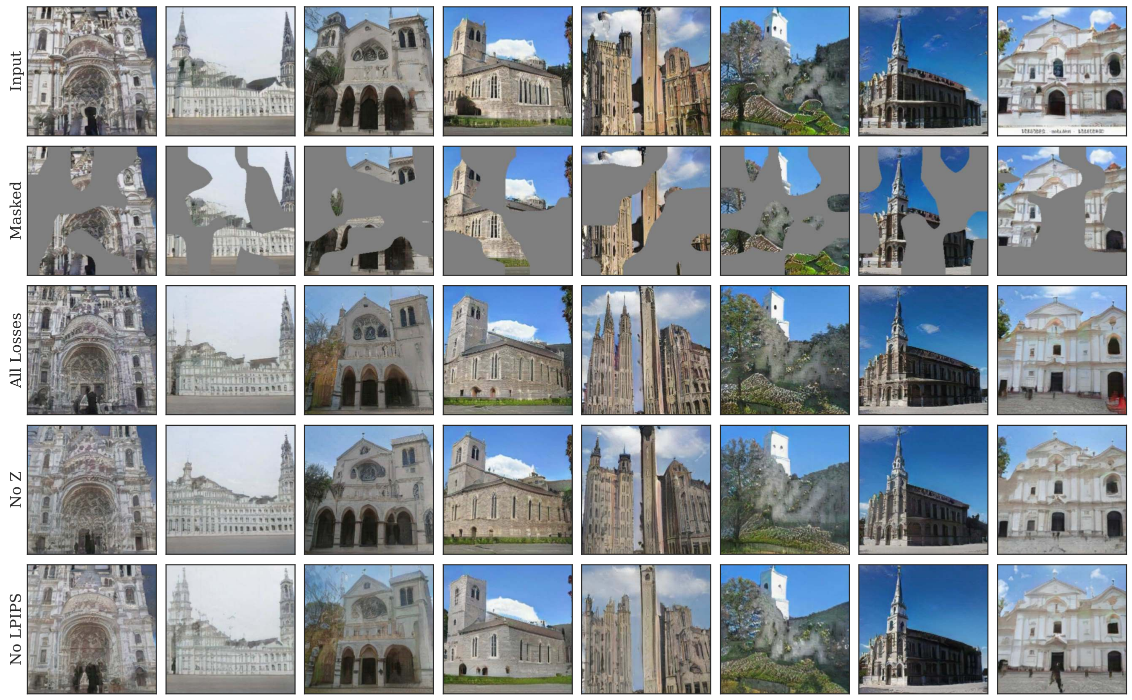


Figure B-8: **Visualization of encoder variations.** Qualitative examples of reconstructions from masked inputs on encoders trained with different combinations of loss terms.

is better; Naeem et al. [161]) and masked L1 reconstruction (lower is better) for each generator and domain. To obtain feature representations for the density and coverage metrics, we resize all images to 256px and use pretrained VGG features prior to the final classification layer [218]. While the composite input collages are highly unrealistic with high FID and low density/coverage, the inverted images are closer to the image manifold. Of the three inversion methods, the RGBM inversion tends to yield lower FID and higher density and coverage, while minimizing L1 reconstruction error. We compare the GAN inversion methods to Poisson blending [177], in which we incrementally blend the 4-8 image layers on top of each other using their respective masks. As Poisson blending does not naturally handle inpainting, we do not mask the bottom-most image layer, but rather use it to fill in any remaining holes in the blended image. We find that Poisson blending is unable to create realistic composites, due to the several overlapping, yet misaligned, image layers used to create the composites.

Comparing different image reconstruction generators. How are image priors different across different image reconstruction pipelines? Our encoder method relies on

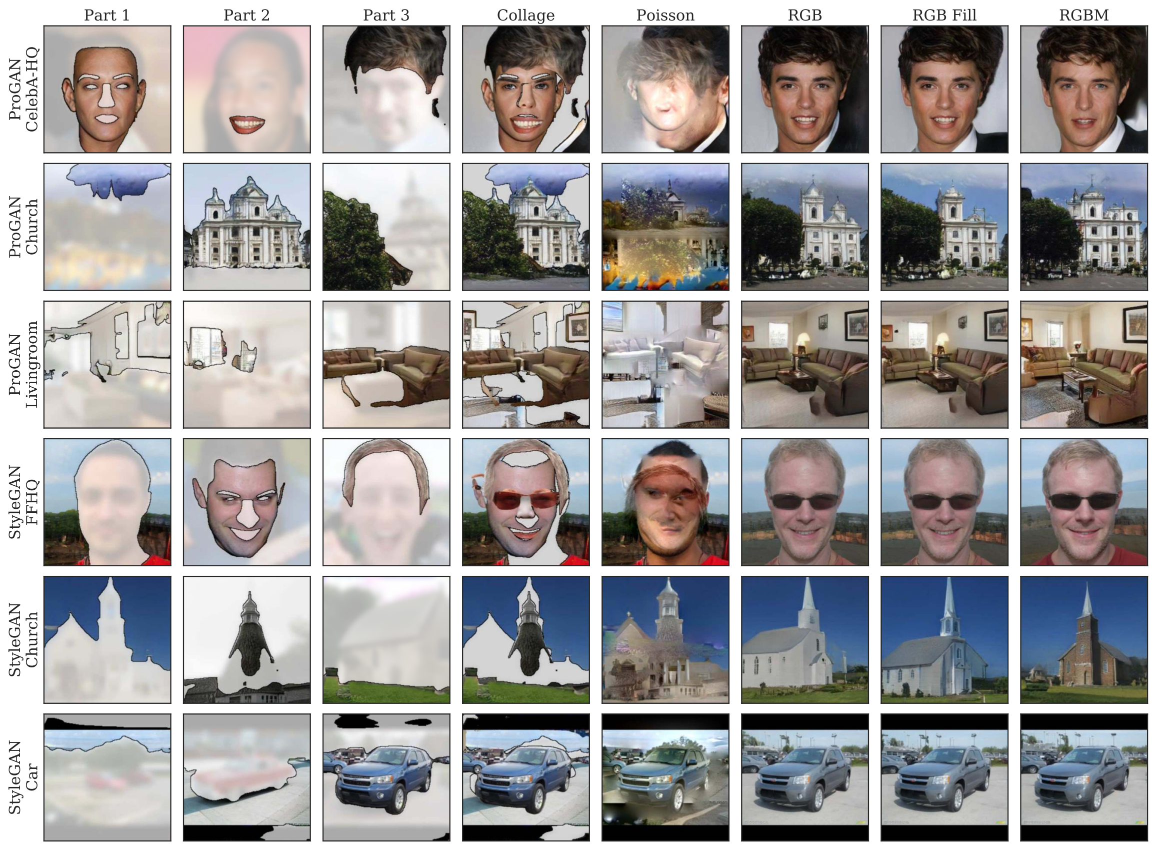


Figure B-9: **Qualitative examples of automatic image composition.** We extract parts of sampled images and overlay them to form a rough collage. We compare Poisson blending the image parts (we use the bottom-most image layer to fill in any remaining holes in the collaged image) to encoder-based methods that invert the collage through the generator, either without knowledge of missing pixels (RGB and RGB Fill methods), or with the objective of inpainting the missing regions (RGBM). While the RGB and RGB Fill methods also demonstrate an alignment-correcting effect, we find quantitatively that the RGBM encoder tends to have lower FID over 50K samples (Tab. B.2).

a pretrained generator, and trains an encoder network to predict the generator's latent code of a given image. Therefore, it can take advantage of the image priors learned by the generator to keep the result close to the image manifold. Here, we compare to different image reconstruction approaches, such as autoencoder architectures or optimization methods rather than feed-forward inference. We construct the same set of input collages using parts of real images and compare a variety of image reconstruction methods encompassing feed-forward networks, pretrained GAN models, encoder networks. Since some reconstruction methods are optimization-based and thus take several minutes, we use a set of 200 images. We then compute the L1

Table B.2: **Quantitative comparison of automated collaging.** The latent regressor and generator pulls the unrealistic composite images closer to the real manifold, yielding high density and coverage and lower FID on output, compared to the collaged inputs and Poisson blending. For ProGAN models, we use PyTorch models from Bau et al. [11]; for StyleGAN models, we use a Pytorch conversion of the Tensorflow models from Karras et al. [107].

Model	Metric	GAN Samples	GAN Reconstructions	Collage	Collage Filled	Poisson Blended	RGB Inverted	RGB Filled Inverted	RGBM Inverted
ProGAN Church	FID	8.01	6.72	21.20	27.35	24.12	10.85	12.90	9.40
	Density	0.94	1.04	0.15	0.15	0.45	1.02	0.92	1.23
	Coverage	0.78	0.82	0.22	0.22	0.41	0.71	0.74	0.78
	Masked L1	–	–	–	–	–	0.26	0.28	0.26
ProGAN Living Room	FID	13.17	10.46	72.82	69.95	47.84	19.50	17.70	14.90
	Density	0.69	0.86	0.01	0.03	0.28	0.98	0.81	1.05
	Coverage	0.63	0.69	0.02	0.02	0.26	0.60	0.59	0.64
	Masked L1	–	–	–	–	–	0.33	0.34	0.30
ProGAN CelebA-HQ	FID	10.43	12.38	81.82	82.09	53.76	16.24	15.35	17.41
	Density	1.27	1.55	0.11	0.14	0.28	1.14	1.19	1.69
	Coverage	0.83	0.84	0.03	0.03	0.15	0.72	0.74	0.78
	Masked L1	–	–	–	–	–	0.26	0.26	0.24
StyleGAN Church	FID	3.90	4.27	15.66	22.89	16.92	6.13	6.75	6.09
	Density	0.88	1.06	0.15	0.14	0.41	1.31	1.31	1.50
	Coverage	0.85	0.87	0.26	0.25	0.50	0.85	0.86	0.87
	Masked L1	–	–	–	–	–	0.32	0.32	0.30
StyleGAN Car	FID	2.31	2.98	17.44	141.21	15.38	6.23	5.87	5.38
	Density	1.07	1.32	0.37	0.42	0.61	1.69	1.53	1.51
	Coverage	0.94	0.94	0.44	0.36	0.54	0.92	0.91	0.91
	Masked L1	–	–	–	–	–	0.32	0.34	0.30
StyleGAN FFHQ	FID	2.86	6.27	92.15	92.00	36.54	26.25	25.63	24.09
	Density	1.17	1.61	0.16	0.17	0.26	1.67	1.58	2.01
	Coverage	0.90	0.89	0.02	0.02	0.24	0.67	0.67	0.74
	Masked L1	–	–	–	–	–	0.30	0.30	0.28

reconstruction error in the valid region of the input, density (measures proximity to the real image manifold; Naeem et al. [161]), and FID (measures realism; Heusel et al. [82]; but note that we are limited to small sample sizes due to optimization time).

For the church domain, we first compare to four methods that do not rely on a pretrained GAN network; rather, the generator and encoder is jointly learned using an autoencoder-like architecture. (1) We train a CycleGAN [293] between image collages and the real image domain, creating a network that is explicitly trained for image composition in an unpaired manner, as there are no ground-truth "composited" images for the randomly generated image collages. (2) We use a pretrained SPADE [172] network which creates images from segmentation maps, but information about object style (e.g. color and texture) is lost in the segmentation input. (3) We use a pretrained inpainting model that is trained to fill in small holes in the image [272], but does not correct for misalignments or global color inconsistencies between image parts. (4) We train Deep Image Prior (DIP) networks [242] which performs test-time optimization on an autoencoder to reconstruct a single image, where using a masked loss allows it to inpaint missing regions.

Next, we use the ProGAN and StyleGAN2 pretrained generators, and experiment with different ways of inverting into the latent code. Methods that leverage a pretrained GAN for inversion, but are optimization-based rather than feed-forward include (5&6) LBFGS methods [138] on ProGAN and StyleGAN, which iteratively optimizes for the best latent code starting from the best latent among 500 random samples, (7) Multi-Code Prior [72], which combines multiple latent codes in the intermediate layers of the GAN, and (8) a StyleGAN2 projection method using perceptual loss [107]. For all optimization-based GAN inversion methods, we modify the objective with a masked loss to only reconstruct valid regions of the input.

We use our trained regressor network for the remaining comparisons. (9&10) We use our ProGAN and StyleGAN regressors to encode the input image as initialization, and then perform LBFGS optimization. (11&12) We use our ProGAN and StyleGAN regressors in a fully feed-forward manner.

Tab. B.3 summarizes the methods and illustrates the tradeoff between reconstruction (L1), realism (Density and FID), and optimization time. Due to the unrealistic nature of the input collages, a method that reduces reconstruction error is less realistic, whereas a more realistic output may offer a worse reconstruction of the input. Furthermore, methods that are not feed-forward incur substantially more time per image. Fig B-10 shows qualitative results, where in particular the third example is a challenging input where the lower part of the tower is missing. The two encoders demonstrate an image prior in which the bottom of the tower is reconstructed on output. While DIP can inpaint missing regions, it cannot leverage learned image prior. CycleGAN can fill in missing patterns, but with visible artifacts, whereas SPADE changes the style of each input component. Iteratively optimizing on a pretrained generator can lose semantic priors as it optimizes towards an unrealistic input.

On the face domain, we compare (1) the inpainting method of Yu et al. [272] pretrained on faces, (2) the Im2StyleGAN method [1] which optimizes within the $\mathcal{W}+$

Table B.3: **Quantifying reconstruction/realism tradeoff on churches.** On a set of 200 input collages constructed from random image parts, we compare compositional properties of several image reconstruction approaches including methods based on autoencoder-style networks that do not leverage a pretrained GAN, methods based on optimization on the latent code of a pretrained GAN, and methods based on encoder networks to regress the latent code. Autoencoder-based and optimization-based methods can achieve lower reconstruction error (L1; lower is better), but are less realistic (Density; higher is better). Another tradeoff is optimization time, as feed-forward methods are orders of magnitude faster than optimization-based approaches. We also report FID (lower is better), but our sample size is limited due to the latency of optimization-based approaches.

Inversion Method	**GAN-Based**	**Encoder-Based**	**Feed-Forward**	**L1 (↓)**	**Density (↑)**	**FID (↓)**	**Time (s) (↓)**
1. CycleGAN			✓	0.14	0.43	74.63	0.10
2. SPADE			✓	0.50	1.01	137.96	0.07
3. Inpainting			✓	0.00	0.34	72.86	0.02
4. DIP				0.04	0.18	86.44	98.90
5. ProGAN LBFGS	✓			0.23	1.04	53.64	188.94
6. StyleGAN LBFGS	✓			0.13	0.50	84.97	368.92
7. Multi-Code Prior	✓			0.13	0.21	104.34	477.93
8. StyleGAN Projection	✓			0.23	0.25	62.62	86.87
9. ProGAN Encode+LBFGS	✓	✓		0.23	0.88	49.60	66.55
10. StyleGAN Encode+LBFGS	✓	✓		0.15	0.38	64.29	176.96
11. Ours: ProGAN Encoder	✓	✓	✓	0.32	0.79	55.11	0.02
12. Ours: StyleGAN Encoder	✓	✓	✓	0.36	1.86	48.08	0.03

latent space of StyleGAN, and (3) the ALAE model [178] which jointly trained the encoder and generator. More similar to our approach, (4&5) the In-domain encoding and diffusion methods [291] encodes and optimizes for a latent code of StyleGAN that matches the target image. (6) We modify and retrain the Pixel2Style2Pixel network (PSP; Richardson et al. [188]), which also leverages the StyleGAN generator, to perform regression with arbitrarily masked regions. The PSP network uses a feature pyramid to predict latent codes. (7&8) We use our regressor network on ProGAN and StyleGAN, which uses a ResNet backbone and predicts the latent code after pooling all spatial features.

Qualitatively, we find that the optimization-based Im2StyleGAN [1] algorithm is not able to realistically inpaint missing regions in the input collage. While the ALAE autoencoder [178] exhibits characteristics of blending the collage into a cohesive output image, the reconstruction error is higher than that of the GAN-based approaches. The In-domain encoder method [291] does not correct for misalignments in the inputs, resulting in low density, although the subsequent optimization step is able to further reduce L1 distance (Fig. B-11). The PSP network modified with the masking objective is conceptually similar to our regressor; we find that it is better able to reconstruct the input image, but produces less realistic output. This suggests that an encoder which processes the input image globally before predicting the latent code output can help retain realism in output images. We measure reconstruction (L1) and realism (Density and FID) over 200 samples in Tab. B.4.

Table B.4: **Quantifying reconstruction/realism tradeoff on faces.** We construct 200 input collages from random face image parts, and compare to several image reconstruction methods. Again, we find a tradeoff between better reconstruction (low L1) and better realism (higher Density, lower FID), due to the imperfect nature of the input collages. There is no ground truth image that perfectly reconstructs the input yet is realistic.

Inversion Method	**GAN-Based**	**Encoder-Based**	**Feed-Forward**	**L1 (↓)**	**Density (↑)**	**FID (↓)**
1. Inpainting			✓	0.02	0.33	90.17
2. Im2StyleGAN	✓			0.21	0.20	106.79
3. ALAE			✓	0.33	0.47	80.13
4. In-domain Encoder	✓	✓	✓	0.21	0.44	77.43
5. In-domain Diffusion	✓	✓		0.12	0.40	73.74
6. Masked PSP	✓	✓	✓	0.15	0.52	73.06
7. Ours: ProGAN Encoder	✓	✓	✓	0.27	1.55	57.20
8. Ours: StyleGAN Encoder	✓	✓	✓	0.26	1.21	64.33

Composing images using alternative definitions of image parts. In the main text, we focus on creating composite images using a pretrained segmentation network. However, we note that the exact process of extracting image parts does not matter, as the encoder and generator form an image prior that will remove inconsistencies in the input. In Fig. B-12 we show composites generated using the output of a pretrained saliency network [140], and in Fig. B-13 we show compositions using hand-selected compositional parts, where the parts extracted from each image do not have to correspond precisely with object boundaries.

Editing with global illumination consistency. A property of the regressor network is that it enforces global coherence of the output, despite an unrealistic input, by learning a mapping from image to latent domains that is averaged over many samples. Thus, it is unable to perform exact reconstructions of the input image, but rather exhibits error-correcting properties when the input is imprecise, e.g. in the case of unaligned parts or abrupt changes in color. In Fig. B-14, we investigate the ability of the regressor network to perform global adjustments to accommodate a desired change in lighting – such as adding a reflections or changing illumination *outside of the manipulated region* – to maintain realism at the tradeoff of higher reconstruction error.

Improving compositions on real images. As the regressor network is trained to minimize reconstruction on average over all images, it can cause slight distortions on any given input image. To retain the compositionality effect of the regressor network, yet better fit a *specific* input image, we can finetune the weights of the regressor towards the given input image. Generally, a few seconds of finetuning suffices (30 optimizer steps; < 5 seconds), and subsequent editing on the image only requires a forward pass. We demonstrate this application in Fig. B-15.

Random composition samples. We show additional random samples of the generated composites across the ProGAN and StyleGAN2 generators for a variety of image domains in Fig. B-16 and Fig. B-17.

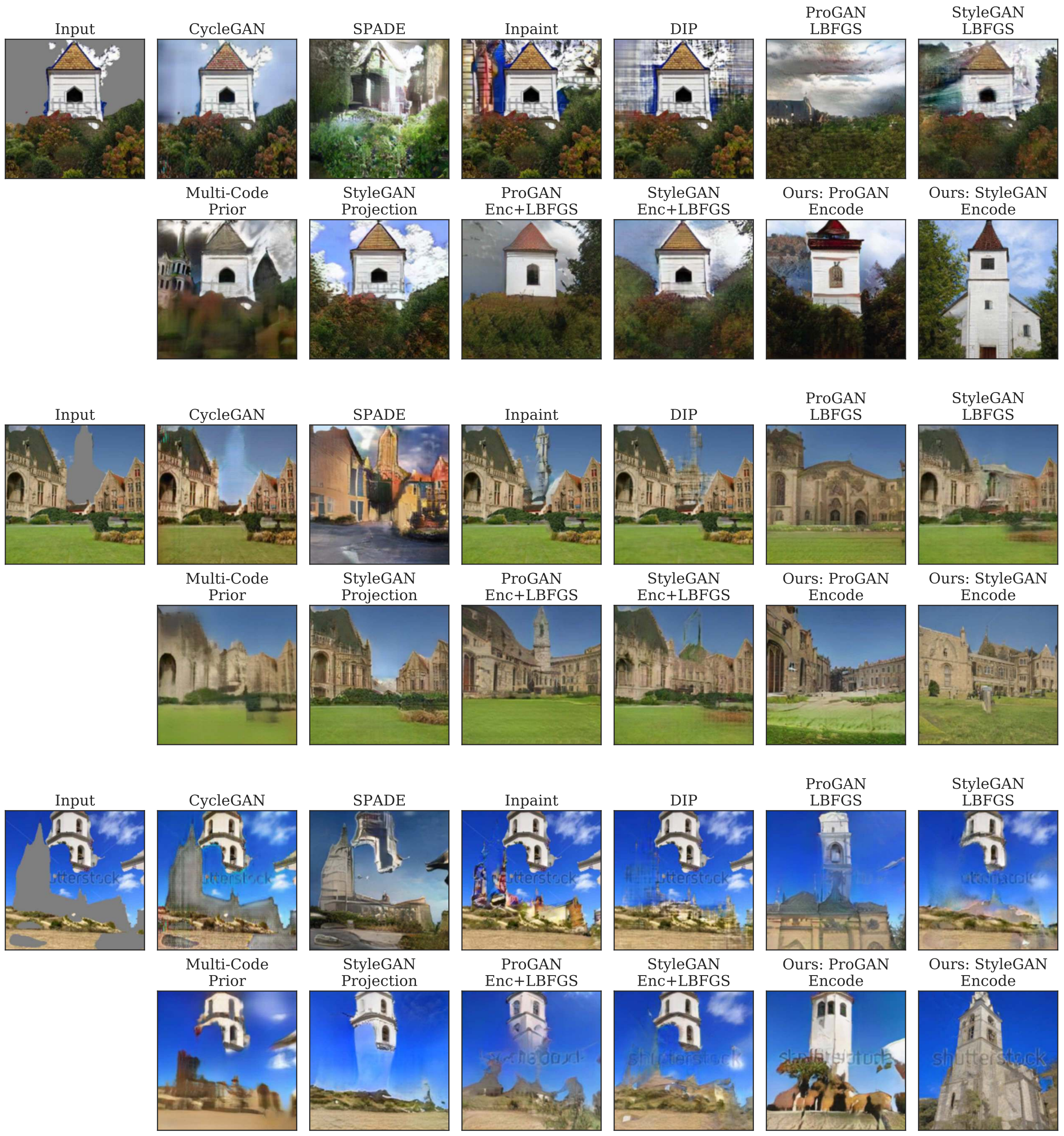


Figure B-10: **Comparing across encoder-decoder or generator-only setups.** For each example, we show the input collage created from randomly selected parts of images (Input), and four autoencoder-based methods (CycleGAN, SPADE, Inpaint, DIP) in which the encoder and decode is trained jointly; as these are fully convolutional they lack global semantic priors. Next, we show four optimization-based methods (ProGAN & StyleGAN LBFGS, Multi-Code Prior, StyleGAN Projection); these can better match the target collage after optimizing for reconstruction, but they are orders of magnitude slower, making real-time interaction infeasible, and also lose semantic priors (as in the "floating tower" in the third example). We then show the combination of our encoder with LBFGS optimization, and our encoder using only a forward pass. While the feed-forward approach retains the tower prior and enables real-time interaction, its can distort the input image parts, illustrating a tradeoff between reconstruction and realism.

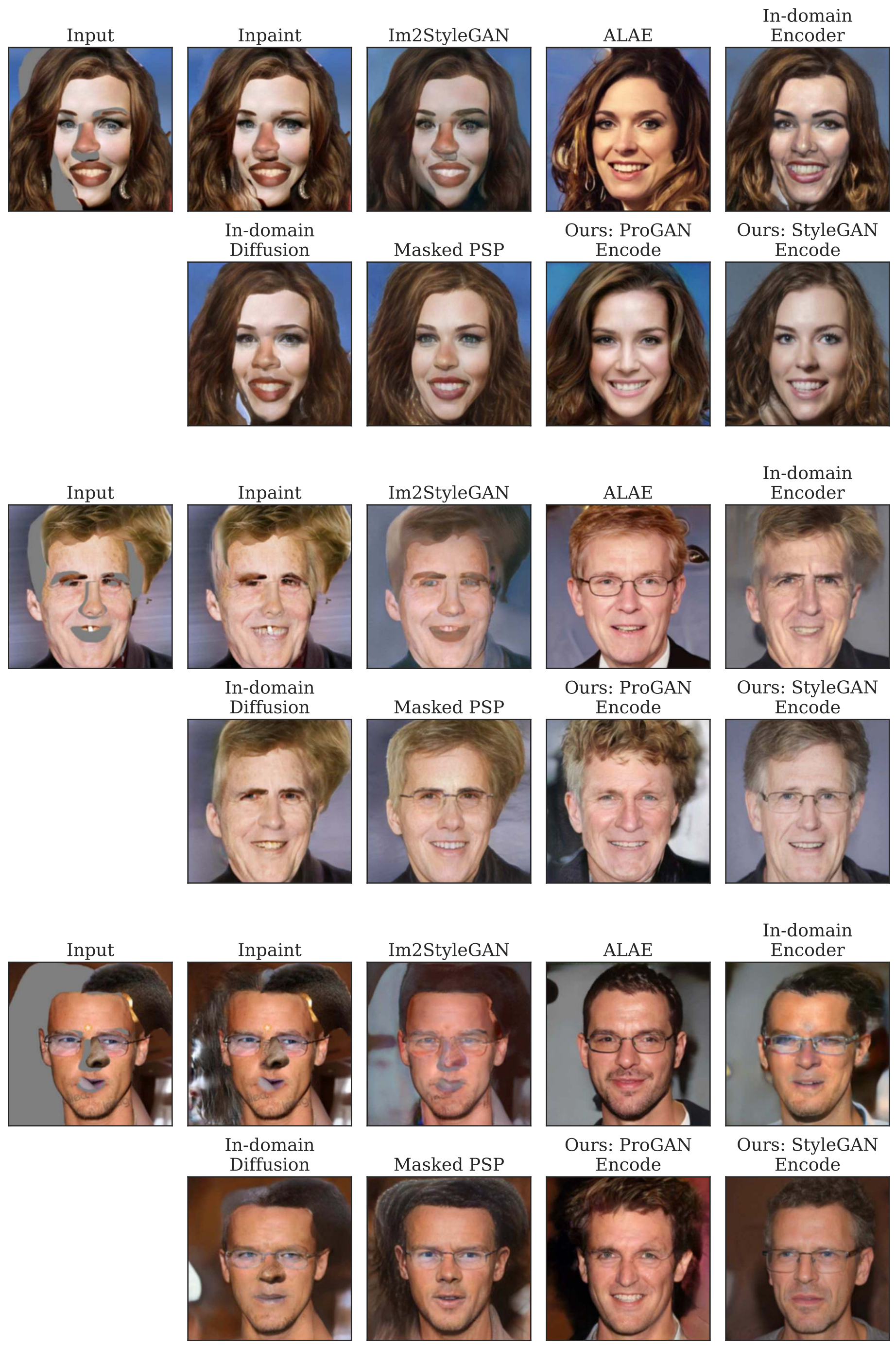


Figure B-11: **Face composites across GAN-based generators.** We show the input image, a collage from parts of random images (Input). We compare to Inpainting, Im2StyleGAN which iteratively optimizes for a latent code, the ALAE autoencoder where the generator and encoder is trained jointly, the In-domain encoder and diffusion which encodes and optimizes the latent, a masked version of the Pixel2Style2Pixel (PSP) network, and our masked encoder approaches (ProGAN Encoder and StyleGAN encoder).

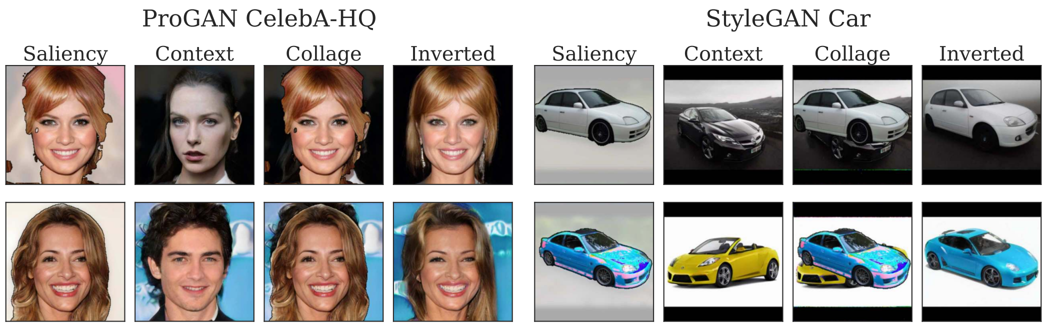


Figure B-12: **Automatic composition using saliency.** Qualitative examples of automatic image composition using a pretrained saliency network to generate input collages rather than a segmentation network. Note that in the second car example, the collage overlays the blue car with the yellow car still visible, but inversion via the encoder corrects this inconsistency.

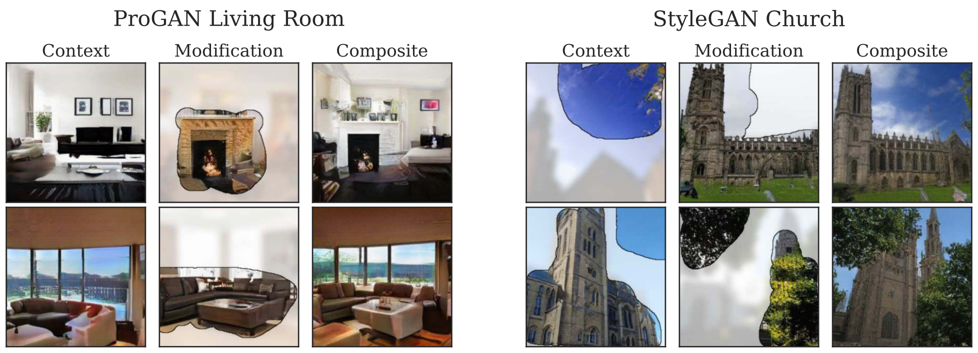


Figure B-13: **Collages from user inputs.** Qualitative examples of collages based on user-selected regions and the corresponding generator output. Note that the selected regions do not have to coincide neatly with object boundaries, and the generator will blend inconsistencies in the inputs together.

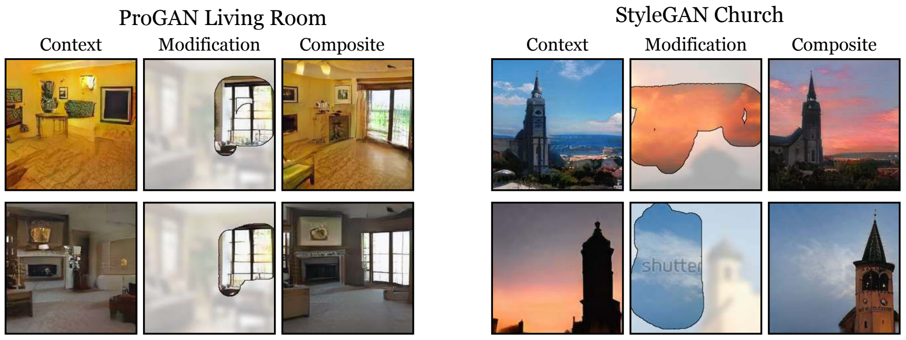


Figure B-14: **Non-local editing effects.** We investigate the capabilities of our instance-based regression and editing approach to modify lighting effects, similar to [12]. Editing lighting induces non-local changes, as the network must also change regions outside of the modified region to ensure global consistency – for example the reflection of the window on the floor (ProGAN Living Room), or changes in the illumination of the building (StyleGAN Church) – indicating a tradeoff between reconstructing the input versus creating a realistic image overall.

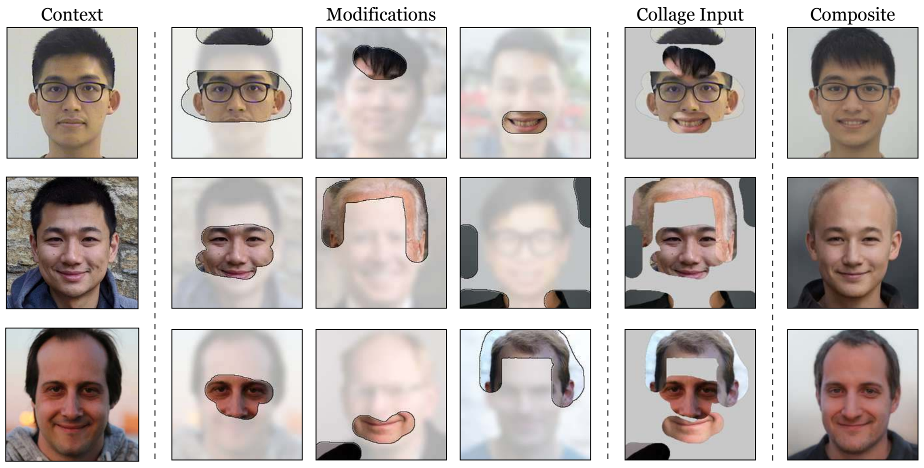
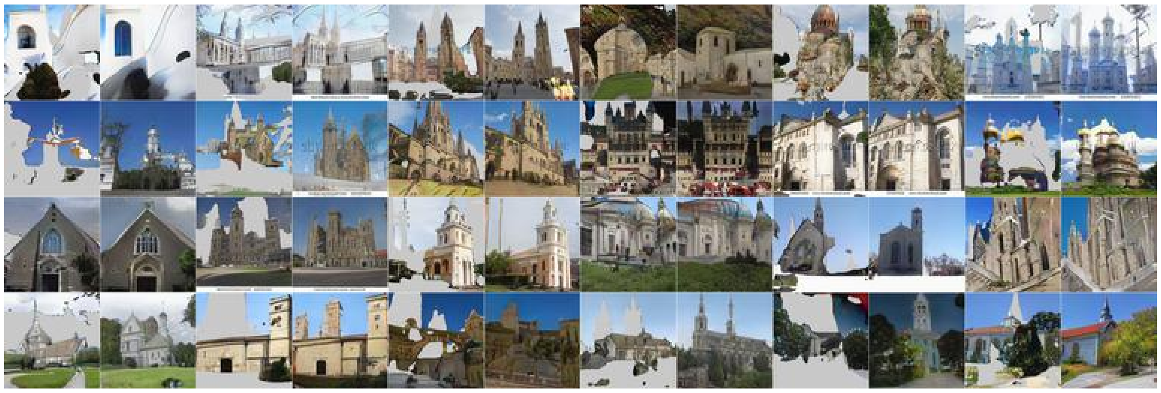
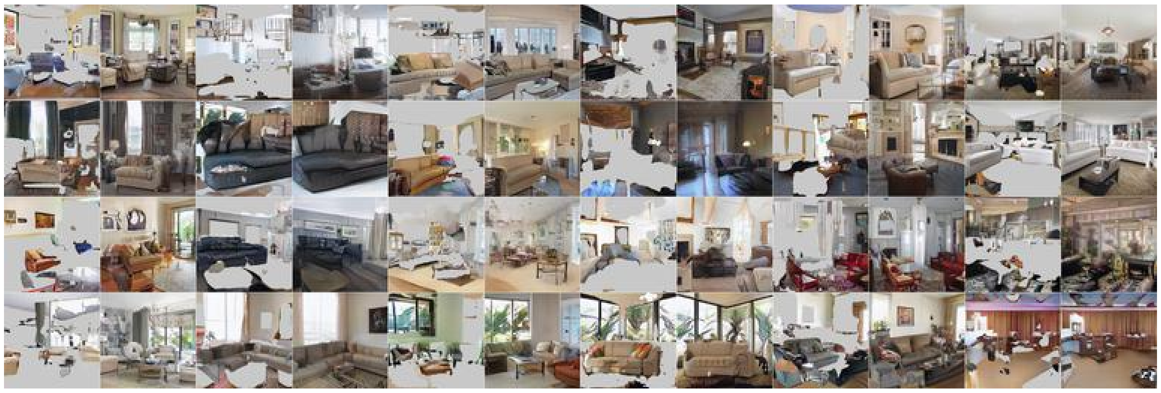


Figure B-15: **Finetuned encoder for collaging on real faces.** To improve the reconstruction towards real faces, yet preserve the properties of real-time composition, we can slightly finetune the regressor towards a context image (30 gradient steps, < 5 seconds). Subsequent modifications to the context image can be performed using just a feed-forward pass, enabling fast editing to create image composites.

(a) ProGAN CelebA-HQ

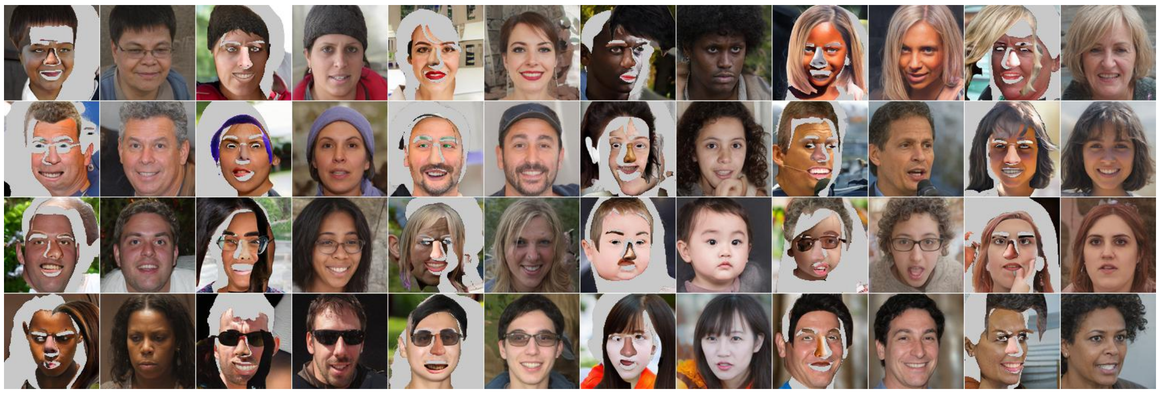


(b) ProGAN Church

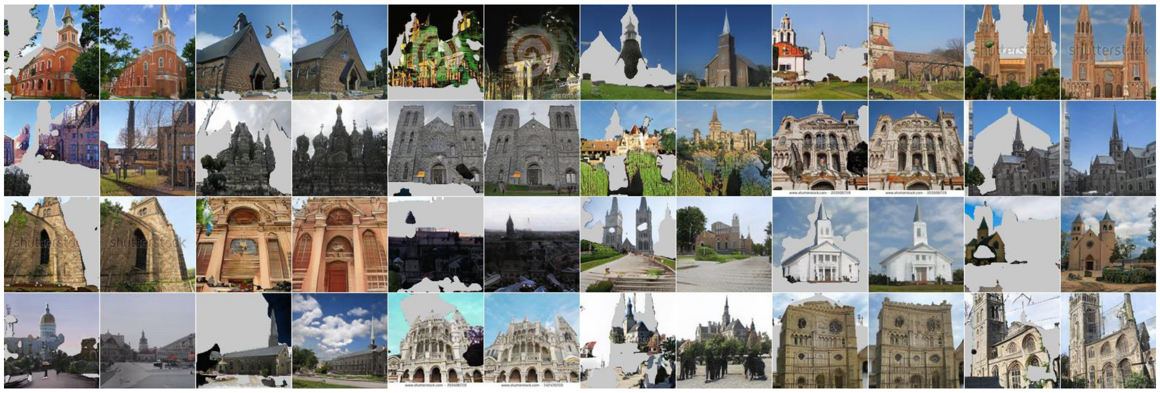


(c) ProGAN Living Room

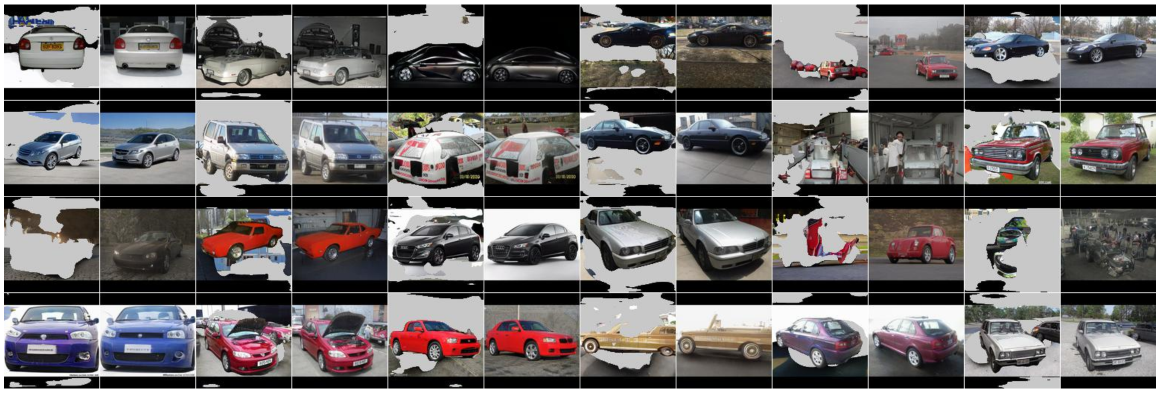
Figure B-16: **Additional collages: ProGAN.** Random samples of automatically generated colleges using a pretrained ProGAN generator.



(a) StyleGAN FFHQ



(b) StyleGAN Church



(c) StyleGAN Car

Figure B-17: **Additional collages: StyleGAN.** Random samples of automatically generated colleges using a pretrained StyleGAN generator.

B.2.5 Additional part variation results

Here, we show additional qualitative results similar to those in Fig. 3-8 in the main paper. In each case, the heatmap shows appearance variation when changing the part marked in red. In Fig. B-18, it can be seen that the variation is usually strongest in the face part that is changed, indicating that the composition of face parts learned by the model is a good match for our intuitive understanding of a face. In Fig. B-19, we vary a single superpixel of a car. The resulting variations show regions of the images that commonly vary together (such as the floor, the body of the car, or the windows), which can be interpreted as a form of unsupervised part discovery.

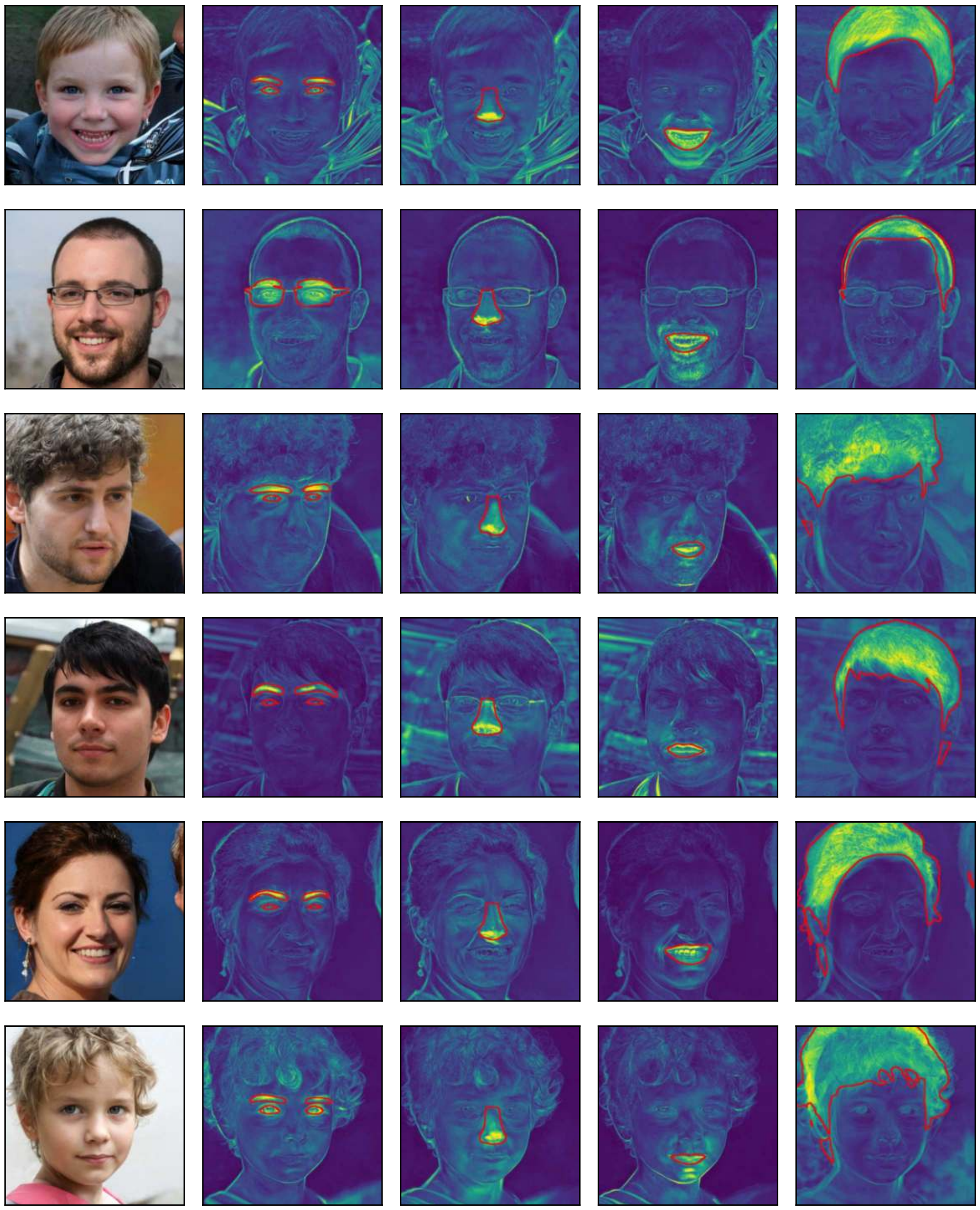


Figure B-18: **Part variation visualization: faces.** When changing semantic parts of a face (such as the eyes or the hair, shown in red), the resulting change in the image is often limited to those parts, indicating that the model parses faces into meaningful and intuitive components.

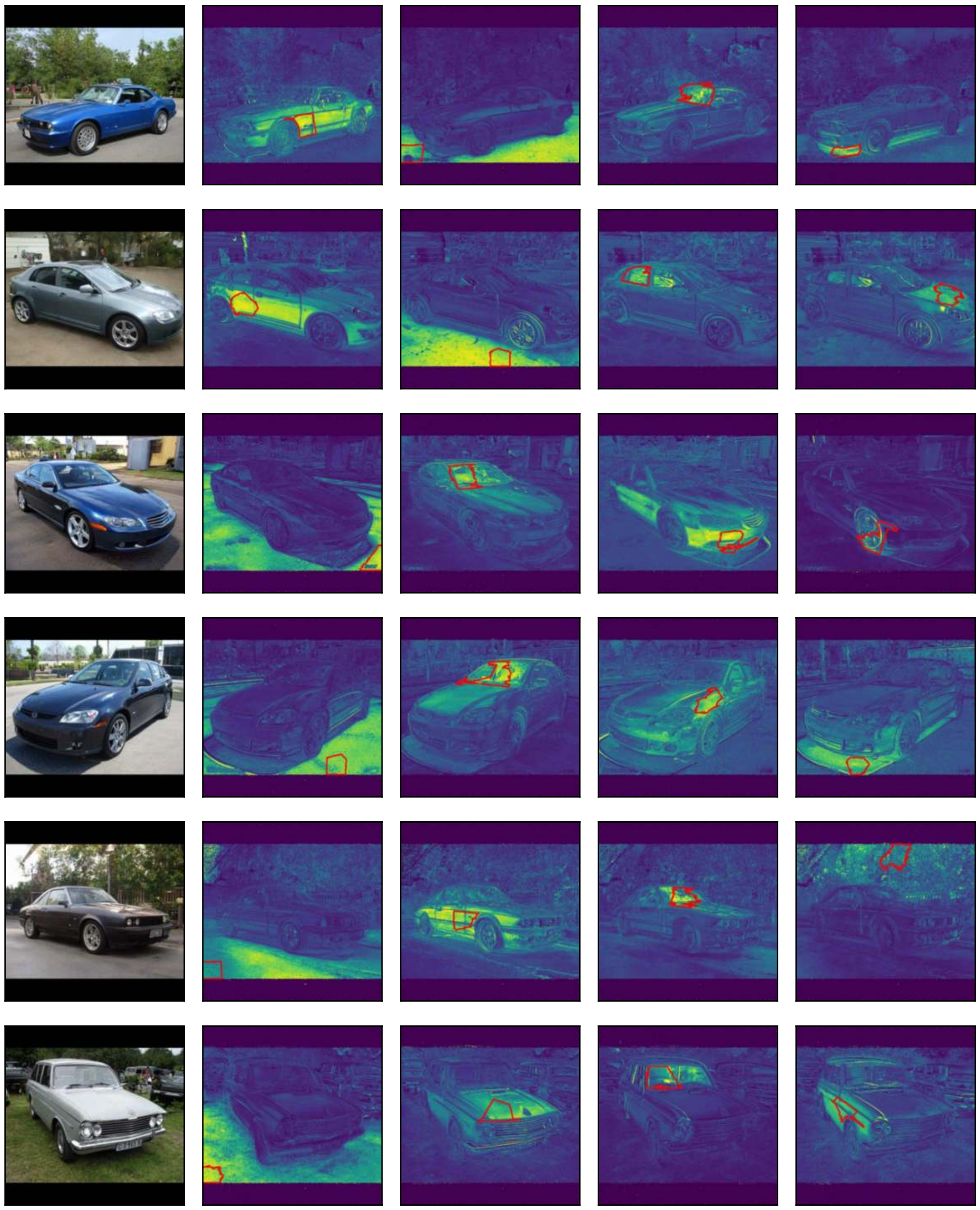


Figure B-19: **Part variation visualization: cars.** Our encoder-generator pair can be used to detect correlated parts of objects. Here, the value indicates how much a pixel changes when the superpixel shown in red is changed; oftentimes, semantic segments emerge, such as the car body, windows, the street, or the background.

Appendix C

Supplementary: Anyres-GAN

In the supplementary, we first demonstrate an extension of our approach towards panorama generation (Section C.1). In Section C.2, we provide additional qualitative and quantitative comparisons to baselines (super-resolution methods, discrete-resolution and oracle generators), explore additional model variations, and investigate the detectability of our method using an off-the-shelf forensics method. We provide implementation details in Section C.3.

C.1 Panorama Generation Extension

Our default training setup assumes that we use low-resolution images to learn global context and patches from high-resolution images to learn details. An alternative setup to learn from patches directly, *without ever knowing the entire global context*. One such scenario is panorama generation, where a large-scale dataset would be much more difficult to obtain than single images. We investigate this setup on the Mountains domain, in which the generator is tasked with synthesizing a panorama from landscape images, without training directly on panoramas. Accordingly, we modify the $[0, 1] \times [0, 1]$ coordinate grid to $[-\pi, \pi] \times [0, 1]$, and enforce continuity on the endpoints by using a sine and cosine encoding prior to Fourier feature embedding. At training time, we sample a “slice” of the coordinate grid for generation corresponding to a random viewing angle, but at inference time the entire panorama can be synthesized by specifying the full grid of coordinates. In this case, we find that it is important to use a *cross-frame* discriminator, in which the discriminator straddles the boundary between two generated slices to enable seamless boundaries in the panorama. Qualitative results are shown in Fig C-1. At inference time, we can spatially interpolate the w latent code with arbitrary spacing, which generates seamless infinite landscapes (Fig. C-2).

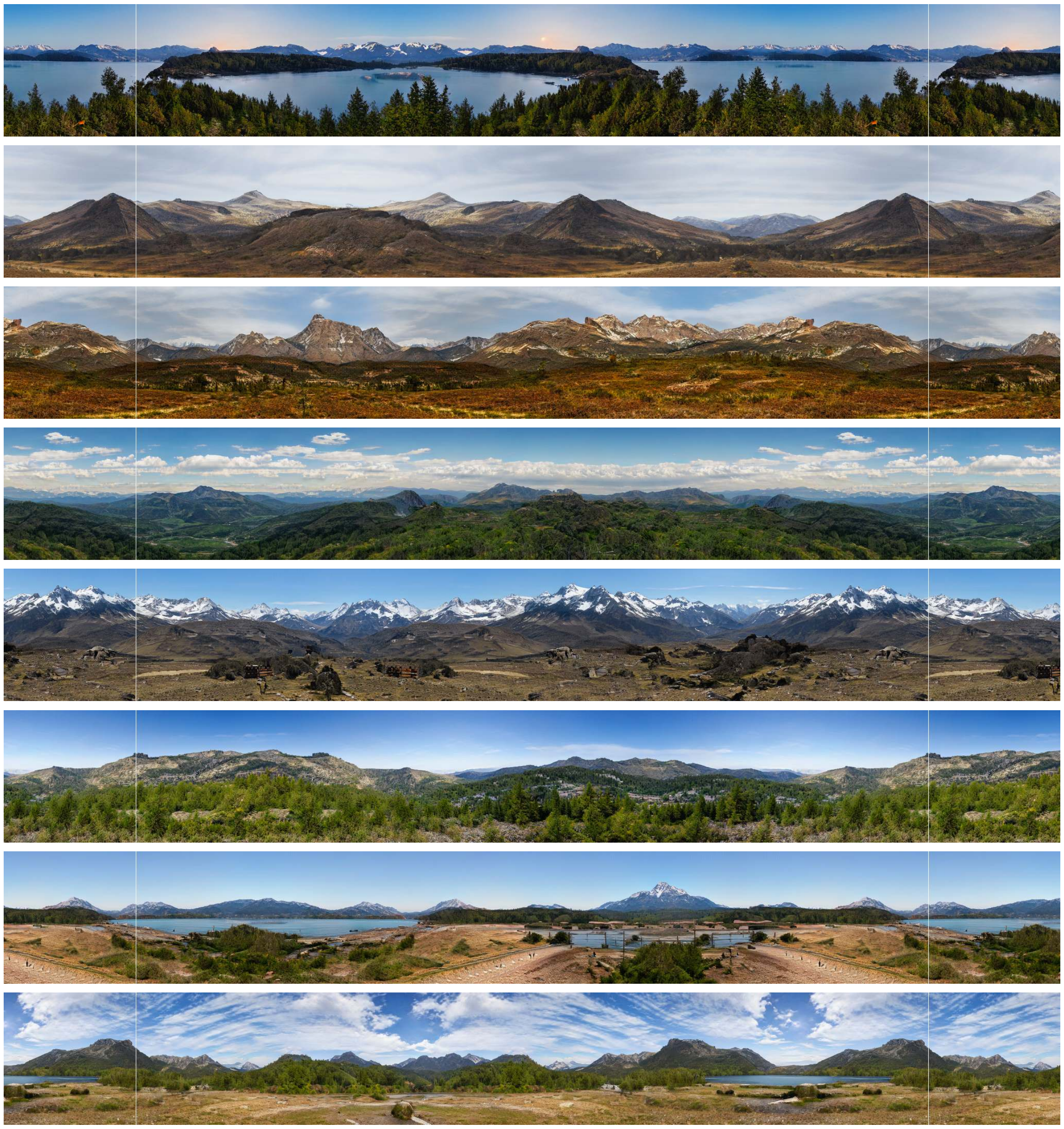


Figure C-1: **Panorama generation from patches.** We modify our training framework to train without the global image context. We map our coordinate grid to $[-\pi, \pi] \times [0, 1]$ and use a cross-frame discriminator to enable seamless transitions between patches. The model is trained with FOV $= 60°$. The vertical white line indicates a full $360°$ revolution.

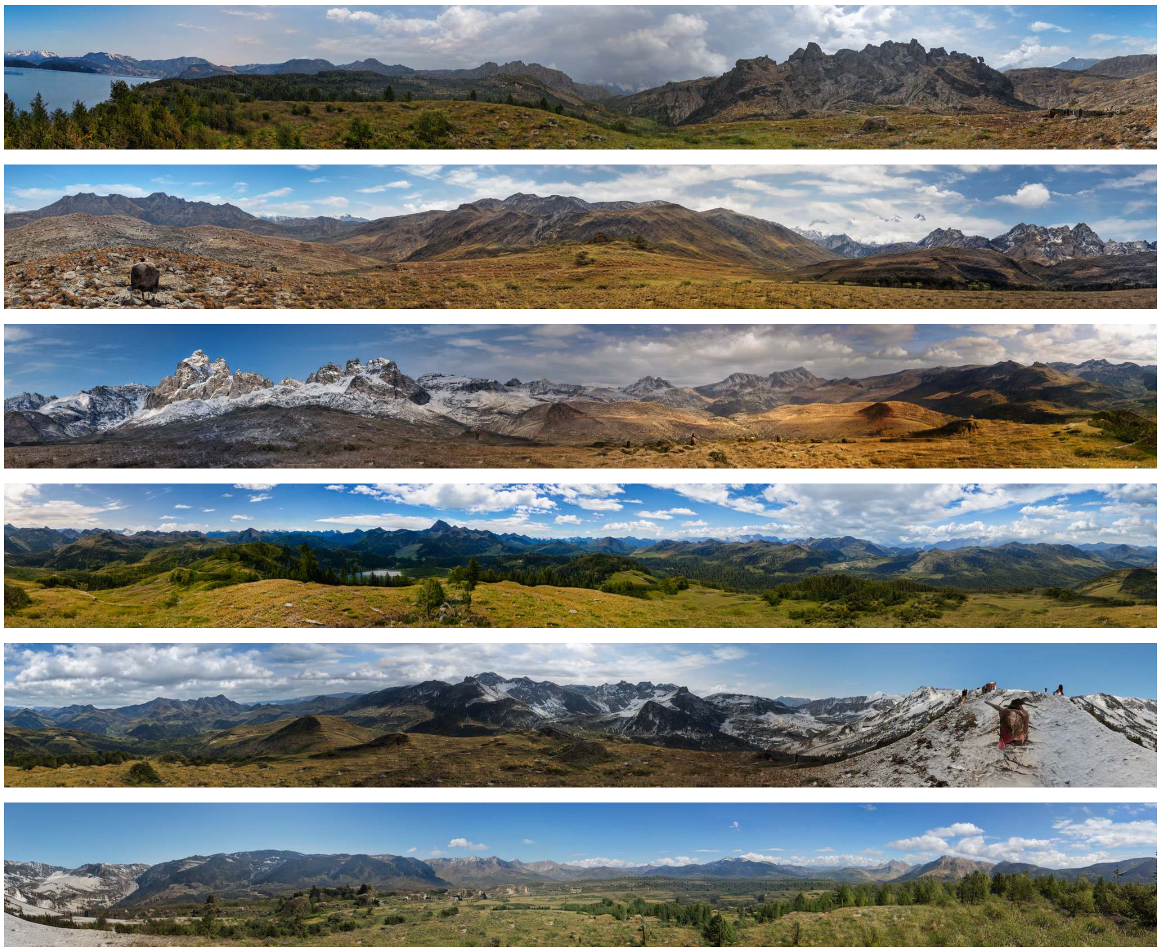


Figure C-2: **Infinite image generation.** By spatially adjusting the latent code only at inference time, the same model is capable of generating infinite landscapes, shown on multiple lines here for visibility.

C.2 Experiments

C.2.1 Dataset collection

To collect our varied-size dataset, we scrape image collections from Flickr photo groups (Tab. C.1). In cases where a standard fixed-resolution dataset is available (*e.g.* LSUN Churches [271]), we seek to find photos that approximately match the domain of the standard dataset. Due to domain mismatches between LSUN and the photos scraped from

Table C.1: **Dataset sources.** Image sources for construction of our varied-resolution datasets.

Domain	Flickr Source
Church	Church Exteriors
Mountains	Mountains Anywhere
Birds	Birding in the Wild

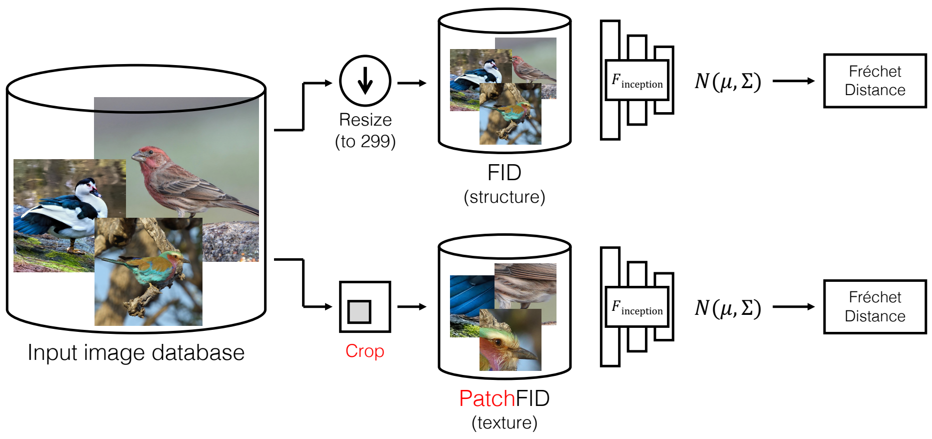


Figure C-3: **Preprocessing for FID and patch-FID (pFID).** FID evaluates global structure, but downsamples all images to a common 299px size which ignores higher resolution details. pFID takes images crops instead rather than downsampling to capture texture realism at higher resolutions. We use both to measure structure and texture properties.

Flickr, we manually filter the collected images to approximately match the LSUN domain, which remains tractable for the few thousand HR images used in the patch-based training phase. As is standard practice [235, 173], and to not violate license permissions, we will release the image IDs but not the images directly.

C.2.2 Patch-FID

We describe details of our Patch FID metric, introduced in Section 4.4.1 of the main paper. The metric is aimed at better capturing the realism of details at high resolution by avoiding downsampling to capture texture details. In our setting, we have a smaller number of real images present at various resolutions. Standard FID, which focuses on global structure, does not capture these varied-resolution details. We modify the FID pipeline to avoid downsampling global images at higher resolutions to a fixed 299 pixel width. Instead, we randomly sample patches of size p from real images at global scale s and locations $c_{\mathbf{v},s}$, and generate the corresponding patch $G\left(z, c_{\mathbf{v},s}, s\right)$. The number of samples is crucial to getting an accurate estimate in FID calculation [39], and 50,000 is typically used. A benefit of this patch-sampling procedure also means that we can obtain the necessary large number of patches for FID computation, more than the

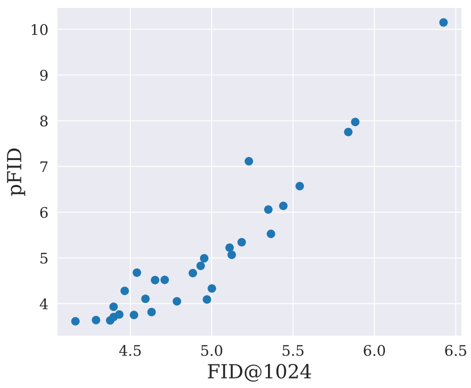


Figure C-4: **pFID vs FID@1024.** These two metrics are largely correlated. FID@1024 generates images at 1024 resolution and then downsamples images, which assumes a fixed-size dataset and ignores details present at higher resolution. On other domains where a ground-truth HR dataset is not available, we primarily use pFID to measure sample quality, particularly in high-resolution detail regions.

available number of real images in the HR dataset. In the Mountains generator, where the patch size p is larger than 299, we subsequently also select a random 299-pixel crop from the patches to compute the image features. Because this avoids downsampling the generated content, we find that it is more sensitive to image quality at high resolutions. Using the full FFHQ dataset as ground-truth, we find that our patch-FID metric is largely correlated to the standard FID numbers at 1024 resolution (Fig. C-4). Therefore, we use Patch-FID as a metric of sample quality on our datasets collected from Flickr, when a full high-resolution dataset of images all at the same resolution is not available.

C.2.3 Additional quantitative results

In Table 4.3 of the main text, we report comparisons of our method and off-the-shelf super-resolution methods using the patch-FID metric. We report additional metrics in Table C.2 here, including the FID at base resolution (the result of the pretraining step), and FID at a higher resolution after downsampling all images in the HR dataset (between 5k-10k images, which is lower than the typical 50k used to compute FID) to a common size. Notably, the base resolution FID is largely similar before and after patch-based training, and in the case of FFHQ and Birds, patch-based training at higher resolutions even improves the low-resolution FID. Without direct multi-scale training, however, the fixed-size model obtained from the pretraining step does not naturally generalize to higher resolutions. We find that our pFID metric is more discriminative to differences in image quality at higher resolutions. In particular,

Table C.2: **Alternative FID evaluation metrics.** We primarily use pFID, which avoids downsampling synthesized content and evaluates multiscale patches, as an evaluation metric. Here, we also report FID at the base resolution from the result of fixed-size pretraining (Fixed-Size), and compare to global FID metrics after downsampling the HR images to a common size. On FFHQ, we also tried applying GFP-GAN [253] which is a facial super-resolution model.

	FFHQ6K			**Church**			**Birds**			**Mountain**		
	FID		pFID	FID		pFID	FID		pFID	FID	pFID	
	256	1024	random	256	1024	random	256	512	random	1024	down	random
Fixed-Size	3.71	33.80	52.95	3.39	242.10	146.24	3.92	12.69	55.42	3.09	13.42	46.20
Upsample	–	11.70	17.29	–	14.21	80.48	–	7.67	30.57	–	4.53	20.00
LIIF	–	7.05	22.93	–	18.66	83.88	–	7.29	30.19	–	4.55	23.10
Real-ESRGAN	–	19.04	16.92	–	12.26	23.04	–	8.51	16.10	–	7.60	19.05
GFP-GAN	–	19.15	16.27	–	–	–	–	–	–	–	–	–
Ours	3.34	4.06	2.96	3.84	6.98	9.89	3.78	6.29	6.52	3.14	4.33	7.99

the LIIF super-resolution model tends to obtain better FID@1024 (corresponding to super-resolving generated images to 1024 resolution and computing FID) compared to Real-ESRGAN, but the outputs are visually blurry. Because our pFID does not perform downsampling, it can better capture this blurriness, reflected in an increased pFID. In another variant, we compute pFID on patches of size 1024 synthesized by the Mountain generator, and then subsequently downsample them, which we denote as pFID (down), rather than cropping. Again, we find that this downsampling operation can obscure image deterioration at higher resolution, producing artificially lower FID scores compared to pFID computed without downsampling.

C.2.4 Comparison to powers-of-two synthesis

Using the same set of generator weights, our model can synthesize images at a specified scale by simply providing the corresponding s and $c_{\mathbf{v},s}$ inputs. On the other hand, other methods for multi-resolution synthesis [136, 106, 45, 105] generate images that are iteratively enlarged by a factor of two, by adding additional network layers. These methods are typically introduced to impove training stability. For this baseline, we modify the recent Anycost-GAN [136] framework to fit our varied-size training setting. Specifically, we downsample all images in FFHQ6K to the nearest power of two, and train the corresponding network layers only on the appropriate subset of data. To generate at any resolution below 1024, we take the nearest model output that is larger than the target resolution, and apply Lanczos downsampling. Similar to our approach, we start with a pretrained model at 256 resolution, and initialize both the generator and discriminator with pretrained weights. Because each increase in output resolution involves training additional weights, and the number of images at a given resolution decreases as resolution increases, we find that this training approach yields visual artifacts at higher resolutions, shown in Fig. C-5.

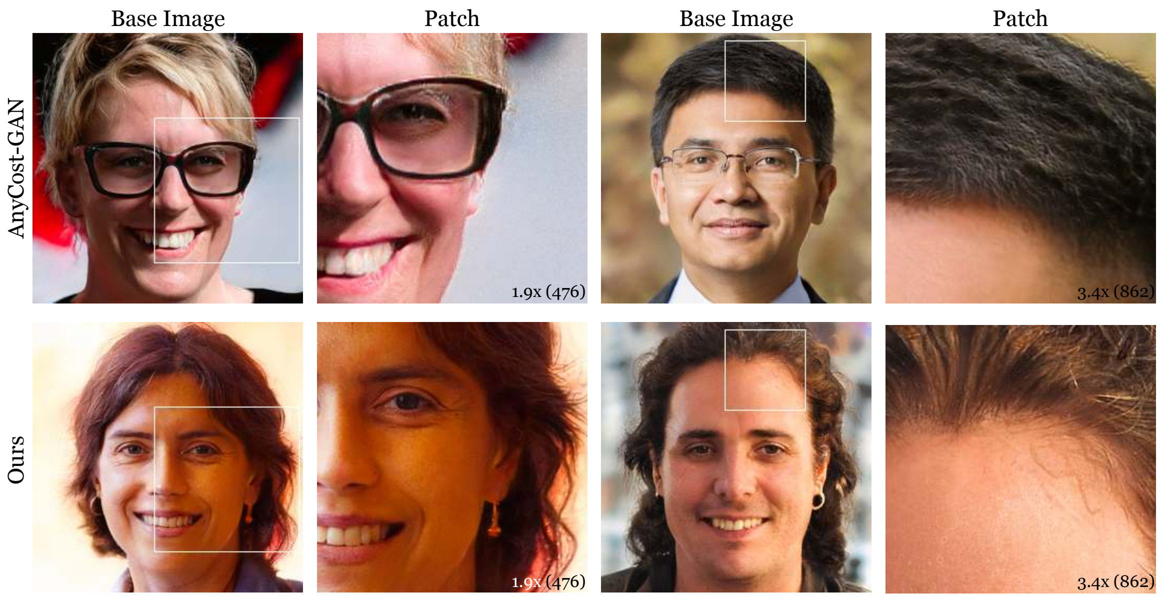


Figure C-5: **Comparison to Anycost-GAN.** Using the same FFHQ6K dataset, we train an Anycost-GAN [136] and compare it to our model. While Anycost-GAN adds additional modules to increase synthesis resolution, our model shares weights across resolutions. Note that the output from Anycost-GAN contain more visual artifacts, particularly in finely textured regions such as hair.

C.2.5 Comparison to oracle generator

In Section 4.4 of the paper, we describe our experimental setup on the face domain, and in Table 4.5, we show competitive performance training on few HR images, even compared training on the whole HR dataset. We provide additional details and visualizations here.

We use the FFHQ dataset as a collection of 70k high-resolution ground-truth images. To simulate more "in-the-wild" settings, we use a fraction (6k) of HR images for patch-based training with a generator of size $p = 256$, where only 1k of the images

Table C.3: **Oracle comparisons.** Comparison of our patch generator (6k images, varied sizes) to oracle generators which train on the entire FFHQ dataset (70k images, 1024 resolution). Although the oracle generators attain better FID, our method enables synthesis at continuous resolutions and can train without assuming that all images are resized to a common resolution. We also include pFID at the maximum 1024 scale.

	FID			pFID	
	256	512	1024	random	1024
SGAN2 Oracle	3.05	2.81	2.69	2.26	4.83
SGAN3 Oracle	3.54	3.23	3.06	2.44	4.29
Patch (Ours)	3.34	3.71	4.06	2.96	8.01

are the full 1024 resolution, and the remainder are uniformly downsampled between 512 and 1024 prior to training. As a comparison, we also evaluate two oracle models that train a generator directly for the $s = 1024$ global image, using the entire 70K images in the FFHQ dataset, and Lanczos downsample the result for FID computations at other resolutions. We also evaluate a variant of pFID, by holding the scale *fixed* at the maximal resolution and randomly sampling crop locations (pFID 1024), in addition to our original pFID metric that randomly samples both scale and location (pFID random). Note that this evaluation is only possible for FFHQ controlled setting, as all images are present at the maximal 1024 size.

Despite being trained to generate patches, our generator can approximately match the frequency content in real images, and that of a StyleGAN3 model trained for 1024 resolution generation on the full FFHQ dataset (Fig. C-6). While StyleGAN2 achieves better FID than StyleGAN3, we find that it has a different frequency profile that is less similar to that of real images. We compare the FID of these oracle models with our continuous patch model in Tab. C.3. While the oracles can achieve lower FID and pFID variants, we note that training the oracle assumes that a sufficient number of high-resolution images of the same size are available, and trains the model specifically for a fixed resolution, whereas we employ mixed resolution training on fewer than 10% of the full HR dataset. Our training strategy therefore allows us to take advantage of the varied resolutions of images in the wild, which is not possible in the oracle setting.

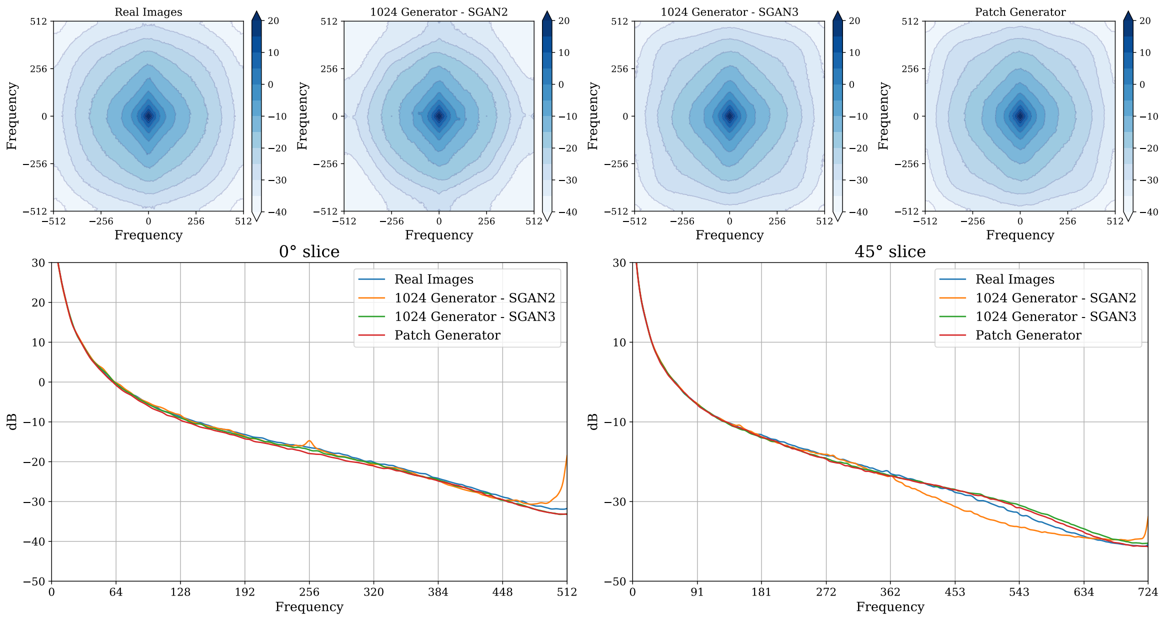


Figure C-6: **Comparison of frequency distribution.** We plot the frequency spectrum of real images, StyleGAN2 and StyleGAN3 trained on the entire FFHQ dataset at 1024 resolution, and our Patch Generator which is trained on $p \times p$ patches of FFHQ6K (which contains approximately 1k images at 1024 resolution and 5k at lower resolutions). The frequency distributions are similar, suggesting that even a smaller generator is able to approximate fine textures well.

C.2.6 Additional model variations

In Section 4.4.2 of the main text, we describe and study variations of our model. Here, we provide additional quantitative and qualitative results and study additional factors.

Alternative metrics. In addition to FID and pFID, we also report precision and recall [196] for our model, naive upsampling, and super-resolution models (Tab. C.4). Super-resolution obtains lower precision (suggesting out-of-distribution results) and similar or lower recall. Upsampling obtains similar or better precision, but lower recall (suggesting that is does not sufficiently cover the real image distribution). Here, we also include pFID measured at the maximum resolution for FFHQ (1024).

Variations on teacher regularization. In the main text, we introduce variations on the teacher regularization including "forward" and "inverse" loss formulations, and discarding the teacher regularization all-together. Tab. C.5 shows the FID comparisons of these three variants, in which the "inverse" loss obtains the best FID scores at

Table C.4: **Precision and recall.** We evaluate additional precision and recall metrics [196], and their corresponding patch variants, for our model, naive upsampling, and super-resolution methods on the FFHQ dataset. We also include pFID at a fixed 1024 scale evaluated in Tab. C.3.

	Precision		Recall		pFID
	1024	Patch	1024	Patch	1024
Upsample	0.68	0.77	0.37	0.20	31.70
Real-ESRGAN	0.40	0.54	0.51	0.47	20.04
GFP-GAN	0.48	0.62	0.34	0.32	20.58
Ours	0.69	0.68	0.47	0.55	8.01

Table C.5: **Variations of teacher regularizer on FFHQ6k.** The inverse teacher regularization outperforms forward regularization, and adding a scale-conditioning branch further improves higher resolutions. Omitting the teacher harms global structure. (*) indicates default setting.

	FID			pFID
	256	512	1024	random
Forward teacher	3.23	4.21	5.35	6.06
Inverse teacher	3.35	4.18	4.88	4.67
No teacher	5.50	5.93	7.13	3.17
Inverse + scale (*)	3.37	4.41	4.47	4.28

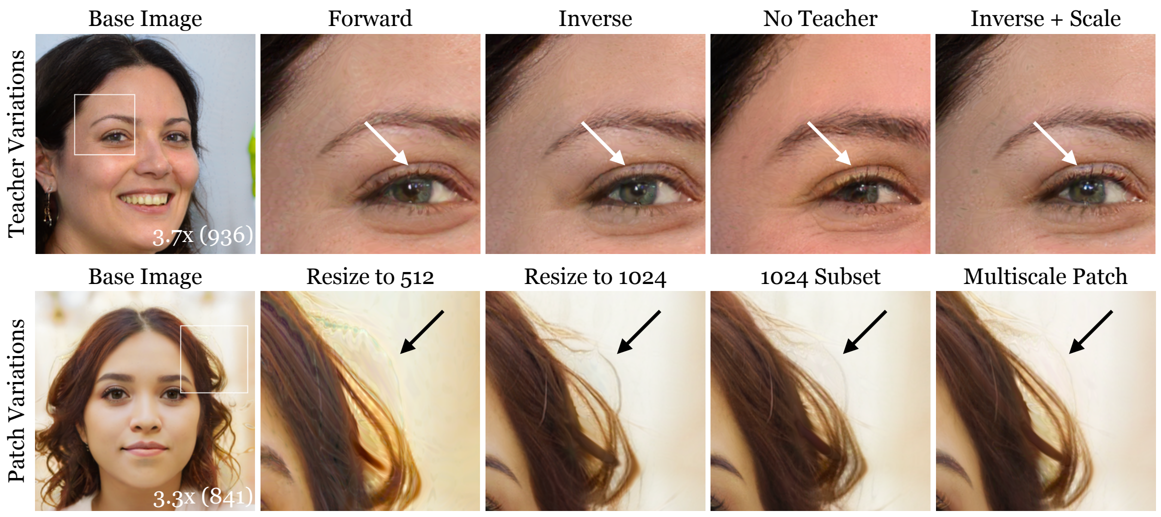


Figure C-7: **Qualitative examples of model variations.** (Top) Using an inverse teacher loss and the scale conditioning branch generates sharper details while also preserving similarity to the base image. (Bottom) We compare our multi-size training approach to methods that do not take advantage of different image sizes and instead train for a fixed resolution. Fixed-resolution training cannot generalize to other resolutions, and upsampling images leads to blurring. Our final model is able learn from mixed-resolution training images and also synthesize at arbitrary resolutions.

the highest 1024 resolution. Adding the scale-conditioning branch to inject scale information throughout the generator further improves FID@1024 and pFID. We show qualitative examples in Fig. C-7 (top), where the inverse teacher with scale-conditioning input can synthesize the cleanest details while still being similar to the base image.

As default, we set $\lambda_{\text{teacher}} = 5$ during the patch-based training phase. Changing

Table C.6: **Teacher regularization.** Teacher regularization weight trades off between improved detail synthesis (pFID) and global realism (full image FIDs). We choose an in-between value ($\lambda_{\text{teacher}} = 5$); this value can be adjusted based on desired similarity to the base resolution. (*) indicates our default setting.

	FID			pFID	L1
	256	512	1024	random	
$\lambda_{\text{teacher}} = 0$	5.50	5.93	7.13	3.17	0.16
$\lambda_{\text{teacher}} = 2$	3.42	4.58	5.46	3.15	0.10
$\lambda_{\text{teacher}} = 5$ (*)	3.37	4.41	4.47	4.28	0.08
$\lambda_{\text{teacher}} = 10$	3.46	4.25	4.61	5.39	0.07

Table C.7: **Patch sampling.** We sample patches from the HR dataset at global resolutions between ($s_{\min}, s_{\max}$). The same model architecture trained on patches from higher resolution images improves the synthesis result at 1024 resolution. (*) indicates our default setting.

	FID			pFID
	256	512	1024	random
(256, 512)	5.20	5.92	19.01	35.66
(256, 1024)	3.43	4.16	4.61	4.19
(512, 1024) (*)	3.28	4.04	4.16	3.61

λ_{teacher} balances between local image quality and similarity to the base resolution image, where higher λ_{teacher} offers the most similarity to the base resolution with lower L1 difference, but lower λ_{teacher} improves pFID, suggesting better quality of the synthesized patches (Tab. C.6).

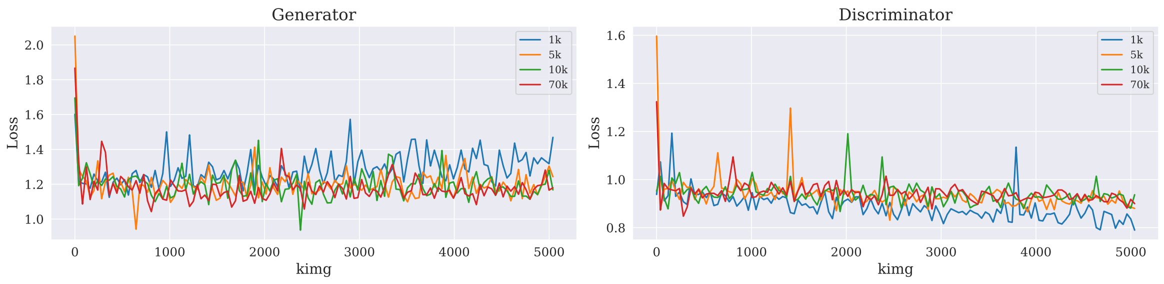


Figure C-8: **Number of training images.** While FID numbers are similar, we find that using 1K HR images for training shows some evidence of divergence. The training dynamics of 5K images is similar to that of the full dataset.

Fixed-size vs Multi-size training. Fig C-7 (bottom) shows an example of a synthesized patch comparing our multi-size training to strategies of fixed-size training. Fixed-size training does not naturally generalize to other sizes, causing deterioration in image quality when sampled at resolutions not equal to the training resolution. Upsampling the training images to a common resolution introduces blurriness in the synthesized output. The result of training on only the subset of images at 1024 resolution looks qualitatively similar to that of multi-scale training, but multi-scale training attains better FID metrics and is able to use more images for training.

Changing the number of training images. While the model FID scores remain largely similar (within a range of 0.3) when training on 1k to 70k high-resolution images, we found that using 1k images showed some evidence of training divergence (Fig. C-8). On the other hand, the training trajectory of using 5k images looks largely similar to that of using the full HR dataset (70k) images. Therefore, when collecting images for the remaining domains, we aim to collect between 5k-10k images to construct the HR dataset.

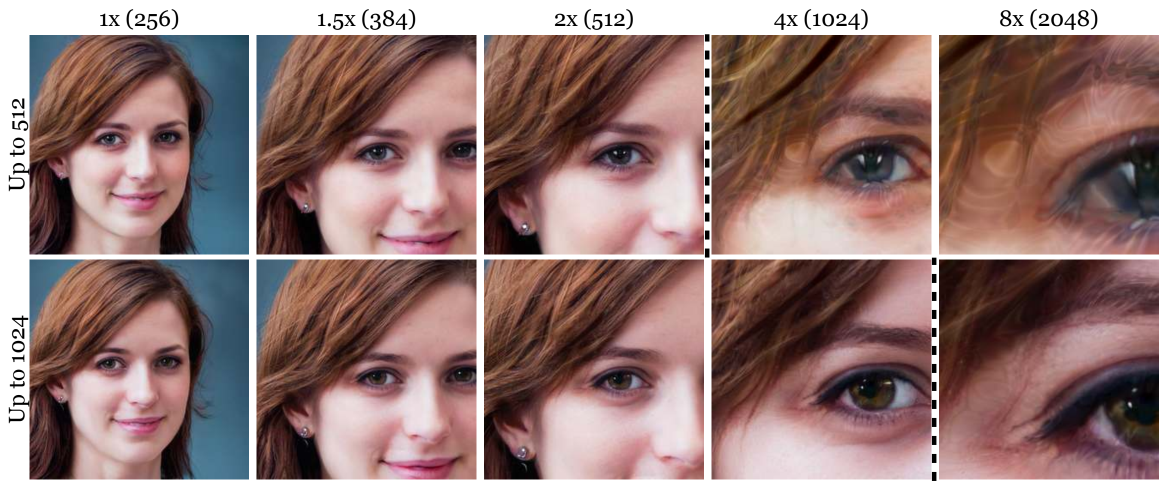


Figure C-9: **Impact of sampling resolutions.** Using the same model architecture and FFHQ6k dataset, we sample images (top) from 256 to 512 resolution, and (bottom) from 512 to 1024 resolution. The dotted line indicates when inference resolution exceeds the maximum training resolution. Watery artifacts start to appear when extrapolation, but this can be tempered by simply training on patches from larger images.

Investigating the impact of sampling resolutions. Our FFHQ6k dataset contains images between 512 and 1024 resolution, and during training the images are randomly downsampled from their native resolution, and can be optionally clipped at an upper resolution. Here, we conduct experiments to study the effects of these sampling ranges. When training the model on resolutions s sampled between 256 and 512, the image quality declines by 1024 resolution at inference time and contains visual artifacts (Tab. C.7, Fig. C-9). Taking the same image and model architecture, but instead training on resolutions between 256 and 1024 offers better FID@1024, and sampling from 512 to 1024 resolution further improves FID@1024. As before, all models are trained on patches of size $p = 256$, and the model is jointly trained on the fixed-size dataset to preserve FID@256. Accordingly for the other domains, our sampled resolutions for HR dataset range between the native resolution s_{im} and the minimum resolution of the HR images. These results suggest that the synthesized resolution can be dictated by the training images; simply adding patches from higher resolution images can allow the same model to better synthesize at a higher resolution. We also tried an experiment using a separately trained smaller 128px patch generator and the same 512 to 1024 resolution patch sampling scheme, but obtained worse FID (8.38 at 1024 resolution compared to 4.16 for our default model); we hypothesize this is because may be due to worse FID from the initial pretraining phase that carries over to the patch training phase (5.14 compared to 3.71 for our default model).

Changing the discriminator. Our final model introduces changes to the generator, but keeps the same discriminator from the initial pretraining step. During patch-training, the discriminator must also learn to distinguish between real and synthesized patches. Here, we investigate alternatives of changing the discriminator setup (Tab. C.8). (1) We remove sampling from the LR dataset, now causing the discriminator to focus entirely on patches. This causes pFID to improve but the remaining global FIDs to worsen. In particular, this allows the generator to forget how to synthesize at the base resolution, causing a large increase in FID@256. The impact of sampling from the base resolution and the teacher regularization have similar outcomes: both encourage global coherence, but the teacher has a stronger effect than base resolution training. (2) We also try adding a second discriminator so that one focuses entirely on the global low-resolution image, and the other entirely on patches. Both discriminators are initialized with the result from pretraining, but we find that this setting leads to suboptimal metrics, compared to using a single discriminator. (3) We inject scale information into the discriminator following a similar method as the generator via weight modulation. In this case, the training becomes unstable as the discriminator is able to out-compete the generator.

Table C.8: **Discriminator variations.** Our default discriminator, which jointly trains globally on the LR dataset and patches from the HR dataset attains the best FID metrics. Other changes to the discriminator did not improve performance. (*) indicates default.

	FID			pFID
	256	512	1024	random
Default Discriminator (*)	3.28	4.04	4.16	3.61
No Base Resolution	9.96	4.23	4.69	2.92
Two Discriminators	3.82	4.63	5.34	3.38
Scale-conditioned Discriminator	31.81	71.33	89.38	120.06

Removing the pretraining step. Our final model is first pretrained at a fixed, smaller resolution before varied-size patch training is enabled. This pretraining phase encourages global coherence and also serves as the teacher model later during patch-based training. We conduct an experiment in which the initial pretraining step is omitted, and the model is trained on randomly sampled patches from the start of training. When trained with the same number of HR image patches, the model without global pretraining suffers in both structure and texture – FID for 1024px generated images is 26.78 and pFID is 8.64 – compared to our original model which performs global pretraining at low resolution – FID at 1024px is 4.50 and pFID is 3.46 after 10M training images.

C.2.7 Detectability

A concern with improved image generation is the potential for more convincing deceiving images, particularly those of higher resolution, which is the focus of our work. We use the off-the-shelf detector from Wang et al. [251] on our Birds (generated 256 $\rightarrow$ 2048) and Mountains (1024 $\rightarrow$ 4096) generators, across a large range of resolutions. As shown in Figure C-10, the scores are well above chance (50%) across both datasets and resolutions. Interestingly, the curve generally trends upwards, indicating that while higher resolution images may look more natural, they are also easier to detect.

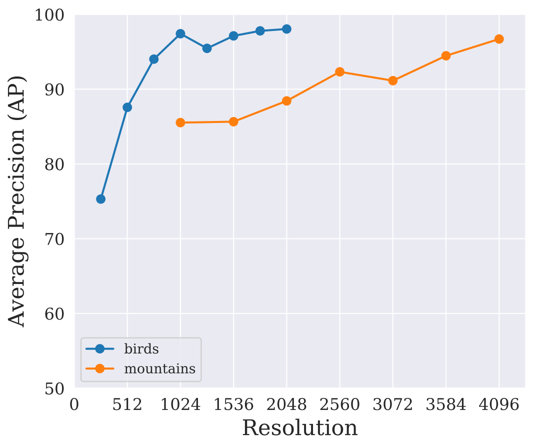


Figure C-10: **Detection score.** We run the model from Wang et. al [251] on our Birds and Mountains datasets. All scores are above chance of 50%. Note in both cases, the detectability of our network trends upwards with resolution.

C.3 Additional Implementation Details

Building off the StyleGAN3 [111] architecture, we describe our coordinate conditioning and scale modulation branch applied to enable generation of multi-scale patches during the second training phase.

C.3.1 Patch-based training

Extracting patches from varied size images From our dataset of images $\mathcal{D}$, we sample an image $x_i \in \mathbb{R}^{H_i \times W_i \times 3} \sim \mathcal{D}$, with short-side $s_{\text{im}} = \min(H_i, W_i)$, and take an s_{im}-by-s_{im} square crop. We then Lanczos downsample the image to an intermediate resolution $s \in [p, s_{\text{im}}]$, which provides "free" additional views from the same image, without introducing image corruptions.

Next, we sample a random crop of size p and record the sampling location $v \in \mathbb{R}^2$. To summarize this procedure, we obtain a patch $x \in \mathbb{R}^{p \times p \times 3}$ from these two operations, while saving the sampled image resolution s and patch center location $\mathbf{v} = (\mathbf{v}_x, \mathbf{v}_y) \in [0,1]^2$ for later use.

$$x, s, \mathbf{v} = \text{Crop}(\text{Downsample}(x_i)) \tag{C.1}$$

Synthesizing patches from the generator. Given a set of patches from the image dataset, the generator is tasked with synthesizing images at the corresponding patch locations. To transform the normalized coordinate domain $[0,1] \times [0,1]$ into patch coordinates, we apply a transformation matrix to each 2D location c in homogenous coordinates:

$$c_{\mathbf{v},s} = T_{\text{patch}} * c = \begin{bmatrix} \frac{p}{s} & 0 & \mathbf{v}_x \\ 0 & \frac{p}{s} & \mathbf{v}_y \\ 0 & 0 & 1 \end{bmatrix} * c \tag{C.2}$$

Following StyleGAN3, these transformed coordinates are then encoded as K random Fourier channels by multiplying by frequencies $B \in \mathbb{R}^{K \times 2}$ and adding phases $\phi \in \mathbb{R}^K$. For patch synthesis, the Fourier feature extraction at index (h, w) becomes:

$$F_{h,w}(c_{\mathbf{v},s}) = \sin\left(2\pi B c_{\mathbf{v},s} + \phi\right) \in \mathbb{R}^K, \tag{C.3}$$

C.3.2 Scale-conditioning branch

As individual coordinate positions $c_{\mathbf{v},s}$ do not directly convey scale information, we found it beneficial additionally incorporate the scale input to intermediate layers of the generator. To do this, we first normalize the target scale s to $[0, 1]$ using:

$$\bar{s} = \frac{s - p}{s_{\max} - p} \tag{C.4}$$

where $s_{\max}$ is selected from dataset statistics and is only present as a normalization factor, but does not clip the upper synthesis bound during inference. Empirically, we found that adding a small offset factor (we use 0.1) to the normalized target scale $\bar{s}$ allows for smoother interpolations between resolutions by avoiding a discontinuity at zero.

We then encode $\bar{s}$ using a parallel mapping network of identical architecture to the latent mapping network $M(z)$, and add the two inputs after undergoing a layer-specific affine transformation into style-space [261, 175] to obtain the final modulation parameter $M(z, s)_k$ at layer k:

$$M(z, s)_k = (W_{z,k} * M_z(z) + b_{z,k}) + (W_{s,k} * M_s(s) + b_{s,k}) \tag{C.5}$$

Because the modulation parameter is a multiplicative factor on the network weights and the scale-conditioning portion is added only during the secondary patch-wise training step, we initialize $b_{z,k} = \mathbf{1}$ and $b_{s,k} = \mathbf{0}$ to allow the network to smoothly transition between the initial pretraining step and secondary patch-based training.

C.3.3 Training procedure

We train our models on four to eight V100 GPUs with 16GB memory. By sampling fixed-size patches, the memory and compute footprint remain constant during training. For FFHQ, we finetune our initial fixed-scale generator from the pretrained FFHQ-U model [111], which reaches a minimum FID within 4M training images. In the remaining domains, we perform the pretraining step from scratch, retaining the checkpoint with the lowest FID, computed over 25M image samples, before continuing with the second, mixed-resolution training phase. Our training procedure is compatible with both 3×3 and 1×1 kernel sizes in StyleGAN3 (T & R configurations, respectively). For the patch-based training step, we proceed with the model configuration that reaches the best FID in pretraining, which is typically Config T with the exception of the FFHQ domain.

Appendix D

Supplementary: Persistent Nature

In supplemental materials, we investigate an alternative 3D feature representation based on extendable triplane units D.1. We provide additional implementation details of our method in Section D.2, additional ablations in Section D.3, and we provide further discussion on our model in Section D.4.

D.1 Extended Triplane Variation

Instead of decoding the scene from a 2D layout feature grid and height of a 3D point above this layout plane, we also experiment with a model variation that adds vertical support planes parallel to the XY and YZ planes. Thus, the layout features are described by a 2D extended XZ layout feature grid, and sets of orthogonal support planes shown in pink in Fig. D-1-left. Decoding a given 3D point projects the point to the XZ plane, the four nearest vertical planes (two parallel to XY and two parallel to YZ, which are weighted linearly according to the distance of the point from each plane).

Qualitatively, the triplane model achieves more geometry diversity, with more mountainous terrain compared to the feature layout model. We attribute this to the additional support provided from the vertical feature planes. Additionally, the vertical feature planes allow for a lighter decoding network with higher neural rendering resolution, allowing for faster video rendering and improved temporal consistency (lower one-step consistency error) due to less reliance on a 2D upsampling operation. We show qualitative examples in Fig. D-2 with video results on our project page, and quantitative evaluations in Tab. D.1. Quantitatively, while this extended triplane variation does not output perform the layout model in terms of FID, we hypothesize that the FID may be impacted by two possible factors: first, this model requires inference-time camera height adjustment to avoid intersecting with increased complexity of the generated geometry, and second, interpolation between vertical feature planes qualitatively produces more muted colors compared to the real image distribution.

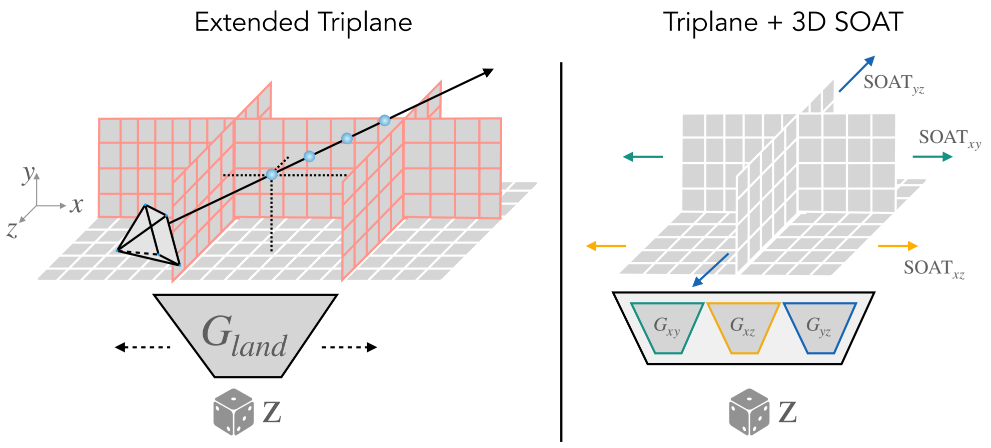


Figure D-1: **Diagram of extended triplane representation.** The extended triplane representation adds a sequence of orthogonal vertical feature planes outlined in pink in addition to the ground plane features outlined in white **(left)**. Each unit consists of a triplane representation [27] generated from three independent generators – G_{XY}, G_{XZ}, and G_{YZ} – tied to the same latent code and mapping network **(right)**. At inference time, the features of each generator are stitched along the appropriate dimensions using the SOAT procedure [40].

We also investigate the impact of using a projected 3D noise pattern as input into the extended triplane upsampler, with results in Tab. D.2. While this improves FID and consistency in the layout representation, we find that the benefits of the projected noise are more limited in the extended triplane setting. Adding projected noise offers improvements in FID, but also a small increase in consistency error. Qualitatively, the model outputs are similar with and without the projected noise, perhaps attributed to decreased reliance on the upsampling operation.

Model	FID			Consistency	Render Time (s)
	C_{train}	C_{forward}	C_{random}		
Extended Layout	**21.42**	**26.67**	**23.39**	3.56	8.49
Extended Triplane	24.47	34.89	34.76	**2.29**	**0.16**

Table D.1: **Extended Layout vs. Extended Triplane.** While the extended layout representation presented in the main paper attains better image quality (lower FID scores), the extended triplane representation offers improved consistency (lower one-step consistency error) and dramatically faster video rendering (as the layout model requires supersampling for video smoothness). We hypothesize that inference-time camera adjustments and interpolation between vertical feature planes may negatively impact FID for the extended triplane model, despite its ability to generate more complex and diverse landscape geometry.

Model	FID			Consistency
	C_{train}	C_{forward}	C_{random}	
Without Noise	**24.47**	34.89	34.76	**2.29**
With 3D noise	25.31	**33.30**	**33.28**	3.06

Table D.2: **Effect of 3D Projected Noise.** Adding projected noise into the upsampler of the extendable triplane representation offers improvements in FID but is slightly more inconsistent, but still more consistent than the layout model.

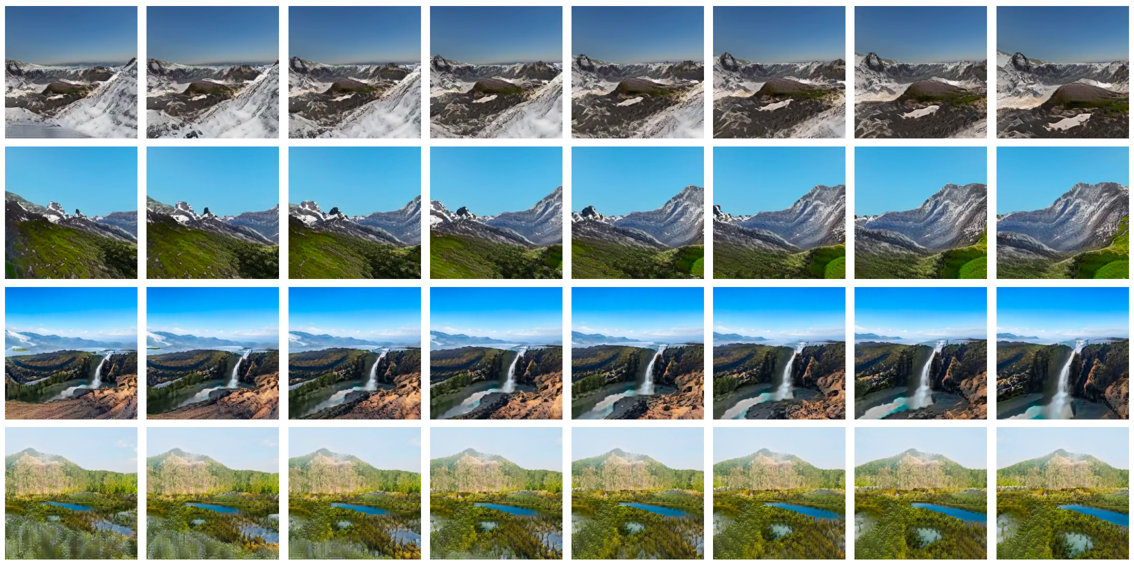


Figure D-2: **Extendable triplane visualization.** Qualitative examples of rendering from the extendable triplane representation. This representation results in larger scene and geometry diversity compared to the layout feature representation, with improved 3D consistency.

D.2 Additional Methodological Details

D.2.1 Preprocessing

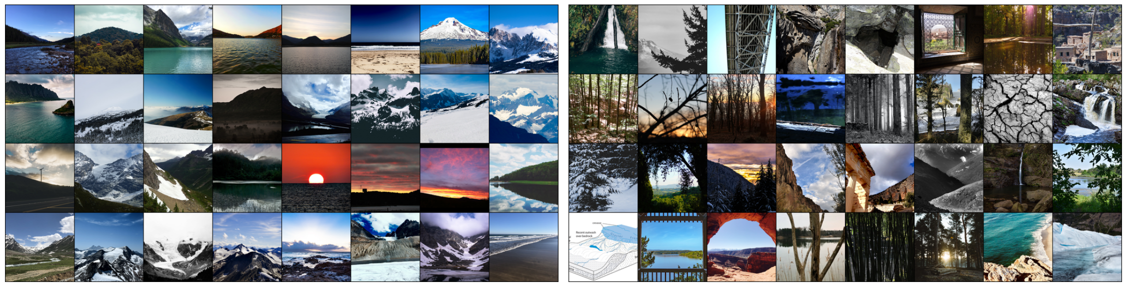

(a) Training Images (b) Removed Images

Figure D-3: **Result of dataset filtering.** The dataset filtering step (a) retains images that contain sufficient sky pixels near the top of the image, and (b) removes images that are not typical images of landscapes. These atypical images include images without sky pixels, or images with nearby occluding objects such as windows or trees. The filtering criteria is based on sky segmentation and disparity estimation obtained from DPT [184].

Dataset Filtering. To remove images in the LHQ [221] dataset that contain occluding objects close to the camera, we apply filtering criteria to construct the training dataset. Using the segmentation output of DPT [184], we detect the sky region and boundaries of the resulting binary sky mask. As the segmentation results can include small regions with inconsistent labels (*e.g.* small holes in the sky), we remove all bounded regions with area under 250 pixels to create a more unified sky mask. Next, using this segmentation mask we filter out images for which any of the following hold: (1) there are more than three bounded sky regions, (2) more than 90% of the scene is not sky pixels, (3) more than 40% of the upper one-fifth of the image is not sky pixels, and (4) less than 80% of the lower quarter of the image is not sky pixels. The first three criteria are meant to filter out images that contain occluding structures (such as trees or windows) or images in which there is no sky region present. The fourth criteria is meant to filter out images taken from unusual camera angles (such as from underneath a bridge). Using the monocular depth prediction from DPT, we also remove images containing too many vertical edges: images are removed if the 99th percentile of the pixel-wise finite difference is greater than 0.05, which tends to be indicative of trees or man-made buildings. Fig. D-3 shows examples of images that were retained for training, and those that were filtered out.

Disparity Normalization. Using the monocular depth prediction from DPT, we normalize the disparity values between 0 and 1 using the 1st and 99th percentile values per image. Next, we clip the minimum disparity for non-sky regions and rescale the disparity values to correspond to the near and far bounds used in volumetric rendering (see § D.2.2). We use 0.05 for our clip value and $1/16$ for the scale factor; this means that after normalization, the disparity values for non-sky pixels range from $1/16$ to 1. The disparity for the sky pixels is clamped at zero.

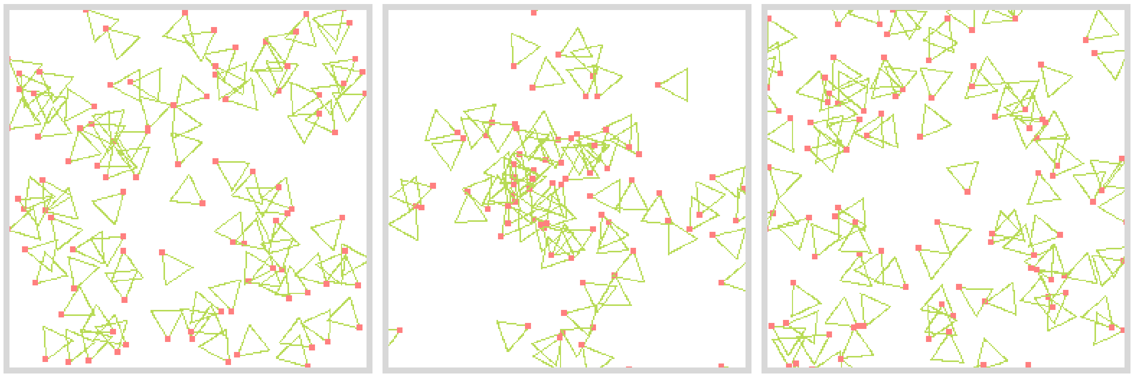

(a) Cameras for training (b) Cameras after forward motion (c) Cameras randomly oriented

Figure D-4: **Illustration of camera distributions.** (a) Cameras used for training are sampled with a random translation uniformly over the scene layout feature grid, with rotation sampled to overlap with this feature grid. To evaluate view extrapolation, we (b) move the cameras forward a distance equivalent to 100 steps of InfiniteNature-Zero [131], corresponding to roughly halfway across the scene layout grid, or (c) randomly sample a random translation and random rotation.

Camera Poses. We sample training camera poses with a random (x, z) position within the layout grid, and a rotation such that the near half of the view frustrum lies entirely within the training grid. To simulate the forward motion of InfiniteNature-Zero [131], we move the camera forward a distance equivalent to 100 steps of InfiniteNature-Zero, corresponding to roughly half of the scene layout grid. To evaluate view extrapolation, we randomize the position and rotation of the cameras at inference time. These settings are illustrated in Fig. D-4.

D.2.2 Training and implementation

Training objective

Each stage of our model is trained following the StyleGAN2 objective [110], with a non-saturating GAN loss V and R_1 regularization [153]:

$$\begin{aligned} &V(D, G(z), I) = D(I) - D(G(z)), \\ &R_1(D, I) = ||\nabla D(x)||^2, \\ &G = \arg\min_G \max_D \mathbb{E}_{z, I\sim\mathcal{D}}\, V(D, G(z), I) + \\ &\qquad \frac{\lambda_{R_1}}{2} R_1(D, I), \end{aligned} \tag{D.1}$$

where G, D refer to the corresponding generator and discriminator networks at each training stage, and x refers to real images sampled from dataset $\mathcal{D}$. Additional auxiliary losses for each part of the model are described in the following sections.

Layout Generator

Our layout generator is based on the architecture from GSN [47], which is comprised of two components: G_{land}, which synthesizes the scene layout grid, and M which decodes the 2D layout feature into a 3D feature.

The layout generator G_{land} follows StyleGAN2 [110], which generates a 256×256 grid of features $f_{\text{land}} \in \mathbb{R}^{32}$. G_{land} contains three mapping layers and the maximum channel dimension is capped at 256; all other parameters are unchanged from StyleGAN2.

The network M is modeled after the style-modulated MLP from CIPS [5], containing eight layers with a hidden channel dimension of 256 and producing features $f_{\text{color}} \in \mathbb{R}^{128}$. The constant input to M is replaced with the y-coordinate (height above the ground plane), and the modulation input is the interpolated feature from f_{land}.

We adapt the rendering procedure of GSN to handle unbounded outdoor scenes. For volumetric rendering, we set the near bound to 1 and the far bound to 16, which corresponds to the scale factor used in disparity normalization during data preprocessing. Each scene layout feature has a unit width of 0.15, such that the full width of the feature grid is $256 \times 0.15 = 38.4$, which is slightly over twice the far bound distance. We omit positional encoding from M, as we found that including positional encoding yielded grid-aligned artifacts in generated images; we also omit the view direction input. Camera rays are sampled using FOV $= 60^\circ$ with linearly spaced sampling between the near bound and the far bound. We use inverse-depth (disparity) supervision rather than depth supervision so that we can represent content at infinite distances. This also encourages the terrain generator to create empty space in the sky content, which will be filled with the skydome generator.

We use the volumetric rendering equations from NeRF [156], in which the weights w_i of the i-th point along a ray depends on densities σ which is predicted by multi-layer perceptron M and the distance between samples δ:

$$\alpha_i = 1 - \exp\left(-\sigma_i \delta_i\right), \quad w_i = \alpha_i \exp\Big(-\sum_{j=1}^{i-1} \sigma_j \delta_j\Big). \tag{D.2}$$

Our training procedure for the layout decoder follows that of GSN [47], which provides the real RGB image I_{RGB} and disparity d (obtained from DPT) to the discriminator $I = \{I_{\text{RGB}}, d\}$, and also adds a reconstruction loss on real images using a decoder network $G\phi$ on discriminator features $D\phi$:

$$\mathcal{L}_{\text{rec}} = (I - G\phi(D\phi(I)))^2. \tag{D.3}$$

The full GAN objective follows Eqn. D.1 with weights $\lambda_{R_1} = 0.01$ and $\lambda_{\text{rec}} = 1000$, and we follow the optimizer settings from StyleGAN2 and train for 12M image samples.

Because the layout decoder tends to generate semi-transparent geometry, which also causes unrealistic sky masks, we regularize the geometry following Eqn. 4 in the main document, and add the sky mask into the discriminator. We finetune with this additional loss for 400k samples with $\lambda_{\text{transparent}}$ which linearly increases from zero to $\lambda_{\text{transparent}} = 80$ over the finetuning procedure.

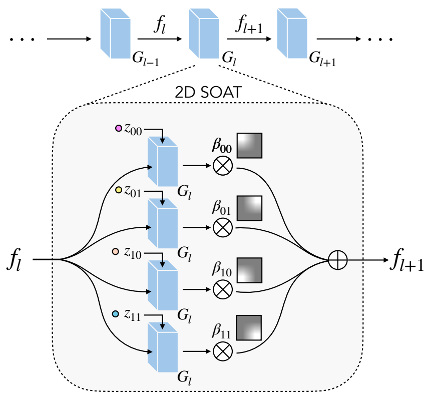


Figure D-5: **Layout extension.** We adapt the procedure in SOAT [40] for 2D layout extension. Operating on each layer of the generator, we take the incoming feature grid f_l, and construct the outgoing feature grid using the generator weights conditioned on each corner latent code z (the conditioning uses weight modulation on the mapping network outputs in StyleGAN2 [110]). Then, these four outgoing feature maps are multiplied with bilinear weights β and the result is summed, to obtain the blended feature for the next layer f_{l+1}.

Layout extension

We use the procedure of SOAT [40] in two dimensions to smoothly transition between adjacent feature grids sampled from independent latent codes. SOAT proceeds by operating on 2x2 sub-grids and stitching each layer of intermediate features in the generator (Fig. D-5). To start, we simply concatenate the StyleGAN constant tensors, to obtain a feature grid f_0 of size $2H_0 \times 2W_0$, where H_0 and W_0 are the height and width of the constant tensor. For each subsequent layer f_{l+1}, we modulate the weights G_l with each of four corner latent codes (after applying the mapping network to obtain the style-code) and apply it in a fully convolutional manner to f_l, obtaining $f_{k,l+1}$ of size $2H_l \times 2W_l$. Then, we multiply each of $f_{k,l+1}$ with bilinear interpolation weights β and take the sum to obtain f_{l+1}. This procedure is repeated for each layer of the generator, obtaining a an output feature grid of size $2H \times 2W$. To reduce the effect of padding, these output feature grids are tiled in an overlapping manner, with a 50% overlap on each side and with weights that linearly decay to zero away from the center of the tile.

Refinement Network

The refinement network G_{up} uses a truncated StyleGAN2 backbone, which replaces the feature input of the 32×32 block with the 32×32 rendered feature f_{im} and initial image I_{LR}, depth d_{LR}, and sky mask m_{LR}. The skip connection of the upsampler takes in I_{LR}, d_{LR}, m_{LR} and predicts I_{HR}, d_{HR}, and m_{HR}. Following the noise injection operation in StyleGAN2, we replace the image-space 2D noise tensor with our 3D-consistent projected noise (Eqn. 7 in the main document). This network uses two mapping layers, taking as input the style latent vector from G_{land}.

We add an additional objective to encourage consistency between the refined color pixels and the sky mask:

$$\begin{aligned}\mathcal{L}_{\text{consistency}} &= |d_{\text{HR}} - d_{\text{LR}\uparrow}| + |m_{\text{HR}} - m_{\text{LR}\uparrow}|, \\ \mathcal{L}_{\text{sky}} &= \exp(-20 * \sum_c |I_{\text{HR}}[c]|) * m_{\text{HR}}.\end{aligned} \tag{D.4}$$

The loss $\mathcal{L}_{\text{consistency}}$ encourages the high resolution depth and mask outputs to match their upsampled low resolution counterparts (this results in a smoother outcome compared to downsampling the high resolution outputs). The loss $\mathcal{L}_{\text{sky}}$ encourages pixel colors to be nonzero (reserved for the gray sky color) when $m_{\text{HR}} = 1$, by summing over the three channels c of the predicted image I_{HR}; this is meant to encourage the RGB colors produced refinement network to be consistent with the mask and depth outputs. The refinement network is trained with the GAN objective (Eqn. D.1) with weights $\lambda_{R1} = 4$, $\lambda_{\text{consistency}} = 5$, and $\lambda_{\text{sky}} = 100$, and the discriminator loss is applied only on the RGB images.

Due to the computational costs of volume rendering, we train the refinement network on 32×32 inputs to produce 256×256 outputs. For 30 fps video visualizations, we supersample the camera rays at 8x spatial density and apply depth-based filtering to the noise input to improve video smoothness; however all metrics in the paper are computed without supersampling for additional smoothness.

We note that while StyleGAN3 [111] is intended to resolve the texture sticking effect caused by the noise input in StyleGAN2, replacing G_{up} with a StyleGAN3 backbone resulted in worse image quality in our setting with FID 67.90, compared to FID 21.42 for our final model.

Skydome Generator

The skydome generator takes as input the CLIP [183] embedding of a single terrain image, and predicts a sky output that is consistent with the terrain. The generator architecture follows StyleGAN3 [111] adapted with cylindrical coordinates to generate 360° panoramas [25].

For the terrain input, we take the filtered LHQ dataset and select the non-sky pixels with normalized disparity greater than 1/16 (this leaves some background mountains to be predicted). We follow the training procedure from [25] with a few adaptations. In addition to concatenating the CLIP embedding of the terrain image to the style-code, the generated sky is composited with the terrain input prior to the discriminator with 50% probability, which is compared to full RGB images from LHQ. The 50% compositing behavior ensures that the bottom of the generated skydome can still appear realistic (when unmasked), while also matching provided terrain image (when masked). This portion is trained with the $\lambda_{R1} = 2$ in the GAN objective (Eqn. D.1), with randomly sampled cylindrical coordinates and a cross-frame discriminator applied to the boundary of two adjacent frames.

D.2.3 Extendable triplane implementation

To construct the extendable triplane representation, we modify the triplane model from EG3D [27] to generate three planes from independent synthesis networks G_{XY}, G_{XZ}, and G_{YZ}, tied to the same latent code and mapping network. Similar to our layout feature model, we train the terrain generator on sky-segmented images and disparity maps as input into the low-resolution discriminator to help the model learn geometry. The upsampler portion of this model and the training procedure is the same as EG3D, using $\lambda_{R1} = 10$. To prevent the model from rendering the segmented sky color (we use white for the sky color, following the background color of NeRF [156]), we finetune the model penalizing for white pixels when the sky mask is one:

$$\mathcal{L}_{\text{sky}} = \exp(-5 * \sum_{c}(I_{\text{LR}}[c] - 1) * m_{\text{LR}}. \tag{D.5}$$

(a) Accumulated ray density with separate skydome

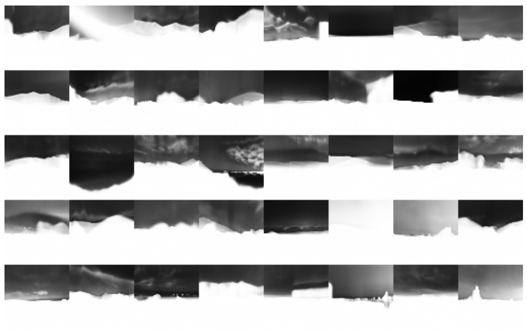

(b) Accumulated ray density without separate skydome

Figure D-6: **Training without a separate skydome.** We supervise the sky content with zero inverse-depth (infinite distance) to ensure that the camera does not intersect the sky as the layout features are extended. As such, we model content at infinite distances with a separate skydome model, such that the terrain model treats sky regions as empty space (left). Without the separated skydome, the model is forced to put sky content at finite distances leading to foggy, semi-transparent content near the camera (right).

The finetuning operation is performed for 400K samples with λ_{sky} increasing from zero to 40 during training. At inference time, we perform SOAT [40] feature stitching to each generator along the appropriate dimensions to obtain the extended triplane representation. As the skydome model does not train on generated images, we use the same skydome model as before. We use 50 randomly sampled camera poses for training, which improves the geometry diversity (more mountainous terrain) the compared to using 1K random training poses.

D.3 Additional Experiments

D.3.1 Training without a separate skydome

Modelling faraway content separately is a common strategy in unbounded scene-reconstruction [9, 76]. To ensure that we cannot intersect the skydome as we arbitrarily extend the layout features, we use zero inverse-depth for sky pixels, which can only render a solid color as the weight of all points along the ray must be zero to obtain zero inverse-depth. In this experiment, we train I_{LR} using the same training strategy as our final I_{LR} model, but instead supervise with full RGB images rather than sky-segmented RGB images. This corresponds to training the model without a separate skydome. We find that without the separate skydome, the model learns incorrect geometry, as it is forced to place some density at finite distances in order to render content in the sky to match the training distribution. Figure D-6 shows the difference in ray accumulations from models trained with the skydome (prior to opacity regularization) and without the skydome. The model without the skydome places semi-transparent content in the sky region, which creates a fog-like effect when moving the camera throughout the landscape.

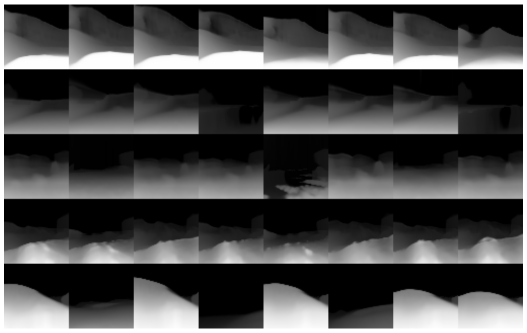
D.3.2 Changing the number of sampled cameras

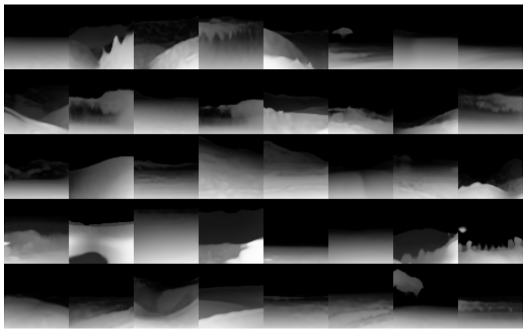
We train our model using a set of one thousand cameras with randomly sampled translations within the layout feature grid, and rotations such that the camera view frustum overlaps with the feature grid. However, one limitation of this training strategy is that we find the model can learn repeating geometry, such that the rendered disparity map may look similar when sampling different random latent codes at the same camera position, despite the pixel color values being different. We hypothesize that the diversity of camera poses sampled during training may obscure the repeating geometry effect from the discriminator, as images sampled from different camera poses will appear different in terms of both color and geometry.

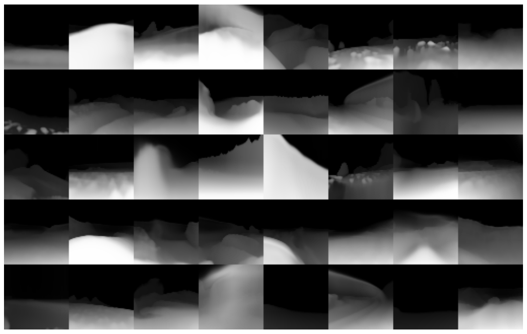
To investigate this effect, we train another model using only five camera poses during training. The disparity maps per camera pose show more diversity in this setting, however we find that this setting results in "holes" and incorrect geometry in the landscape when moving the camera away from the training poses, illustrated in Figure D-7. We use one thousand training cameras as our default setting, but a more optimal setting may involve fewer training cameras, while still ensuring adequate coverage over the feature grid.

D.4 Discussion

A limiting factor of our method is the reliance on a volume rendering operation to decode the 2D layout feature grid into a 3D feature at each sampled point along the ray.


(a) 1K training cameras; training poses


(b) 5 training cameras; training poses


(c) 1K training cameras; independent test poses

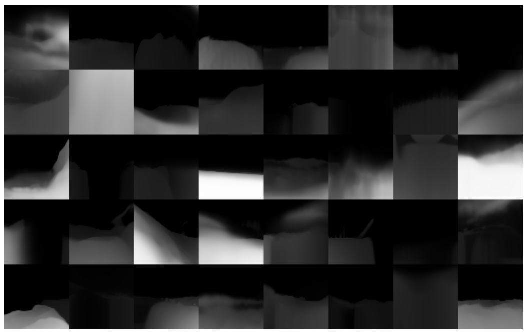

(d) 5 training cameras; independent test poses

Figure D-7: **Adjusting the set of training cameras.** We plot disparity maps corresponding to training with one thousand cameras, and five cameras. (a) With our default setting of one thousand training cameras with camera origins uniformly sampled over the layout feature grid, we find that the model can learn repeating geometry, such that the disparity map generated from the same pose but different latent codes tends to look similar (each row corresponds to the same pose), despite the RGB colors appearing different. (b) With fewer training cameras, the models learns more diversity in the rendered geometry, where again each row corresponds to the same camera pose. (c & d) However, the model trained with one thousand cameras generalizes better to an independent set of cameras, whereas the model trained with five cameras has a greater frequency to put holes in the decoded landscape (evidenced by completely black disparity maps, or disparity maps that have no nearby content and thus are darker overall) or regions of solid content without sky (evidenced by disparity maps that do not fade to black near the top of each image). We use one thousand training cameras as our default setting, but a more optimal setting may involve fewer training cameras, while still ensuring adequate coverage over the feature grid.

Due to this operation, the rendered output I_{LR} can only be trained at low resolution (32x32), and does not learn to generate detailed textures. (In contrast to NeRF-style models which can use per-ray supervision, we must render a complete image as an input for the discriminator.) We rely on a refinement module to upsample the result and add additional textures, but any refinement in image space is prone to losing 3D consistency. Our extended triplane variation reduces the computational expense of volume rendering by reducing the capacity of the decoder MLP and increasing the capacity of the feature representation, thus allowing for neural rendering at 64x64 resolution (we find that geometry degrades at higher resolutions) and decreasing reliance on the upsampler. While we did not find improvements when training on rendered patches, improved patch sampling techniques could help in adding more detail to the rendered result [223].

As our model does not have explicit 3D or aerial supervision, we find that it may generate unnatural or repeating geometry. This can appear as thin mountains, sloping water, or hills of a similar shape but different appearance when sampling from different random noise codes.

Bibliography

[1] Rameen Abdal, Yipeng Qin, and Peter Wonka. Image2stylegan: How to embed images into the stylegan latent space? In *Int. Conf. Comput. Vis.*, 2019.

[2] Rameen Abdal, Yipeng Qin, and Peter Wonka. Image2stylegan++: How to edit the embedded images? In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020.

[3] Amjad Almahairi, Sai Rajeswar, Alessandro Sordoni, Philip Bachman, and Aaron Courville. Augmented cyclegan: Learning many-to-many mappings from unpaired data. In *Int. Conf. Machine Learning*, 2018.

[4] Alexander Amini, Ava P Soleimany, Wilko Schwarting, Sangeeta N Bhatia, and Daniela Rus. Uncovering and mitigating algorithmic bias through learned latent structure. In *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, pages 289–295, 2019.

[5] Ivan Anokhin, Kirill Demochkin, Taras Khakhulin, Gleb Sterkin, Victor Lempitsky, and Denis Korzhenkov. Image generators with conditionally-independent pixel synthesis. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 14278–14287, 2021.

[6] Shekoofeh Azizi, Simon Kornblith, Chitwan Saharia, Mohammad Norouzi, and David J Fleet. Synthetic data from diffusion models improves imagenet classification. *arXiv preprint arXiv:2304.08466*, 2023.

[7] Aharon Azulay and Yair Weiss. Why do deep convolutional networks generalize so poorly to small image transformations? *Journal of Machine Learning Research*, 2020.

[8] Jonathan T Barron, Ben Mildenhall, Matthew Tancik, Peter Hedman, Ricardo Martin-Brualla, and Pratul P Srinivasan. Mip-nerf: A multiscale representation for anti-aliasing neural radiance fields. In *Int. Conf. Comput. Vis.*, pages 5855–5864, 2021.

[9] Jonathan T Barron, Ben Mildenhall, Dor Verbin, Pratul P Srinivasan, and Peter Hedman. Mip-NeRF 360: Unbounded anti-aliased neural radiance fields. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 5470–5479, 2022.

[10] David Bau, Jun-Yan Zhu, Hendrik Strobelt, Bolei Zhou, Joshua B Tenenbaum, William T Freeman, and Antonio Torralba. Gan dissection: Visualizing and understanding generative adversarial networks. In *Int. Conf. Learn. Represent.*, 2019.

[11] David Bau, Jun-Yan Zhu, Jonas Wulff, William Peebles, Hendrik Strobelt, Bolei Zhou, and Antonio Torralba. Seeing what a gan cannot generate. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019.

[12] David Bau, Hendrik Strobelt, William Peebles, Bolei Zhou, Jun-Yan Zhu, Antonio Torralba, et al. Semantic photo manipulation with a generative image prior. *ACM Trans. Graph.*, 2020.

[13] Michel Besserve, Arash Mehrjou, Rémy Sun, and Bernhard Schölkopf. Counterfactuals uncover the modular structure of deep generative models. In *Int. Conf. Learn. Represent.*, 2018.

[14] Irving Biederman. Recognition-by-components: a theory of human image understanding. *Psychological review*, 94(2):115, 1987.

[15] Mikołaj Bińkowski, Danica J Sutherland, Michael Arbel, and Arthur Gretton. Demystifying MMD GANs. In *Int. Conf. Learn. Represent.*, 2018.

[16] Andreas Blattmann, Robin Rombach, Huan Ling, Tim Dockhorn, Seung Wook Kim, Sanja Fidler, and Karsten Kreis. Align your latents: High-resolution video synthesis with latent diffusion models. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[17] Ashish Bora, Ajil Jalal, Eric Price, and Alexandros G Dimakis. Compressed sensing using generative models. In *Int. Conf. Machine Learning*, 2017.

[18] Richard Strong Bowen, Huiwen Chang, Charles Herrmann, Piotr Teterwak, Ce Liu, and Ramin Zabih. OCONet: Image extrapolation by object completion. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 2307–2317, 2021.

[19] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. In *Int. Conf. Learn. Represent.*, 2018.

[20] Tim Brooks, Janne Hellsten, Miika Aittala, Ting-Chun Wang, Timo Aila, Jaakko Lehtinen, Ming-Yu Liu, Alexei A Efros, and Tero Karras. Generating long videos of dynamic scenes. In *Adv. Neural Inform. Process. Syst.*, 2022.

[21] Tim Brooks, Aleksander Holynski, and Alexei A Efros. Instructpix2pix: Learning to follow image editing instructions. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 18392–18402, 2023.

[22] Peter Burt and Edward Adelson. The laplacian pyramid as a compact image code. *IEEE Transactions on communications*, 31(4):532–540, 1983.

[23] Lucy Chai, David Bau, Ser-Nam Lim, and Phillip Isola. What makes fake images detectable? understanding properties that generalize. In *Eur. Conf. Comput. Vis.*, 2020.

[24] Lucy Chai, Jun-Yan Zhu, Eli Shechtman, Phillip Isola, and Richard Zhang. Ensembling with deep generative views. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021.

[25] Lucy Chai, Michael Gharbi, Eli Shechtman, Phillip Isola, and Richard Zhang. Any-resolution training for high-resolution image synthesis. In *Eur. Conf. Comput. Vis.*, 2022.

[26] Eric R Chan, Marco Monteiro, Petr Kellnhofer, Jiajun Wu, and Gordon Wetzstein. pi-GAN: Periodic implicit generative adversarial networks for 3D-aware image synthesis. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 5799–5809, 2021.

[27] Eric R. Chan, Connor Z. Lin, Matthew A. Chan, Koki Nagano, Boxiao Pan, Shalini De Mello, Orazio Gallo, Leonidas Guibas, Jonathan Tremblay, Sameh Khamis, Tero Karras, and Gordon Wetzstein. Efficient geometry-aware 3D generative adversarial networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022.

[28] Eric R Chan, Koki Nagano, Matthew A Chan, Alexander W Bergman, Jeong Joon Park, Axel Levy, Miika Aittala, Shalini De Mello, Tero Karras, and Gordon Wetzstein. Generative novel view synthesis with 3d-aware diffusion models. *arXiv preprint arXiv:2304.02602*, 2023.

[29] Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman. Maskgit: Masked generative image transformer. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 11315–11325, 2022.

[30] Huiwen Chang, Han Zhang, Jarred Barber, AJ Maschinot, Jose Lezama, Lu Jiang, Ming-Hsuan Yang, Kevin Murphy, William T Freeman, Michael Rubinstein, et al. Muse: Text-to-image generation via masked generative transformers. In *Int. Conf. Machine Learning*, 2023.

[31] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *Int. Conf. Machine Learning*, pages 1691–1703. PMLR, 2020.

[32] Xu Chen, Jie Song, and Otmar Hilliges. Monocular neural image based rendering with continuous view control. In *Int. Conf. Comput. Vis.*, pages 4090–4100, 2019.

[33] Yinbo Chen, Sifei Liu, and Xiaolong Wang. Learning continuous image representation with local implicit image function. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 8628–8638, 2021.

[34] Zhaoxi Chen, Guangcong Wang, and Ziwei Liu. Scenedreamer: Unbounded 3d scene generation from 2d image collections. *arXiv preprint arXiv:2302.01330*, 2023.

[35] Yen-Chi Cheng, Chieh Hubert Lin, Hsin-Ying Lee, Jian Ren, Sergey Tulyakov, and Ming-Hsuan Yang. In&Out: Diverse image outpainting via GAN inversion. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 11431–11440, 2022.

[36] Inchang Choi, Orazio Gallo, Alejandro Troccoli, Min H Kim, and Jan Kautz. Extreme view synthesis. In *Int. Conf. Comput. Vis.*, pages 7781–7790, 2019.

[37] Jooyoung Choi, Jungbeom Lee, Yonghyun Jeong, and Sungroh Yoon. Toward spatially unbiased generative models. In *Int. Conf. Comput. Vis.*, 2021.

[38] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 8789–8797, 2018.

[39] Min Jin Chong and David Forsyth. Effectively unbiased fid and inception score and where to find them. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 6070–6079, 2020.

[40] Min Jin Chong, Hsin-Ying Lee, and David Forsyth. StyleGAN of All Trades: Image Manipulation with Only Pretrained StyleGAN. *arXiv preprint arXiv:2111.01619*, 2021.

[41] Taco S Cohen, Maurice Weiler, Berkay Kicanaoglu, and Max Welling. Gauge equivariant convolutional networks and the icosahedral cnn. In *Int. Conf. Machine Learning*, 2019.

[42] Edo Collins, Raja Bala, Bob Price, and Sabine Süsstrunk. Editing in style: Uncovering the local semantics of gans. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020.

[43] Yu Deng, Jiaolong Yang, Jianfeng Xiang, and Xin Tong. Gram: Generative radiance manifolds for 3D-aware image generation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 10673–10683, June 2022.

[44] Emily Denton and Rob Fergus. Stochastic video generation with a learned prior. In *Int. Conf. Machine Learning*, pages 1174–1183. PMLR, 2018.

[45] Emily Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. In *Adv. Neural Inform. Process. Syst.*, 2015.

[46] Emily Denton, Ben Hutchinson, Margaret Mitchell, and Timnit Gebru. Detecting bias with generative counterfactual face attribute augmentation. *arXiv preprint arXiv:1906.06439*, 2019.

[47] Terrance DeVries, Miguel Angel Bautista, Nitish Srivastava, Graham W Taylor, and Joshua M Susskind. Unconstrained scene generation with locally conditioned radiance fields. In *Int. Conf. Comput. Vis.*, pages 14304–14313, 2021.

[48] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. In *Adv. Neural Inform. Process. Syst.*, volume 34, 2021.

[49] Jeff Donahue and Karen Simonyan. Large scale adversarial representation learning. In *Adv. Neural Inform. Process. Syst.*, 2019.

[50] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. In *Int. Conf. Learn. Represent.*, 2017.

[51] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. Adversarially learned inference. In *Int. Conf. Learn. Represent.*, 2016.

[52] Alexei A Efros and William T Freeman. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 341–346, 2001.

[53] Alexei A Efros and Thomas K Leung. Texture synthesis by non-parametric sampling. In *Int. Conf. Comput. Vis.*, volume 2, pages 1033–1038. IEEE, 1999.

[54] Dave Epstein, Allan Jabri, Ben Poole, Alexei A Efros, and Aleksander Holynski. Diffusion self-guidance for controllable image generation. *arXiv preprint arXiv:2306.00986*, 2023.

[55] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 12873–12883, 2021.

[56] Patrick Fernandes, Aman Madaan, Emmy Liu, António Farinhas, Pedro Henrique Martins, Amanda Bertsch, José GC de Souza, Shuyan Zhou, Tongshuang Wu, Graham Neubig, et al. Bridging the gap: A survey on integrating (human) feedback for natural language generation. *arXiv preprint arXiv:2305.00955*, 2023.

[57] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *Adv. Neural Inform. Process. Syst.*, 2016.

[58] John Flynn, Michael Broxton, Paul Debevec, Matthew DuVall, Graham Fyffe, Ryan Overbeck, Noah Snavely, and Richard Tucker. DeepView: View synthesis with learned gradient descent. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 2367–2376, 2019.

[59] Gereon Fox, Ayush Tewari, Mohamed Elgharib, and Christian Theobalt. StyleVideoGAN: A temporal generative model using a pretrained StyleGAN. In *Brit. Mach. Vis. Conf.*, 2021.

[60] William T. Freeman and Edward H Adelson. The design and use of steerable filters. *IEEE Trans. Pattern Anal. Mach. Intell.*, (9):891–906, 1991.

[61] Rafail Fridman, Amit Abecasis, Yoni Kasten, and Tali Dekel. Scenescape: Text-driven consistent scene generation. *arXiv preprint arXiv:2302.01133*, 2023.

[62] Stephanie Fu*, Netanel Tamir*, Shobhita Sundaram*, Lucy Chai, Richard Zhang, Tali Dekel, and Phillip Isola. Dreamsim: Learning new dimensions of human visual similarity using synthetic data. *arXiv:2306.09344*, 2023.

[63] Rinon Gal, Yuval Alaluf, Yuval Atzmon, Or Patashnik, Amit H Bermano, Gal Chechik, and Daniel Cohen-Or. An image is worth one word: Personalizing text-to-image generation using textual inversion. *arXiv preprint arXiv:2208.01618*, 2022.

[64] Rohit Gandikota, Joanna Materzynska, Jaden Fiotto-Kaufman, and David Bau. Erasing concepts from diffusion models. *arXiv preprint arXiv:2303.07345*, 2023.

[65] Songwei Ge, Thomas Hayes, Harry Yang, Xi Yin, Guan Pang, David Jacobs, Jia-Bin Huang, and Devi Parikh. Long video generation with time-agnostic VQGAN and time-sensitive transformer. *arXiv preprint arXiv:2204.03638*, 2022.

[66] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. In *Int. Conf. Learn. Represent.*, 2019.

[67] Daniel Glasner, Shai Bagon, and Michal Irani. Super-resolution from a single image. In *Int. Conf. Comput. Vis.*, pages 349–356. IEEE, 2009.

[68] Lore Goetschalckx, Alex Andonian, Aude Oliva, and Phillip Isola. Ganalyze: Toward visual definitions of cognitive image properties. In *Int. Conf. Comput. Vis.*, 2019.

[69] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Adv. Neural Inform. Process. Syst.*, 2014.

[70] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *Int. Conf. Learn. Represent.*, 2015.

[71] Jiatao Gu, Lingjie Liu, Peng Wang, and Christian Theobalt. StyleNeRF: A style-based 3D-aware generator for high-resolution image synthesis. In *Int. Conf. Learn. Represent.*, 2022.

[72] Jinjin Gu, Yujun Shen, and Bolei Zhou. Image processing using multi-code gan prior. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019.

[73] Shuyang Gu, Jianmin Bao, Hao Yang, Dong Chen, Fang Wen, and Lu Yuan. Mask-guided portrait editing with conditional gans. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019.

[74] Shanyan Guan, Ying Tai, Bingbing Ni, Feida Zhu, Feiyue Huang, and Xiaokang Yang. Collaborative learning for faster stylegan embedding. *arXiv preprint arXiv:2007.01758*, 2020.

[75] David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Adv. Neural Inform. Process. Syst.*, volume 31, 2018.

[76] Zekun Hao, Arun Mallya, Serge Belongie, and Ming-Yu Liu. Gancraft: Unsupervised 3D neural rendering of Minecraft worlds. In *Int. Conf. Comput. Vis.*, pages 14072–14082, 2021.

[77] Erik Härkönen, Aaron Hertzmann, Jaakko Lehtinen, and Sylvain Paris. Ganspace: Discovering interpretable gan controls. In *Adv. Neural Inform. Process. Syst.*, 2020.

[78] James Hays and Alexei A Efros. Scene completion using millions of photographs. In *ACM Trans. Graphics (SIGGRAPH North America)*, 2007.

[79] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2016.

[80] Ari Heljakka, Yuxin Hou, Juho Kannala, and Arno Solin. Deep automodulators. In *Adv. Neural Inform. Process. Syst.*, 2019.

[81] Amir Hertz, Ron Mokady, Jay Tenenbaum, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. Prompt-to-prompt image editing with cross attention control. *arXiv preprint arXiv:2208.01626*, 2022.

[82] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *Adv. Neural Inform. Process. Syst.*, volume 30, 2017.

[83] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. Transforming auto-encoders. In *International Conference on Artificial Neural Networks*, pages 44–51. Springer, 2011.

[84] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

[85] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Adv. Neural Inform. Process. Syst.*, volume 33, pages 6840–6851, 2020.

[86] Jonathan Ho, William Chan, Chitwan Saharia, Jay Whang, Ruiqi Gao, Alexey Gritsenko, Diederik P Kingma, Ben Poole, Mohammad Norouzi, David J Fleet, et al. Imagen video: High definition video generation with diffusion models. *arXiv preprint arXiv:2210.02303*, 2022.

[87] Jun-Ting Hsieh, Bingbin Liu, De-An Huang, Li F Fei-Fei, and Juan Carlos Niebles. Learning to decompose and disentangle representations for video prediction. In *Adv. Neural Inform. Process. Syst.*, volume 31, 2018.

[88] Ronghang Hu, Nikhila Ravi, Alexander C. Berg, and Deepak Pathak. Worldsheet: Wrapping the world in a 3D sheet for view synthesis from a single image. In *Int. Conf. Comput. Vis.*, 2021.

[89] Xuecai Hu, Haoyuan Mu, Xiangyu Zhang, Zilei Wang, Tieniu Tan, and Jian Sun. Meta-sr: A magnification-arbitrary network for super-resolution. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1575–1584, 2019.

[90] Jia-Bin Huang, Abhishek Singh, and Narendra Ahuja. Single image super-resolution from transformed self-exemplars. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 5197–5206, 2015.

[91] Minyoung Huh, Andrew Liu, Andrew Owens, and Alexei A Efros. Fighting fake news: Image splice detection via learned self-consistency. In *Eur. Conf. Comput. Vis.*, pages 101–117, 2018.

[92] Minyoung Huh, Richard Zhang, Jun-Yan Zhu, Sylvain Paris, and Aaron Hertzmann. Transforming and projecting images into class-conditional generative networks. In *Eur. Conf. Comput. Vis.*, 2020.

[93] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Globally and locally consistent image completion. *ACM Trans. Graph.*, 2017.

[94] Michal Irani and Shmuel Peleg. Improving resolution by image registration. *CVGIP: Graphical models and image processing*, 53(3):231–239, 1991.

[95] Phillip Isola and Ce Liu. Scene collaging: Analysis and synthesis of natural images with semantic layers. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2013.

[96] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1125–1134, 2017.

[97] Ali Jahanian, SVN Vishwanathan, and Jan P Allebach. Learning visual balance from large-scale datasets of aesthetically highly rated images. In *Human Vision and Electronic Imaging*, volume 9394, page 93940Y. International Society for Optics and Photonics, 2015.

[98] Ali Jahanian, Lucy Chai, and Phillip Isola. On the "steerability" of generative adversarial networks. In *Int. Conf. Learn. Represent.*, 2020.

[99] Ali Jahanian, Xavier Puig, Yonglong Tian, and Phillip Isola. Generative models as a data source for multiview representation learning. In *Int. Conf. Learn. Represent.*, 2021.

[100] Varun Jampani, Huiwen Chang, Kyle Sargent, Abhishek Kar, Richard Tucker, Michael Krainin, Dominik Kaeser, William T Freeman, David Salesin, Brian Curless, et al. SLIDE: Single image 3D photography with soft layering and depth-aware inpainting. In *Int. Conf. Comput. Vis.*, pages 12518–12527, 2021.

[101] Wonbong Jang and Lourdes Agapito. CodeNeRF: Disentangled neural radiance fields for object categories. In *Int. Conf. Comput. Vis.*, pages 12949–12958, 2021.

[102] Yuming Jiang, Kelvin CK Chan, Xintao Wang, Chen Change Loy, and Ziwei Liu. Robust reference-based super-resolution via c2-matching. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 2103–2112, 2021.

[103] Biliana Kaneva, Josef Sivic, Antonio Torralba, Shai Avidan, and William T. Freeman. Infinite images: Creating and exploring a large photorealistic virtual space. In *Proceedings of the IEEE*, 2010.

[104] Minguk Kang, Jun-Yan Zhu, Richard Zhang, Jaesik Park, Eli Shechtman, Sylvain Paris, and Taesung Park. Scaling up gans for text-to-image synthesis. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 10124–10134, 2023.

[105] Animesh Karnewar and Oliver Wang. MSG-GAN: Multi-scale gradients for generative adversarial networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 7799–7808, 2020.

[106] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *Int. Conf. Learn. Represent.*, 2017.

[107] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019.

[108] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data. In *Adv. Neural Inform. Process. Syst.*, 2020.

[109] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020.

[110] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 8110–8119, 2020.

[111] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Alias-free generative adversarial networks. In *Adv. Neural Inform. Process. Syst.*, volume 34, 2021.

[112] Seung Wook Kim, Jonah Philion, Antonio Torralba, and Sanja Fidler. Drivegan: Towards a controllable high-quality neural simulation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021.

[113] Davis E. King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10:1755–1758, 2009.

[114] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *Int. Conf. Learn. Represent.*, 2015.

[115] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[116] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Adv. Neural Inform. Process. Syst.*, 2018.

[117] Jing Yu Koh, Honglak Lee, Yinfei Yang, Jason Baldridge, and Peter Anderson. Pathdreamer: A world model for indoor navigation. In *Int. Conf. Comput. Vis.*, pages 14738–14748, 2021.

[118] Johannes Kopf, Kevin Matzen, Suhib Alsisan, Ocean Quigley, Francis Ge, Yangming Chong, Josh Patterson, Jan-Michael Frahm, Shu Wu, Matthew Yu, Peizhao Zhang, Zijian He, Peter Vajda, Ayush Saraf, and Michael Cohen. One shot 3D photography. *ACM Trans. Graphics (SIGGRAPH North America)*, 39 (4), 2020.

[119] Adam Kortylewski, Ju He, Qing Liu, and Alan L Yuille. Compositional convolutional neural networks: A deep architecture with innate robustness to partial occlusion. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020.

[120] Nupur Kumari, Bingliang Zhang, Sheng-Yu Wang, Eli Shechtman, Richard Zhang, and Jun-Yan Zhu. Ablating concepts in text-to-image diffusion models. *arXiv preprint arXiv:2303.13516*, 2023.

[121] Nupur Kumari, Bingliang Zhang, Richard Zhang, Eli Shechtman, and Jun-Yan Zhu. Multi-concept customization of text-to-image diffusion. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1931–1941, 2023.

[122] Mingi Kwon, Jaeseok Jeong, and Youngjung Uh. Diffusion models already have a semantic latent space. *arXiv preprint arXiv:2210.10960*, 2022.

[123] Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 29–37. JMLR Workshop and Conference Proceedings, 2011.

[124] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[125] Wonkwang Lee, Whie Jung, Han Zhang, Ting Chen, Jing Yu Koh, Thomas Huang, Hyungsuk Yoon, Honglak Lee, and Seunghoon Hong. Revisiting hierarchical approach for persistent long-term video prediction. In *Int. Conf. Learn. Represent.*, 2021.

[126] Karel Lenc and Andrea Vedaldi. Understanding image representations by measuring their equivariance and equivalence. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 991–999, 2015.

[127] Marc Levoy and Pat Hanrahan. Light field rendering. In *ACM Trans. Graphics (SIGGRAPH North America)*, 1996.

[128] Jiaxin Li, Zijian Feng, Qi She, Henghui Ding, Changhu Wang, and Gim Hee Lee. MINE: Towards continuous depth MPI with NeRF for novel view synthesis. In *Int. Conf. Comput. Vis.*, pages 12578–12588, October 2021.

[129] Muheng Li, Yueqi Duan, Jie Zhou, and Jiwen Lu. Diffusion-sdf: Text-to-shape via voxelized diffusion. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 12642–12651, 2023.

[130] Wenbo Li, Zhe Lin, Kun Zhou, Lu Qi, Yi Wang, and Jiaya Jia. MAT: Mask-aware transformer for large hole image inpainting. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 10758–10768, 2022.

[131] Zhengqi Li, Qianqian Wang, Noah Snavely, and Angjoo Kanazawa. InfiniteNature-Zero: Learning perpetual view generation of natural scenes from single images. In *Eur. Conf. Comput. Vis.*, pages 515–534. Springer, 2022.

[132] Chen-Hsuan Lin, Jun Gao, Luming Tang, Towaki Takikawa, Xiaohui Zeng, Xun Huang, Karsten Kreis, Sanja Fidler, Ming-Yu Liu, and Tsung-Yi Lin. Magic3d: High-resolution text-to-3d content creation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 300–309, 2023.

[133] Chieh Hubert Lin, Chia-Che Chang, Yu-Sheng Chen, Da-Cheng Juan, Wei Wei, and Hwann-Tzong Chen. Coco-gan: Generation by parts via conditional coordinating. In *Int. Conf. Comput. Vis.*, pages 4512–4521, 2019.

[134] Chieh Hubert Lin, Hsin-Ying Lee, Yen-Chi Cheng, Sergey Tulyakov, and Ming-Hsuan Yang. InfinityGAN: Towards infinite-pixel image synthesis. In *Int. Conf. Learn. Represent.*, 2022.

[135] Chieh Hubert Lin, Hsin-Ying Lee, Willi Menapace, Menglei Chai, Aliaksandr Siarohin, Ming-Hsuan Yang, and Sergey Tulyakov. InfiniCity: Infinite-scale city synthesis. *arXiv preprint arXiv:2301.09637*, 2023.

[136] Ji Lin, Richard Zhang, Frieder Ganz, Song Han, and Jun-Yan Zhu. Anycost gans for interactive image synthesis and editing. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 14986–14996, 2021.

[137] Andrew Liu, Richard Tucker, Varun Jampani, Ameesh Makadia, Noah Snavely, and Angjoo Kanazawa. Infinite nature: Perpetual view generation of natural scenes from a single image. In *Int. Conf. Comput. Vis.*, pages 14458–14467, 2021.

[138] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[139] Hongyu Liu, Ziyu Wan, Wei Huang, Yibing Song, Xintong Han, and Jing Liao. PD-GAN: Probabilistic diverse GAN for image inpainting. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 9371–9381, 2021.

[140] Nian Liu, Junwei Han, and Ming-Hsuan Yang. Picanet: Learning pixel-wise contextual attention for saliency detection. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018.

[141] Ruoshi Liu, Rundi Wu, Basile Van Hoorick, Pavel Tokmakov, Sergey Zakharov, and Carl Vondrick. Zero-1-to-3: Zero-shot one image to 3d object. *arXiv preprint arXiv:2303.11328*, 2023.

[142] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Eur. Conf. Comput. Vis.*, pages 21–37. Springer, 2016.

[143] Yuchen Liu, Zhixin Shu, Yijun Li, Zhe Lin, Federico Perazzi, and Sun-Yuan Kung. Content-aware GAN compression. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 12156–12166, 2021.

[144] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Int. Conf. Comput. Vis.*, 2015.

[145] Stephen Lombardi, Tomas Simon, Jason Saragih, Gabriel Schwartz, Andreas Lehrmann, and Yaser Sheikh. Neural volumes: Learning dynamic renderable volumes from images. *ACM Trans. Graphics (SIGGRAPH North America)*, 38 (4):65:1–65:14, July 2019.

[146] Liying Lu, Wenbo Li, Xin Tao, Jiangbo Lu, and Jiaya Jia. Masa-sr: Matching acceleration and spatial adaptation for reference-based image super-resolution. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 6368–6377, 2021.

[147] Jingwei Ma, Lucy Chai, Minyoung Huh, Tongzhou Wang, Ser-Nam Lim, Phillip Isola, and Antonio Torralba. Totems: Physical objects for verifying visual integrity. In *Eur. Conf. Comput. Vis.*, pages 164–180. Springer, 2022.

[148] Ye Ma, Jin Ma, Min Zhou, Quan Chen, Tiezheng Ge, Yuning Jiang, and Tong Lin. Boosting image outpainting with semantic layout prediction. *arXiv preprint arXiv:2110.09267*, 2021.

[149] Arun Mallya, Ting-Chun Wang, Karan Sapra, and Ming-Yu Liu. World-consistent video-to-video synthesis. In *Eur. Conf. Comput. Vis.*, pages 359–378. Springer, 2020.

[150] Chengzhi Mao, Amogh Gupta, Augustine Cha, Hao Wang, Junfeng Yang, and Carl Vondrick. Generative interventions for causal learning. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020.

[151] Ishit Mehta, Michaël Gharbi, Connelly Barnes, Eli Shechtman, Ravi Ramamoorthi, and Manmohan Chandraker. Modulated periodic activations for generalizable local functional representations. In *Int. Conf. Comput. Vis.*, pages 14214–14223, 2021.

[152] Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. Sdedit: Guided image synthesis and editing with stochastic differential equations. In *Int. Conf. Learn. Represent.*, 2022.

[153] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for GANs do actually converge? In *Int. Conf. Machine Learning*, pages 3481–3490, 2018.

[154] Elad Mezuman and Yair Weiss. Learning about canonical views from internet image collections. In *Adv. Neural Inform. Process. Syst.*, pages 719–727, 2012.

[155] Ben Mildenhall, Pratul P Srinivasan, Rodrigo Ortiz-Cayon, Nima Khademi Kalantari, Ravi Ramamoorthi, Ren Ng, and Abhishek Kar. Local light field fusion: Practical view synthesis with prescriptive sampling guidelines. In *ACM Trans. Graphics (SIGGRAPH North America)*, 2019.

[156] 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 *Eur. Conf. Comput. Vis.*, pages 405–421. Springer, 2020.

[157] Ron Mokady, Sagie Benaim, Lior Wolf, and Amit Bermano. Mask based unsupervised content transfer. In *Int. Conf. Learn. Represent.*, 2019.

[158] Thomas Möllenhoff and Daniel Cremers. Flat metric minimization with applications in generative modeling. In *Int. Conf. Machine Learning*, 2019.

[159] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graphics (SIGGRAPH North America)*, 2022.

[160] Andres Munoz, Mohammadreza Zolfaghari, Max Argus, and Thomas Brox. Temporal shift GAN for large scale video generation. In *Proc. Winter Conf. on Computer Vision (WACV)*, pages 3179–3188, 2021.

[161] Muhammad Ferjad Naeem, Seong Joon Oh, Youngjung Uh, Yunjey Choi, and Jaejun Yoo. Reliable fidelity and diversity metrics for generative models. In *Int. Conf. Machine Learning*, 2020.

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

[163] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *Int. Conf. Machine Learning*, pages 8162–8171. PMLR, 2021.

[164] Michael Niemeyer and Andreas Geiger. CAMPARI: Camera-aware decomposed generative neural radiance fields. In *Int. Conf. on 3D Vision (3DV)*, pages 951–961. IEEE, 2021.

[165] Michael Niemeyer and Andreas Geiger. GIRAFFE: Representing scenes as compositional generative neural feature fields. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 11453–11464, 2021.

[166] Simon Niklaus, Long Mai, Jimei Yang, and Feng Liu. 3D Ken Burns effect from a single image. *ACM Trans. Graph.*, 38(6):1–15, 2019.

[167] Evangelos Ntavelis, Mohamad Shahbazi, Iason Kastanis, Radu Timofte, Martin Danelljan, and Luc Van Gool. Arbitrary-scale image synthesis. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022.

[168] OpenAI. Gpt-4 technical report. 2023.

[169] Roy Or-El, Xuan Luo, Mengyi Shan, Eli Shechtman, Jeong Joon Park, and Ira Kemelmacher-Shlizerman. StyleSDF: High-Resolution 3D-Consistent Image and Geometry Generation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 13503–13513, June 2022.

[170] Xingang Pan, Xiaohang Zhan, Bo Dai, Dahua Lin, Chen Change Loy, and Ping Luo. Exploiting deep generative prior for versatile image restoration and manipulation. In *Eur. Conf. Comput. Vis.*, 2020.

[171] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019.

[172] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019.

[173] Taesung Park, Jun-Yan Zhu, Oliver Wang, Jingwan Lu, Eli Shechtman, Alexei A. Efros, and Richard Zhang. Swapping autoencoder for deep image manipulation. In *Adv. Neural Inform. Process. Syst.*, 2020.

[174] Gaurav Parmar, Richard Zhang, and Jun-Yan Zhu. On aliased resizing and surprising subtleties in gan evaluation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022.

[175] Or Patashnik, Zongze Wu, Eli Shechtman, Daniel Cohen-Or, and Dani Lischinski. Styleclip: Text-driven manipulation of stylegan imagery. In *Int. Conf. Comput. Vis.*, pages 2085–2094, October 2021.

[176] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2016.

[177] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. In *ACM Trans. Graphics (SIGGRAPH North America)*, pages 313–318. 2003.

[178] Stanislav Pidhorskyi, Donald A Adjeroh, and Gianfranco Doretto. Adversarial latent autoencoders. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020.

[179] Marcos Pividori, Guillermo L Grinblat, and Lucas C Uzal. Exploiting gan internal capacity for high-quality reconstruction of natural images. *arXiv preprint arXiv:1911.05630*, 2019.

[180] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3D using 2D diffusion. In *Int. Conf. Learn. Represent.*, 2023.

[181] Ori Press, Tomer Galanti, Sagie Benaim, and Lior Wolf. Emerging disentanglement in auto-encoder based unsupervised image content transfer. In *Int. Conf. Learn. Represent.*, 2020.

[182] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *Int. Conf. Comput. Vis.*, 2015.

[183] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *Int. Conf. Machine Learning*, pages 8748–8763. PMLR, 2021.

[184] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision transformers for dense prediction. In *Int. Conf. Comput. Vis.*, pages 12179–12188, 2021.

[185] Suman Ravuri and Oriol Vinyals. Classification accuracy score for conditional generative models. In *Adv. Neural Inform. Process. Syst.*, 2019.

[186] Daniel Rebain, Mark Matthews, Kwang Moo Yi, Dmitry Lagun, and Andrea Tagliasacchi. LOLNeRF: Learn from one look. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1558–1567, 2022.

[187] Xuanchi Ren and Xiaolong Wang. Look outside the room: Synthesizing a consistent long-term 3D scene video from a single image. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 3563–3573, 2022.

[188] Elad Richardson, Yuval Alaluf, Or Patashnik, Yotam Nitzan, Yaniv Azar, Stav Shapiro, and Daniel Cohen-Or. Encoding in style: a stylegan encoder for image-to-image translation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021.

[189] Gernot Riegler and Vladlen Koltun. Free view synthesis. In *Eur. Conf. Comput. Vis.*, 2020.

[190] Chris Rockwell, David F Fouhey, and Justin Johnson. PixelSynth: Generating a 3D-consistent experience from a single image. In *Int. Conf. Comput. Vis.*, pages 14104–14113, 2021.

[191] Robin Rombach, Patrick Esser, and Björn Ommer. Geometry-free view synthesis: Transformers and no 3D priors. In *Int. Conf. Comput. Vis.*, pages 14356–14366, 2021.

[192] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 10684–10695, 2022.

[193] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 22500–22510, 2023.

[194] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vis.*, 115(3): 211–252, 2015.

[195] Chitwan Saharia, William Chan, Huiwen Chang, Chris Lee, Jonathan Ho, Tim Salimans, David Fleet, and Mohammad Norouzi. Palette: Image-to-image diffusion models. In *ACM Trans. Graphics (SIGGRAPH North America)*, pages 1–10, 2022.

[196] Mehdi SM Sajjadi, Olivier Bachem, Mario Lucic, Olivier Bousquet, and Sylvain Gelly. Assessing generative models via precision and recall. In *Adv. Neural Inform. Process. Syst.*, volume 31, 2018.

[197] Sara Fridovich-Keil and Alex Yu, Matthew Tancik, Qinhong Chen, Benjamin Recht, and Angjoo Kanazawa. Plenoxels: Radiance fields without neural networks. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2022.

[198] Mert Bülent Sarıyıldız, Karteek Alahari, Diane Larlus, and Yannis Kalantidis. Fake it till you make it: Learning transferable representations from synthetic imagenet clones. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 8011–8021, 2023.

[199] Axel Sauer and Andreas Geiger. Counterfactual generative networks. In *Int. Conf. Learn. Represent.*, 2021.

[200] Axel Sauer, Katja Schwarz, and Andreas Geiger. Stylegan-xl: Scaling stylegan to large diverse datasets. In *ACM Trans. Graphics (SIGGRAPH North America)*, pages 1–10, 2022.

[201] Axel Sauer, Tero Karras, Samuli Laine, Andreas Geiger, and Timo Aila. Stylegan-t: Unlocking the power of gans for fast large-scale text-to-image synthesis. In *Int. Conf. Machine Learning*, 2023.

[202] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. Laion-5b: An open large-scale dataset for training next generation image-text models. In *Adv. Neural Inform. Process. Syst.*, volume 35, pages 25278–25294, 2022.

[203] Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. In *Adv. Neural Inform. Process. Syst.*, volume 33, pages 20154–20166, 2020.

[204] Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. Singan: Learning a generative model from a single natural image. In *Int. Conf. Comput. Vis.*, pages 4570–4580, 2019.

[205] Tamar Rott Shaham, Michaël Gharbi, Richard Zhang, Eli Shechtman, and Tomer Michaeli. Spatially-adaptive pixelwise networks for fast image translation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 14882–14891, 2021.

[206] Prafull Sharma, Ayush Tewari, Yilun Du, Sergey Zakharov, Rares Ambrus, Adrien Gaidon, William T Freeman, Fredo Durand, Joshua B Tenenbaum, and Vincent Sitzmann. Seeing 3D objects in a single image via self-supervised static-dynamic disentanglement. In *Int. Conf. Learn. Represent.*, 2023.

[207] Eli Shechtman and Michal Irani. Matching local self-similarities across images and videos. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1–8. IEEE, 2007.

[208] Yuan Shen, Wei-Chiu Ma, and Shenlong Wang. SGAM: Building a virtual 3D world through simultaneous generation and mapping. In *Adv. Neural Inform. Process. Syst.*, 2022.

[209] Yujun Shen and Bolei Zhou. Closed-form factorization of latent semantics in gans. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020.

[210] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of gans for semantic face editing. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2019.

[211] Lixin Shi, Haitham Hassanieh, Abe Davis, Dina Katabi, and Fredo Durand. Light field reconstruction using sparsity in the continuous fourier domain. In *ACM Trans. Graphics (SIGGRAPH North America)*, 2014.

[212] Meng-Li Shih, Shih-Yang Su, Johannes Kopf, and Jia-Bin Huang. 3D photography using context-aware layered depth inpainting. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020.

[213] Assaf Shocher, Nadav Cohen, and Michal Irani. "zero-shot" super-resolution using deep internal learning. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 3118–3126, 2018.

[214] Assaf Shocher, Shai Bagon, Phillip Isola, and Michal Irani. Ingan: Capturing and retargeting the "dna" of a natural image. In *Int. Conf. Comput. Vis.*, pages 4492–4501, 2019.

[215] Assaf Shocher, Yossi Gandelsman, Inbar Mosseri, Michal Yarom, Michal Irani, William T Freeman, and Tali Dekel. Semantic pyramid for image generation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020.

[216] J Ryan Shue, Eric Ryan Chan, Ryan Po, Zachary Ankner, Jiajun Wu, and Gordon Wetzstein. 3d neural field generation using triplane diffusion. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 20875–20886, 2023.

[217] Joel Simon. Ganbreeder. `http:/https://ganbreeder.app/`, accessed 2019-03-22.

[218] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Int. Conf. Learn. Represent.*, 2015.

[219] Uriel Singer, Adam Polyak, Thomas Hayes, Xi Yin, Jie An, Songyang Zhang, Qiyuan Hu, Harry Yang, Oron Ashual, Oran Gafni, et al. Make-a-video: Text-to-video generation without text-video data. *arXiv preprint arXiv:2209.14792*, 2022.

[220] Ivan Skorokhodov, Savva Ignatyev, and Mohamed Elhoseiny. Adversarial generation of continuous images. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 10753–10764, 2021.

[221] Ivan Skorokhodov, Grigorii Sotnikov, and Mohamed Elhoseiny. Aligning latent and image spaces to connect the unconnectable. In *Int. Conf. Comput. Vis.*, pages 14144–14153, 2021.

[222] Ivan Skorokhodov, Sergey Tulyakov, and Mohamed Elhoseiny. StyleGAN-V: A continuous video generator with the price, image quality and perks of StyleGAN2. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 3626–3636, 2022.

[223] Ivan Skorokhodov, Sergey Tulyakov, Yiqun Wang, and Peter Wonka. EpiGRAF: Rethinking training of 3D GANs. In *Adv. Neural Inform. Process. Syst.*, 2022.

[224] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *Int. Conf. Learn. Represent.*, 2021.

[225] Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. In *Adv. Neural Inform. Process. Syst.*, volume 32, 2019.

[226] Yang Song and Stefano Ermon. Improved techniques for training score-based generative models. In *Adv. Neural Inform. Process. Syst.*, volume 33, pages 12438–12448, 2020.

[227] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[228] Roman Suvorov, Elizaveta Logacheva, Anton Mashikhin, Anastasia Remizova, Arsenii Ashukha, Aleksei Silvestrov, Naejin Kong, Harshith Goka, Kiwoong Park, and Victor Lempitsky. Resolution-robust large mask inpainting with fourier convolutions. In *Proc. Winter Conf. on Computer Vision (WACV)*, pages 2149–2159, 2022.

[229] Ryohei Suzuki, Masanori Koyama, Takeru Miyato, Taizan Yonetsuji, and Huachun Zhu. Spatially controllable image synthesis with internal representation collaging. *arXiv preprint arXiv:1811.10153*, 2018.

[230] Domen Tabernik, Matej Kristan, Jeremy L Wyatt, and Aleš Leonardis. Towards deep compositional networks. In *Int. Conf. Pattern Recog.*, 2016.

[231] Matthew Tancik, Pratul Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. In *Adv. Neural Inform. Process. Syst.*, volume 33, pages 7537–7547, 2020.

[232] Piotr Teterwak, Aaron Sarna, Dilip Krishnan, Aaron Maschinot, David Belanger, Ce Liu, and William T Freeman. Boundless: Generative adversarial networks for image extension. In *Int. Conf. Comput. Vis.*, pages 10521–10530, 2019.

[233] Ayush Tewari, Michael Zollhofer, Hyeongwoo Kim, Pablo Garrido, Florian Bernard, Patrick Perez, and Christian Theobalt. Mofa: Model-based deep convolutional face autoencoder for unsupervised monocular reconstruction. In *IEEE Conf. Comput. Vis. Pattern Recog. Worksh.*, pages 1274–1283, 2017.

[234] 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 *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020.

[235] Bart Thomee, David A Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. Yfcc100m: The new data in multimedia research. *Communications of the ACM*, 59(2):64–73, 2016.

[236] Yonglong Tian, Lijie Fan, Phillip Isola, Huiwen Chang, and Dilip Krishnan. Stablerep: Synthetic images from text-to-image models make strong visual representation learners. In *arXiv preprint arXiv:2306.00984*, 2023.

[237] Antonio Torralba and Alexei A Efros. Unbiased look at dataset bias. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2011.

[238] Richard Tucker and Noah Snavely. Single-view view synthesis with multiplane images. In *IEEE Conf. Comput. Vis. Pattern Recog.*, June 2020.

[239] Shubham Tulsiani, Richard Tucker, and Noah Snavely. Layer-structured 3D scene inference via view synthesis. In *Eur. Conf. Comput. Vis.*, pages 302–317, 2018.

[240] Sergey Tulyakov, Ming-Yu Liu, Xiaodong Yang, and Jan Kautz. Mocogan: Decomposing motion and content for video generation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1526–1535, 2018.

[241] Narek Tumanyan, Michal Geyer, Shai Bagon, and Tali Dekel. Plug-and-play diffusion features for text-driven image-to-image translation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1921–1930, 2023.

[242] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018.

[243] Paul Upchurch, Jacob Gardner, Geoff Pleiss, Robert Pless, Noah Snavely, Kavita Bala, and Kilian Weinberger. Deep feature interpolation for image content changes. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 7064–7073, 2017.

[244] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Adv. Neural Inform. Process. Syst.*, volume 29, 2016.

[245] Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *Int. Conf. Machine Learning*, pages 1747–1756. PMLR, 2016.

[246] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Adv. Neural Inform. Process. Syst.*, volume 30, 2017.

[247] Ruben Villegas, Jimei Yang, Seunghoon Hong, Xunyu Lin, and Honglak Lee. Decomposing motion and content for natural video sequence prediction. In *Int. Conf. Learn. Represent.*, 2017.

[248] Carl Vondrick and Antonio Torralba. Generating the future with adversarial transformers. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1020–1028, 2017.

[249] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. In *Adv. Neural Inform. Process. Syst.*, 2016.

[250] Qianqian Wang, Zhicheng Wang, Kyle Genova, Pratul P Srinivasan, Howard Zhou, Jonathan T Barron, Ricardo Martin-Brualla, Noah Snavely, and Thomas Funkhouser. IBRNet: Learning multi-view image-based rendering. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 4690–4699, 2021.

[251] Sheng-Yu Wang, Oliver Wang, Richard Zhang, Andrew Owens, and Alexei A Efros. Cnn-generated images are surprisingly easy to spot... for now. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2020.

[252] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. Esrgan: Enhanced super-resolution generative adversarial networks. In *Proceedings of the European conference on computer vision (ECCV) workshops*, 2018.

[253] Xintao Wang, Yu Li, Honglun Zhang, and Ying Shan. Towards real-world blind face restoration with generative facial prior. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2021.

[254] Xintao Wang, Liangbin Xie, Chao Dong, and Ying Shan. Real-esrgan: Training real-world blind super-resolution with pure synthetic data. In *Int. Conf. Comput. Vis.*, pages 1905–1914, 2021.

[255] Yi Wang, Xin Tao, Xiaoyong Shen, and Jiaya Jia. Wide-context semantic image extrapolation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 1399–1408, 2019.

[256] Yunbo Wang, Mingsheng Long, Jianmin Wang, Zhifeng Gao, and Philip S Yu. PredRNN: Recurrent neural networks for predictive learning using spatiotemporal LSTMs. In *Adv. Neural Inform. Process. Syst.*, pages 879–888, 2017.

[257] Daniel Watson, William Chan, Ricardo Martin-Brualla, Jonathan Ho, Andrea Tagliasacchi, and Mohammad Norouzi. Novel view synthesis with diffusion models. *arXiv preprint arXiv:2210.04628*, 2022.

[258] Yonatan Wexler, Eli Shechtman, and Michal Irani. Space-time completion of video. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(3):463–476, 2007.

[259] Tom White. Sampling generative networks. *arXiv preprint arXiv:1609.04468*, 2016.

[260] Olivia Wiles, Georgia Gkioxari, Richard Szeliski, and Justin Johnson. SynSin: End-to-end view synthesis from a single image. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 7467–7477, 2020.

[261] Zongze Wu, Dani Lischinski, and Eli Shechtman. Stylespace analysis: Disentangled controls for stylegan image generation. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 12863–12872, 2021.

[262] Jonas Wulff and Antonio Torralba. Improving inversion and generation diversity in stylegan using a gaussianized latent space. *arXiv preprint arXiv:2009.06529*, 2020.

[263] Bin Xia, Yapeng Tian, Yucheng Hang, Wenming Yang, Qingmin Liao, and Jie Zhou. Coarse-to-fine embedded patchmatch and multi-scale dynamic aggregation for reference-based super-resolution. *AAAI Conf. Artificial Intelligence*, 2022.

[264] Jianfeng Xiang, Jiaolong Yang, Binbin Huang, and Xin Tong. 3d-aware image generation using 2d diffusion models. *arXiv preprint arXiv:2303.17905*, 2023.

[265] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *Eur. Conf. Comput. Vis.*, 2018.

[266] Rui Xu, Xintao Wang, Kai Chen, Bolei Zhou, and Chen Change Loy. Positional encoding as spatial inductive bias in gans. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 13569–13578, 2021.

[267] Fuzhi Yang, Huan Yang, Jianlong Fu, Hongtao Lu, and Baining Guo. Learning texture transformer network for image super-resolution. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 5791–5800, 2020.

[268] Zongxin Yang, Jian Dong, Ping Liu, Yi Yang, and Shuicheng Yan. Very long natural scenery image prediction by outpainting. In *Int. Conf. Comput. Vis.*, pages 10561–10570, 2019.

[269] Yufei Ye, Maneesh Singh, Abhinav Gupta, and Shubham Tulsiani. Compositional video prediction. In *Int. Conf. Comput. Vis.*, 2019.

[270] Alex Yu, Vickie Ye, Matthew Tancik, and Angjoo Kanazawa. PixelNeRF: Neural radiance fields from one or few images. In *IEEE Conf. Comput. Vis. Pattern Recog.*, pages 4578–4587, 2021.

[271] 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 preprint arXiv:1506.03365*, 2015.

[272] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Generative image inpainting with contextual attention. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018.

[273] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Free-form image inpainting with gated convolution. In *Int. Conf. Comput. Vis.*, pages 4471–4480, 2019.

[274] Ning Yu, Larry S Davis, and Mario Fritz. Attributing fake images to gans: Learning and analyzing gan fingerprints. In *Int. Conf. Comput. Vis.*, 2019.

[275] Sihyun Yu, Jihoon Tack, Sangwoo Mo, Hyunsu Kim, Junho Kim, Jung-Woo Ha, and Jinwoo Shin. Generating videos with dynamics-aware implicit generative adversarial networks. In *Int. Conf. Learn. Represent.*, 2022.

[276] Yu Zeng, Zhe Lin, Jimei Yang, Jianming Zhang, Eli Shechtman, and Huchuan Lu. High-resolution image inpainting with iterative confidence feedback and guided upsampling. In *Eur. Conf. Comput. Vis.*, 2020.

[277] Kai Zhang, Martin Danelljan, Yawei Li, Radu Timofte, Jie Liu, Jie Tang, Gangshan Wu, Yu Zhu, Xiangyu He, Wenjie Xu, et al. Aim 2020 challenge on efficient super-resolution: Methods and results. In *Eur. Conf. Comput. Vis.*, pages 5–40. Springer, 2020.

[278] Lvmin Zhang and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models. *arXiv preprint arXiv:2302.05543*, 2023.

[279] Richard Zhang, Jun-Yan Zhu, Phillip Isola, Xinyang Geng, Angela S Lin, Tianhe Yu, and Alexei A Efros. Real-time user-guided image colorization with learned deep priors. *ACM Trans. Graph.*, 2017.

[280] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018.

[281] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *IEEE Conf. Comput. Vis. Pattern Recog.*, 2018.

[282] Xu Zhang, Svebor Karaman, and Shih-Fu Chang. Detecting and simulating artifacts in gan fake images. In *2019 IEEE International Workshop on Information Forensics and Security (WIFS)*, 2019.

[283] Yuxuan Zhang, Huan Ling, Jun Gao, Kangxue Yin, Jean-Francois Lafleche, Adela Barriuso, Antonio Torralba, and Sanja Fidler. Datasetgan: Efficient labeled data factory with minimal human effort. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10145–10155, 2021.

[284] Shengyu Zhao, Zhijian Liu, Ji Lin, Jun-Yan Zhu, and Song Han. Differentiable augmentation for data-efficient gan training. In *Adv. Neural Inform. Process. Syst.*, volume 33, pages 7559–7570, 2020.

[285] Shengyu Zhao, Jonathan Cui, Yilun Sheng, Yue Dong, Xiao Liang, Eric I Chang, and Yan Xu. Large scale image completion via co-modulated generative adversarial networks. In *Int. Conf. Learn. Represent.*, 2021.

[286] Shengyu Zhao, Jonathan Cui, Yilun Sheng, Yue Dong, Xiao Liang, Eric I Chang, and Yan Xu. Large scale image completion via co-modulated generative adversarial networks. In *Int. Conf. Learn. Represent.*, 2021.

[287] Zhengli Zhao, Zizhao Zhang, Ting Chen, Sameer Singh, and Han Zhang. Image augmentations for gan training. *arXiv preprint arXiv:2006.02595*, 2020.

[288] Haitian Zheng, Mengqi Ji, Haoqian Wang, Yebin Liu, and Lu Fang. Crossnet: An end-to-end reference-based super resolution network using cross-scale warping. In *Eur. Conf. Comput. Vis.*, pages 88–104, 2018.

[289] Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. In *ACM Trans. Graphics (SIGGRAPH North America)*, 2018.

[290] Yang Zhou, Zhen Zhu, Xiang Bai, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Non-stationary texture synthesis by adversarial expansion. In *ACM Trans. Graphics (SIGGRAPH North America)*, 2018.

[291] Jiapeng Zhu, Yujun Shen, Deli Zhao, and Bolei Zhou. In-domain gan inversion for real image editing. In *Eur. Conf. Comput. Vis.*, 2020.

[292] Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A Efros. Generative visual manipulation on the natural image manifold. In *Eur. Conf. Comput. Vis.*, 2016.

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

[294] Xiru Zhu, Fengdi Che, Tianzi Yang, Tzuyang Yu, David Meger, and Gregory Dudek. Detecting gan generated errors. *arXiv preprint arXiv:1912.00527*, 2019.