FrankenGAN: Guided Detail Synthesis for Building Mass Models Using Style-Synchonized GANs

TOM KELLY*, University College London and University of Leeds
PAUL GUERRERO*, University College London
ANTHONY STEED, University College London
PETER WONKA†, KAUST
NILOY J. MITRA†, University College London

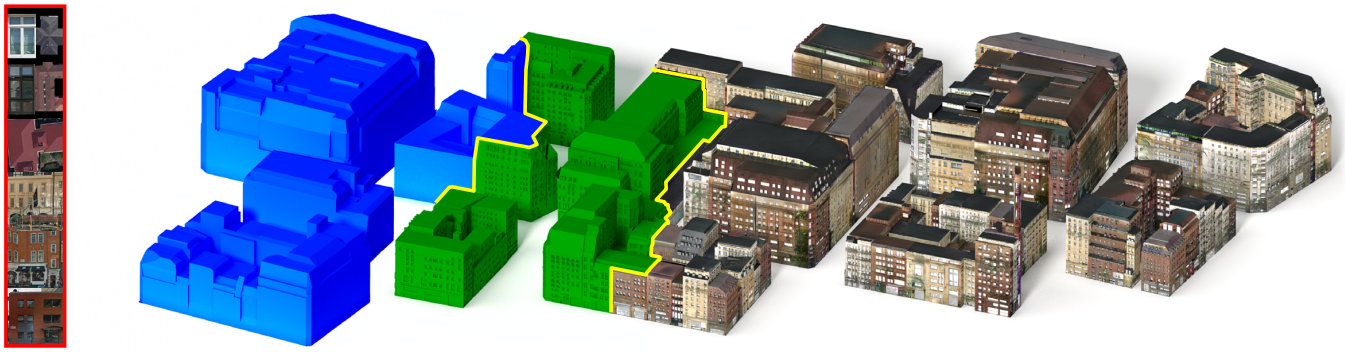

Fig. 1. We introduce FrankenGAN, a method to add detail to coarse mass models. This method allows the user to automatically generate diverse geometry and texture detail for a mass model (blue), while also giving the user control over the resulting style (through exemplar images, red). Geometric details (green) and textures (right) are generated by multiple generative adversarial networks with synchronized styles. A detailed view of this model is shown in Figure 14.

Coarse building mass models are now routinely generated at scales ranging from individual buildings to whole cities. Such models can be abstracted from raw measurements, generated procedurally, or created manually. However, these models typically lack any meaningful geometric or texture details, making them unsuitable for direct display. We introduce the problem of automatically and realistically decorating such models by adding semantically consistent geometric details and textures. Building on the recent success of generative adversarial networks (GANs), we propose FrankenGAN, a cascade of GANs that creates plausible details across multiple scales over large neighborhoods. The various GANs are synchronized to produce consistent style distributions over buildings and neighborhoods. We provide the user with direct control over the variability of the output. We allow him/her to interactively specify the style via images and manipulate style-adapted sliders to control style variability. We test our system on several large-scale examples. The generated outputs are qualitatively evaluated via a set of perceptual studies and are found to be realistic, semantically plausible, and consistent in style.

Additional Key Words and Phrases: Urban modeling, facades, texture, GANs, style.

1 INTRODUCTION

We propose a framework to add geometric and texture details to coarse building models, commonly referred to as *mass models*. There are multiple sources of such coarse models: they can be generated from procedural models, reconstructed from aerial images and LIDAR data, or created manually by urban planners, artists and architects.

By themselves, these mass models lack geometric and texture details. They therefore look unrealistic when directly displayed. For many applications, decorating these models by adding details that are automatically generated would be desirable. However, naive attempts to decorate mass models lead to unconvincing results. Examples include assigning uniform colors to different building fronts, 'attaching' rectified street-view images (when available) to the mass model faces, or synthesizing facade textures using current generative adversarial networks (see Figure 2). A viable alternative is to use rule-based procedural models, for both geometric and texture details. However, such an approach requires time and expertise. Instead, we build on the recent success of machine learning using generative adversarial networks (GANs) to simplify the modeling process and to learn facade details directly from real-world data. We focus on two issues to make GANs suitable for our application. First, current GANs are concerned only with the generation of textures, but we also require semantic and geometric details for our urban modeling applications. Second, current GANs provide little *style control*. We would like to improve control over the style of facade details. For example, a user may wish to decorate a mass model in the style of a given example facade, or to specify how similar facades should be within an urban neighborhood.

Here, we consider the problem of automatically and realistically *detailing* mass models using user-supplied building images for style-guidance, with the option to adjust style variety. By details, we refer to both geometric and texture, details. Geometric details include balconies, window frames, roof types, chimneys, etc., while texture

*joint first authors
†joint last authors

.

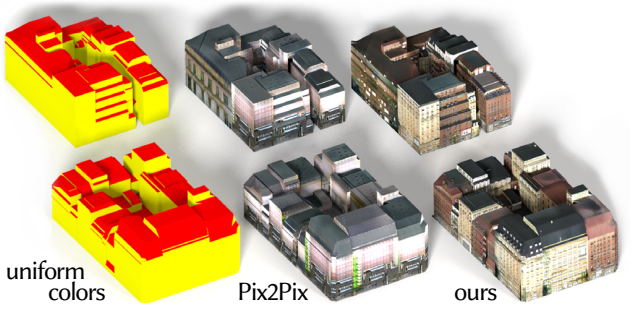

Fig. 2. Naive options to texture a mass model give unconvincing results. Pix2Pix shows heavy mode collapse, discussed in Section 6.3.

details refer to realistic facade appearances that are consistent with (latent) semantic elements such as windows, sills, etc. Further, we require the added details to be stylistically consistent and guided by user-provided input image(s). We do *not* expect the input images to be semantically-annotated or to correspond to the target mass models. The output of our algorithm (see Figure 1) is a mixture of 2.5D geometry with textures and realistic variations.

Technically, we perform detailing in stages via a cascade of GANs. We engineer the individual GANs to generate particular styles that are encoded as latent vectors. As a result, we can synchronize the different GAN outputs by selecting style vectors from appropriate distributions. We demonstrate how to perform such style-guided synthesis for both geometric and texture details. By allowing style vectors to be guided by input images, we allow users to perform, at interactive rates, drag-and-drop stylization by simply providing different target images (see supplementary video). Our system, called FrankenGAN, ensures that the resultant detailed models are realistic and are stylistically consistent both within individual buildings and between buildings in a neighborhood.

We test our system by detailing a variety of mass models over large-scale city neighborhoods. We then compare the quality of the detailed models against baseline alternatives using a perceptual study. In summary, our main contributions are: (i) introducing the problem of realistically *detailing* mass models with semantically consistent geometric and texture details; (ii) presenting FrankenGAN as an interactive system that utilizes latent style vectors via a cascade of synchronized GANs guided by examplar style images; and (iii) demonstrating the system on several large-scale examples and qualitatively evaluating the generated output. Source code and pre-trained networks are available at *http://geometry.cs.ucl.ac.uk/projects/2018/frankengan/*.

2 RELATED WORK

In the following, we review related work on the computational design of facade layouts and generative adversarial networks.

Computational facade layouts. Rule-based procedural modeling can be used to model facades and mass models [Mueller et al. 2006; Schwarz and Müller 2015; Wonka et al. 2003]. An alternative to pure procedural modeling is the combination of optimization with declarative or procedural descriptions. Multiple recent frameworks specifically target the modeling of facades and buildings [Bao et al. 2013; Bokeloh et al. 2012; Dang et al. 2014; Ilčík et al. 2015; Lin et al. 2011], and urban layouts [Vanegas et al. 2012]. There are also multiple approaches that target more general procedural modeling [Ritchie et al. 2015; Talton et al. 2011; Yeh et al. 2013].

Several methods obtain a facade layout by segmenting existing facade images, and then either fitting a procedural model [Teboul et al. 2013], or optimizing the segmentation to follow architectural principles [Cohen et al. 2014; Mathias et al. 2016]. This reconstruction is different from our *generative* approach, which synthesizes novel detail layouts without reference images.

Another important avenue of recent work is the combination of machine learning and procedural modeling. One goal is inverse procedural modeling, where grammar rules and grammar parameters are learned from data. One example approach is Bayesian model merging [Stolcke and Omohundro 1994], which was adopted by multiple authors for learning grammars [Martinovic and Van Gool 2013; Talton et al. 2012]. While this approach shares some goals with our project, the learning techniques employed were not powerful enough to encode design knowledge and thus only very simple examples were presented. Recently, deep learning was used for shape and scene synthesis of architectural content. Nishida et al. [2016] proposed an interactive framework to interpret sketches as the outer shell of 3D building models. More recently, Wang et al. [2018] used deep learning to predict furniture placement inside a room. Automatic detailing of mass models, which is the focus of our method, has not yet been attempted using a data-driven approach.

Generative adverserial networks (GANs). Supervised deep learning has played a crucial role in recent developments in several computer vision tasks, e.g., object recognition [He et al. 2016; Krizhevsky et al. 2012], semantic segmentation [Long et al. 2015], and activity recognition/detection [Escorcia et al. 2016; Tran et al. 2015]. In contrast, weakly supervised or unsupervised deep learning has been popular for image and texture synthesis tasks. In this context, Generative Adversarial Networks (GANs) have emerged as a promising family of unsupervised learning techniques that have recently been used to model simple natural images (e.g., faces and flowers) [Goodfellow et al. 2014]. They can learn to emulate the training set, enable sampling from that domain and use the learned knowledge for useful applications. Since their introduction, many variations of GANs have been proposed to overcome some of the impediments they face (e.g., instability during training)
[Arjovsky et al. 2017; Berthelot et al. 2017; Denton et al. 2015; Gurumurthy et al. 2017; Salimans et al. 2016; Springenberg 2016; Warde-Farley and Bengio 2017; Zhao et al. 2017]. Three versions of GANs are of particular interest for our work. First, the Pix2Pix framework using conditional GANs [Isola et al. 2017] is useful for image to image translation. The authors provide results for translating a facade label image into a textured facade image. Second, CycleGAN [Zhu et al. 2017a] is useful to learn image-to-image translation tasks *without* requiring a corresponding pair of images in the two styles. Third, BicycleGAN [Zhu et al. 2017b] is another extension of image to image translation that improves the variations that can be generated.

Interestingly, GANs have proven useful in several core image processing and computer vision tasks, including image inpainting [Yeh et al. 2016], style transfer [Johnson et al. 2016], super-resolution

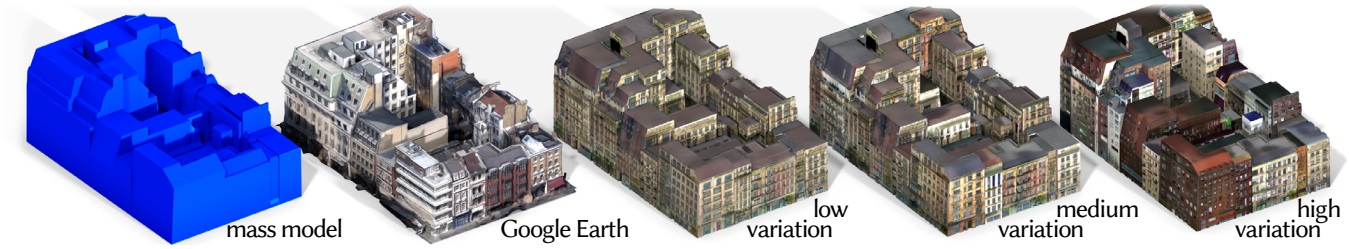

Fig. 3. In contrast to photogrammetric reconstruction (second column), our method can be used to synthesize new facade layouts and textures. Style and variation can be controlled by the user; in columns 3 to 5, we show details generated by FrankenGAN with low to high style variation.

[Isola et al. 2017; Ledig et al. 2017], manifold traversing [Zhu et al. 2016], hand pose estimation [Wan et al. 2017], and face recognition [Tran et al. 2017]. However, the applications of GANs are not limited to the computer vision and image processing communities; adversarial models are being explored for graphics applications. Examples include street network synthesis [Hartmann et al. 2017], volume rendering engines [Berger et al. 2018], and adaptive city-scene generation [Veeravasarapu et al. 2017]. In this paper, we adapt GANs to detail synthesis for detailing building mass models using exemplar images for style guidance.

3 OVERVIEW

The input to our method is a coarse building mass model with a given set of flat facades and a roof with known hip- and ridge-lines. Our goal is to generate texture and geometric details on the facades and the roof. These details are generated with a cascade of GANs. Our generative approach contrasts with the reconstruction performed by photogrammetric approaches, see Figure 3 for a comparison. In contrast to traditional GAN setups, our method allows control over the style of the synthesized outputs. It generates geometric detail in addition to texture detail. Geometric detail is generated by training the GANs to output an additional *label map* that can be used to select the type of geometry to place at each location on the facade and roof (if any).

We interleave GANs that output structural labels with those that output textures. This leads to several desirable properties that are difficult to obtain with traditional end-to-end GAN setups: Firstly, the output should exhibit a plausible structure. For example, windows tend to be arranged in grids, and the ground floor usually has a different structure than the other floors. In our experiments we found that training a GAN end-to-end makes it difficult to obtain plausible structure; the structure is never given explicitly as an objective and it must be deduced from the facade texture. The structural labels also allow us to map the output bitmaps to 3D geometry, regularize the outputs using known priors, and permit users to manually modify the structure. Secondly, the user should have some control over the style of the output. Facades of the same building usually have the same style, while the amount of style variation in a block of buildings depends on the city and area. Generating realistic city blocks therefore requires control over the style. Figure 3 presents a few examples. Thirdly, we wish to improve upon the quality achievable with a generic GAN architecture like Pix2Pix [Isola et al. 2017]. While recent work has shown remarkable quality and resolution [Karras et al. 2018], achieving this quality with a single network trained end-to-end comes at a prohibitive resource cost.

We improve these three desirable properties in our outputs at a reasonable resource cost by splitting the traditional single-GAN setup into multiple smaller steps that can be trained and evaluated separately. Synchronization across different steps using a low-dimensional embedding of style ensures the consistency of outputs. Additionally, the style embedding can be manipulated by the user, giving control over the style distribution on a building, a block or multiple blocks.

Figure 4 shows an overview of the steps performed to generate facade and roof details. In the following, we assume that the user is working on a block of buildings, but similar steps also apply to multiple building blocks or a single building. First, the user defines style distributions for the building block. Style distributions can be provided for several building *properties*, such as facade texture, roof texture, and window layouts. Each distribution is modeled as a mixture of Gaussians in a low-dimensional *style space*, where the user may choose n modes by providing n reference images (which do not need to be consistent with the building shape or each other), and optionally a custom variance. Each building in the block samples one style vector from this distribution and uses it for all windows, facades and the roof. Specifying styles gives more control over the result, but is optional; the style for any building property can instead be entirely random. Section 5.1 provides details.

After each building style is defined, two separate chains of GANs with similar architectures generate the facade and roof textures, as well as the corresponding label maps. Each GAN in the chain performs image-to-image mapping based on BicycleGAN [Zhu et al. 2017b]. We extend this architecture with several conditional inputs, including a mask of the empty facade or roof and several metrics describing the input, such as its approximate scale and a distance transform of the input boundary. This information about the global context makes it easier for a network that operates only on local patches to make global decisions, such as generating details at the correct scale, or placing objects such as doors at the correct height. Details about the GAN architecture are provided in Section 4.

To generate facade textures and label maps, three of these GANs are chained together, each performing one step in the construction of the final facade details. The first GAN generates window labels from a blank facade mask; the second GAN transform these labels into the facade texture; and the final GAN detects non-window labels in the facade texture to generate a second detailed label map. Similar steps are performed for roof textures and label maps. As we

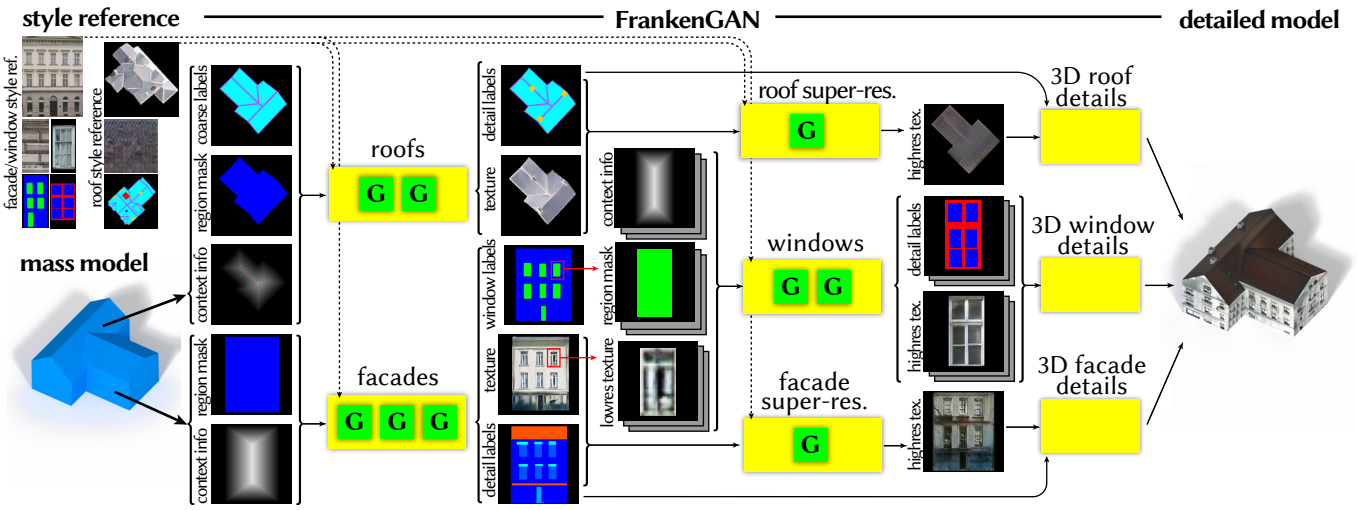

Fig. 4. FrankenGAN overview. Individual GANs are denoted by G, and yellow rectangles denote GAN chains or geometry generation modules. Given a mass model and an optional style reference (any part can be replaced by a random style), the model is detailed in three steps. First, two chains of GANs generate texture and label maps for facades and roofs. Then, the resolution of the generated textures is increased by a dedicated window generation chain and two super-resolution GANs for roofs and facades. Finally, 3D details are generated for roofs, windows, and facades based on the generated textures and label maps.

will show, this multi-step approach results in higher-quality details compared to an end-to-end approach.

The resolution of the facade and roof textures is limited by the memory requirements of the GANs. To obtain higher-resolution textures without significantly increasing the memory requirements, we employ two strategies: first, since windows are prominent features that usually exhibit fine details, we texture windows individually. A GAN chain creates window-pane labels in a first step and window textures from these labels in a second step. Second, we increase the resolution of the roof and wall textures using a super-resolution GAN applied to wall patches of fixed size. This GAN has the same architecture as the GANs used in all other steps. Details on all GAN chains are given in Section 5.2.

Finally, 3D geometric details are created based on the generated label maps using procedural geometry , and the resulting detailed mass models are textured using the generated textures. We also use label maps and textures to define decorative 3D details and material properties maps. Details are provided in Section 5.4. The source code and pre-trained network weights are available from the project webpage *http://geometry.cs.ucl.ac.uk/projects/2018/frankengan/*.

4 GAN ARCHITECTURE

Our texture and label maps are generated in multiple steps, where each step is an image-to-image transformation, implemented with a GAN. Traditional image-to-image GANs such as Pix2Pix [Isola et al. 2017] can learn a wide range of image-to-image transformations, but the variation of outputs that can be generated for a given input is limited. Since we aim to generate multiple different output styles for a given input, we base our GAN architecture on the recently introduced BicycleGAN architecture [Zhu et al. 2017b], which explicitly encodes the style in a low-dimensional *style space* and allows outputs to be generated with multiple styles. In this section, we briefly recap the BicycleGAN setup and describe our modifications.

Image-to-image GANs train a generator function,

$$B = G(A, Z) : \mathbb{R}^n \times \mathbb{R}^k \rightarrow \mathbb{R}^n, \tag{1}$$

that transforms an input image A to an output image B. For example, A might be an image containing color-coded facade labels, such as windows, and B a corresponding facade texture. The second input Z is a vector of latent variables that describe properties of the output image that are not given by the input image, such as the wall color. We call Z the *style vector* and the space containing Z the *style space* $\mathcal{Z}$. The embedding of properties into this space is learned by the generator during training. Typically, Z is chosen to be random during training and evaluation, effectively randomizing the style.

The generator's goal is to approximate some desired but unknown joint distribution $p(A, B)$ by training it with known samples from this distribution, in the form of two datasets $\mathcal{A}$ and $\mathcal{B}$ of matching input/output pairs. For example, $p(A, B)$ may be the distribution of matching pairs of facade labels and facade textures. During generator training, the difference between the generated distribution $p(A, G(A, Z))$ and the desired unknown distribution $p(A, B)$ is measured in an adversarial setup, where a *discriminator* function D is trained to distinguish between samples from the two distributions, with the following cross-entropy classification loss:

$$\begin{aligned}\mathcal{L}^D_{\text{GAN}}(G, D) = \mathbb{E}_{A,B\sim p(A,B)}\big[-\log D(A, B)\big] + \\ \mathbb{E}_{A,B\sim p(A,B),\ Z\sim p(Z)}\big[-\log\left(1 - D(A, G(A, Z))\right)\big],\end{aligned} \tag{2}$$

where $p(Z) = \mathcal{N}(0, I)$ is the prior over style vectors, defined to be a standard normal distribution. The generator is trained to output samples that are misclassified by the discriminator as being from the desired distribution:

$$\mathcal{L}^G_{\text{GAN}}(G, D) = \mathbb{E}_{A,B\sim p(A,B),\ Z\sim p(Z)}\big[-\log D(A, G(A, Z))\big]. \tag{3}$$

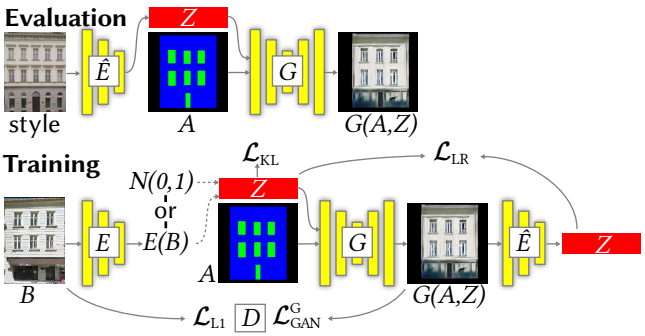

Fig. 5. GAN architecture. The setup used during evaluation is shown in the top row, and the training setup is shown in the bottom row. Dotted lines denote random sampling.

Additionally, an L1 or L2 loss term between the generated output and the ground truth is usually included:

$$\mathcal{L}_{\text{L1}}(G) = \mathbb{E}_{A,B\sim p(A,B),\ Z\sim p(Z)}\left\|B - G(A,Z)\right\|_1. \tag{4}$$

In general, the conditional distribution $p(B|A)$ for a fixed A may be a multi-modal distribution with large variance; for example, there is a wide range of possible facade textures for a given facade label image. However, previous work [Isola et al. 2017] has shown that in typical image-to-image GAN setups, the style vector Z is largely ignored, resulting in a generator output that is almost fully determined by A and restricting $p(B|A)$ to have low variance. To solve this problem, BicycleGAN uses an *encoder* E that obtains the style from an image and combines additional loss terms introduced in previous works [Donahue et al. 2016; Dumoulin et al. 2016; Larsen et al. 2016] to ensure that the style is not ignored by the generator.

First, based on ideas from Variational Autoencoders [Kingma and Welling 2014], the encoder outputs a distribution $E(B)$ of styles for each image instead of a single style vector. In Equations 2 to 4, $p(Z) = E(B)$ is used instead of the standard normal distribution. The distribution $E(B)$ is regularized to be close to a standard normal distribution to encourage style vectors to form a large contiguous region in style space that can easily be sampled:

$$\mathcal{L}_{\text{KL}}(E) = \mathbb{E}_{B\sim p(B)}\left[\mathcal{D}_{KL}(E(B)\|\mathcal{N}(0,I))\right], \tag{5}$$

where $\mathcal{D}_{KL}$ is the KL-divergence. Second, the generator is encouraged not to ignore style by including a style reconstruction term:

$$\mathcal{L}_{\text{LR}}(E) = \mathbb{E}_{A\sim p(A),Z\sim\mathcal{N}(0,I)}\left\|Z - \hat{E}(G(A,Z))\right\|_1, \tag{6}$$

where $\hat{E}$ denotes the mean of the distribution output by E. Intuitively, this term measures the reconstruction error between the style given to the generator as input and the style obtained from the generated image. The full loss for the generator and encoder is then:

$$\begin{aligned}\mathcal{L}^{G}(G,D,E) = \lambda_{\text{GAN}}\mathcal{L}^{G}_{\text{GAN}}(G,D) + \lambda_{\text{L1}}\mathcal{L}_{\text{L1}}(G) + \\ \lambda_{\text{KL}}\mathcal{L}_{\text{KL}}(E) + \lambda_{\text{LR}}\mathcal{L}_{\text{LR}}(E).\end{aligned} \tag{7}$$

The hyper-parameters λ control the relative weight of each loss. A diagram of this architecture is shown in Figure 5.

FrankenGAN trains a BicycleGAN for each individual step, with one exception that we discuss in Section 5. In addition to style, we also need to encode the real-world scale of the input mass model, so that the output details can be generated in the desired scale. We condition the GANs on an additional constant input channel that contains the scale of the facade in real-world units. This channel is appended to A.

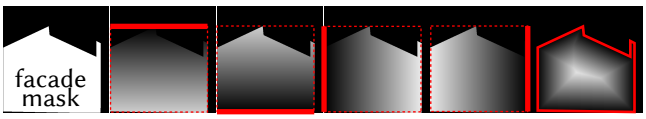

Fig. 6. Additional input channels. GANs are conditioned on additional channels that include information about the global context at each pixel. Given a facade/roof mask, we include the distance to the facade boundary and the distance to each bounding box side, making it easy for the network to decide how far it is from the boundary and at what height on the facade.

In our experiments, we observed that using this BicycleGAN setup to go directly from inputs to outputs in each step often resulted in implausible global structure, such as misaligned windows or ledges on facades. The limited receptive field of outputs in both the generator and the discriminator constrains coordination between distant output pixels, making it difficult to create globally consistent structure. Increasing the depth of the networks to increase the receptive field alleviates the problem but has a significant resource cost and destabilizes training. We found that conditioning the GANs on additional information about the global context of each pixel was more efficient. More specifically, we conditioned the GANs on five additional channels that are appended to A: the distance in real world units to each side of the bounding box and to the nearest boundary of a facade or roof. Examples are shown in Figure 6.

5 FRANKENGAN

Detail generation is performed by a cascade of textures and label maps, as shown in Figure 4. These are generated by FrankenGAN in several separate chains of GANs, where each GAN is trained and run independently. Splitting up this task into multiple steps rather than training end-to-end has several advantages. First, we can provide more guidance in the form of intermediate objectives, for example window layouts, or low-resolution textures. In our experiments, we show that such intermediate objectives provide a significant advantage over omitting this guidance. While in theory there are several ways to provide intermediate objectives for end-to-end networks, for example by concatenating our current GANs, this would result in extremely large networks, leading to the second advantage of our approach: GAN training is notoriously unstable, and training small GANs is more feasible than training large ones. An end-to-end network with intermediate objectives would need to have a very large generator with multiple discriminators, making stable training difficult achieve. In addition, splitting the network reduces resource costs during training. Instead of a single very large network, we can separately train multiple smaller networks. Note that training a very large network one part at a time would require storing and loading parts of the network from disk in each forward and backward pass, which is prohibitive in terms of training times. Finally, using separate GANs, we can regularize intermediate results with operations that are not are not differentiable or would not provide a good gradient signal.

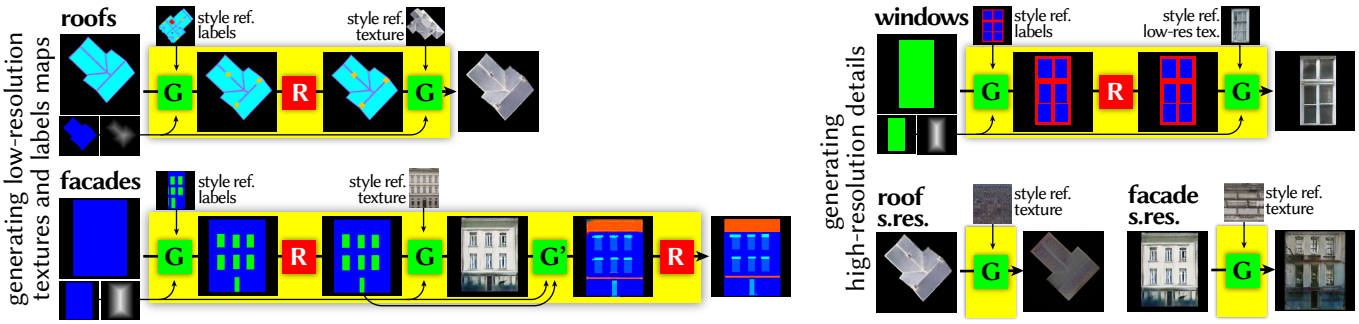

Fig. 7. FrankenGAN details. Each GAN chain (yellow rectangles) consists of several GANs (G) that each perform an image-to-image transformation. GANs are usually conditioned on additional inputs (arrows along the bottom) and are guided by a reference style (arrows along the top). Label outputs are regularized (R) to obtain clean label rectangles. Figure 4 shows these chains in context.

5.1 Style Control

One difficulty with using separate GANs is achieving stylistically consistent results. For example, windows on the same facade usually have a similar style, as do ledges or window sills. Style control is also necessary beyond single roof or facade textures: adjacent facades on a building usually look similar, and city blocks may have buildings with similar facade and roof styles. A comparison of generated details with and without style control is given in Figure 8. In FrankenGAN, style can be specified for eight properties: the coarse facade and roof texture, facade and roof texture details, such as brick patterns, the window layout on the facade, the glass pane layout in the windows, the window texture, and the layout of chimneys and windows on a roof. The user can describe the style distribution of a property over a building block with a mixture of isotropic Gaussians in style space $\mathcal{Z}$:

$$p(Z|\mathbf{S}, \boldsymbol{\sigma}) = \sum_{i=1}^{m} \phi_i \mathcal{N}(E(S_i), \sigma_i), \tag{8}$$

where $Z \in \mathcal{Z}$ is the style vector, $\mathcal{N}$ is the normal distribution and the weights ϕ_i must sum to 1. Z provides a compact representation of style. We use an eight-dimensional style space in our experiments. The means of the Gaussians are specified by encoding m style reference images $S_i \in \mathbf{S}$ with the encoder described in the previous section. The variance $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_m)$ specifies the diversity of the generated details and can be adjusted per reference image with a slider. One of these distributions can be specified per property and the styles are sampled independently.

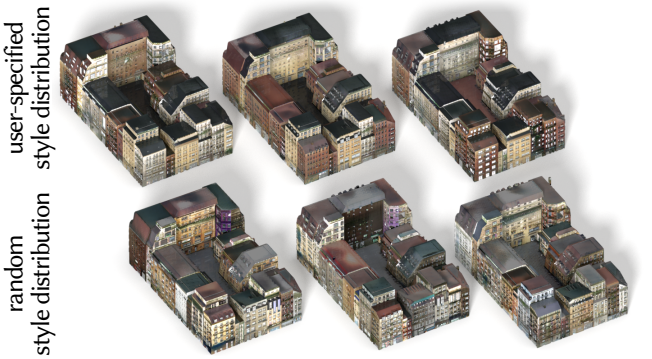

Fig. 8. Specifying a style distribution gives control over the generated details. The top row shows three results generated with the same user-specified style distribution, while the bottom row uses the style prior, giving random styles. Note how the buildings in the top row have a consistent style while still allowing for some variation (depending on the variance chosen by the user), while the bottom row does not have a consistent style.

In many cases, however, the styles of different properties are dependent. For example, the color of roofs and facades may be correlated. To specify these dependencies, several sets of property distributions may be specified, each set $\mathcal{S}_i = \{(\mathbf{S}_p, \boldsymbol{\sigma}_p)\}_{p=1\ldots8}$ contains one mixture model per property p. For each building, one of these sets is chosen at random. The special case of having a single Gaussian ($m = 1$) per property in each set effectively gives a Gaussian mixture model over the *joint* space of all properties, with each set being one component. The user does not need to provide the style for all properties. Any number of properties may be left unspecified, in which case the style vector is sampled from the GAN's style prior, which is a standard normal distribution.

5.2 Detail Generation

FrankenGAN uses five chains of GANs, which can be split into two groups: two chains for generating initial coarse details (textures and label maps) for roofs and facades, and three chains for increasing the resolution given the coarse outputs of the first group. Details of these chains are shown in Figure 7. Most of the chains have intermediate results, which are used for the geometry synthesis that we will describe in Section 5.4. Each GAN takes an input image and outputs a transformed image. In addition to the input image, all GANs except for the super-resolution networks are conditioned on the scale and context information described in the previous section, making it easier to generate consistent global structure. Each GAN is also guided by a style that is drawn from a distribution, as described above. Figure 7 shows reference images S_i that are used to specify the distribution. Images output by GANs are either label maps L, or textures T. Each label map output by a GAN is passed through a regularizer R, denoted by the red rectangles in Figure 7, to produce a clean set of boxes $R(L)$ before being passed to the next step. We now describe each chain in detail. The regularizers are described in Section 5.3.

Roofs. The roof chain generates roof detail labels and the coarse roof texture. The chain starts with a coarse label map of the roof as the input image. This label map includes ridge and valley lines of the roof, which are obtained directly from the mass model. Flat

Fig. 9. Super-resolution. Given inputs (green, magenta), the super-resolution network creates high-quality textures for the walls, while the window GAN chain provides high-quality windows. Note that the window label and window texture networks each run once for every window.

roofs are labeled with a separate color. The first GAN adds chimneys and pitched roof windows. These labels are regularized and then used by the second GAN to generate the coarse roof texture.

Facades. The facade chain generates window labels, full facade labels, and the coarse facade texture. Window labels are generated separately from the full labels, since they may be occluded by some of the other labels. The first GAN starts by creating window and door labels from facade boundaries. These are regularized before being used as input in the second GAN to generate the coarse facade texture. The third GAN detects the full set of facade labels from the facade texture, including ledges, window sills, and balconies, which are also regularized to give a cleaner set of labels. The third GAN has a different architecture: since we expect there to be a single correct label map for each facade texture, we do not need style input, which simplifies the GAN to the Pix2Pix architecture [Isola et al. 2017]. Since the window and door labels are known at this point, we also condition this GAN on these labels. Detecting the full label set from the facade texture instead of generating it beforehand and using it as input for the texture generation step is a design choice that we made after experimenting with both variants. Detail outlines in the generated texture tend to follow the input labels very closely, and constraining the details in this way results in unrealistic shapes and reduced variability. For all three GANs, areas occluded by nearby facades are set to the background colour; this ensures that the feature distribution takes into account the nearby geometry.

Since the layout of dormer windows needs to be consistent with the facade layout, we create these windows in the facade chain. More specifically, we extend the input facade mask with a projection of the roof to the facade plane. This allows us to treat the roof as part of the facade and to generate a window layout that extends to the roof. Roof features (chimneys or pitched windows) that intersect dormer windows are removed.

Windows. To obtain high-resolution window textures, we apply the window chain to each window separately, using a consistent style. Each window is cut out from the window label map that was generated in the facade chain, and scaled up to the input resolution of the GAN. The steps in the window chain are then similar to those in the roof chain. We generate glass pane labels from the window region, regularize them, and generate the high-resolution window texture from the glass pane labels.

Super-resolution. High-resolution roof and facade textures are obtained with two GANs that are trained to generate texture detail, such as bricks or roof tiles from a given low-resolution input. Roof and facade textures are split into a set of patches that are processed separately. Each patch is scaled up to the input size of the GANs before generating the high-resolution output. Consistency can be maintained by fixing the style for the entire building. The output patches are then assembled to obtain the high-resolution roof and facade textures. Boundaries between patches are blended linearly to avoid seams. Examples are shown in Figure. 9. Our interleaved GANs allow us to augment the super-resolution texture map with texture cues from the label maps. For example, window sills are lightened, and roof crests are drawn; these augmentations take the form of drawing the labels in a single colour with a certain alpha. *Note that because of the large scale of the super-resolution bitmaps, we explicitly state which figures use these two networks.*

5.3 Regularizers

Our GANs are good at producing varied label maps that follow the data distribution in our training set. Alignment between individual elements is, however, usually not perfect. For example, window sills may not be rectangular or have different sizes in adjacent windows, or ledges may not be perfectly straight. Although the discrepancy is usually small, it is still noticeable in the final output. Our multi-step approach allows us to use any non-differentiable (or otherwise) regularization. We exploit domain-specific knowledge to craft simple algorithms to improve the alignment of the label maps. We then provide 3D locations for geometric features. In the following, we describe our regularizers in detail.

Roof detail labels. Chimneys and pitched roof windows are regularized by fitting circles to the network output. These are then converted to rectangles, which may be oriented to the roof pitch, see Figure 10, top row. We take the center of each connected component of a given label and use its area to estimate a circle size. Smaller circles are removed before we convert each to a square. We observe that roof features such as chimneys and roof windows are typically aligned to the roof's slope. We therefore orient the bottom edge of each feature to the gutter of the associated roof pitch. Finally, we shrink it so that it lies entirely within a single roof pitch and avoids dormer windows.

Facade window and door labels. The window and door layout on a facade has to be regularized without removing desirable irregularities introduced by the GAN that reflect the actual data distribution, such as different window sizes on different floors, or multiple overlayed grid layouts. We start by fitting axis-aligned bounding boxes to doors and windows (see Figure 10, second row center). Then we collect a set of properties for each window, including the x and y extents and the spacing between neighbours, over which we perform a mean-shift clustering. We use a square kernel of size 0.4

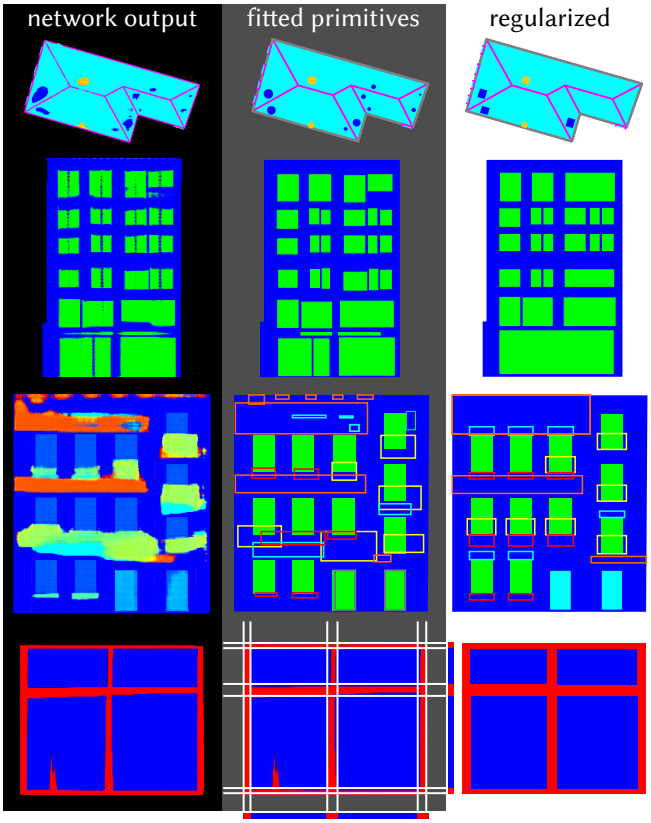

Fig. 10. Regularization. All our label maps are regularized before being used by the next GAN in the chain. Rows: roof details, facade windows, facade details, and window details (regularized with an outer product).

meters for each property, until convergence or a maximum of 50 mean-shift iterations. This ensures that these properties can have a multi-modal distribution, which preserves desirable irregularities, while also removing small-scale irregularities (see Figure 10, second row right).

Facade detail labels. Since adjacent details are often not perfectly aligned, we snap nearby details, such as window sills and windows to improve the alignment. We also observed that in the generated label map, the placement of small details such as window sills and moldings is sometimes not coordinated over larger distances on the facade. To improve regularity, we propagate details such as window sills and moldings that are present in more than 50% of the windows in a row to all remaining windows (see Figure 10, third row).

Window detail labels. The glass pane layout in a window is usually more regular than the window layout on facades, allowing for a simpler regularization: we transform the label map into a binary glass pane mask and approximate this 2D mask by the outer product of two 1D masks, one for the columns, and one for the rows of the 2D mask. This representation ensures that the mask contains a grid of square glass panes. The two 1D masks are created by taking the mean of the 2D mask in the x- and y-directions, and thresholding them at 0.33. An example is shown in Figure 10, bottom row.

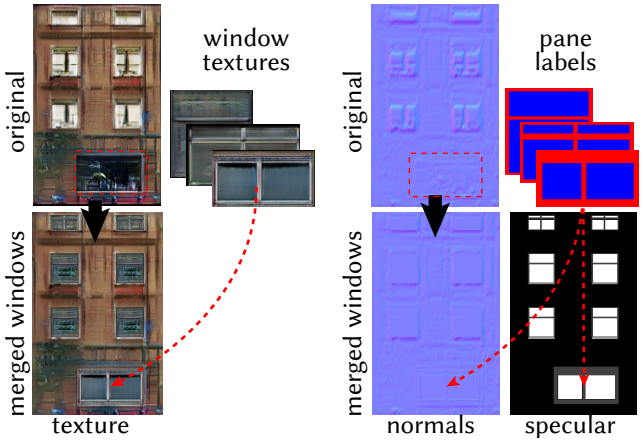

Fig. 11. Generated window textures and labels are merged back into the facade texture, increasing fidelity of window textures, normals and materials.

5.4 Geometry Synthesis

As output of the five GAN chains, we have high-resolution roof, facade, and window textures, as well as regularized label maps for roof details, facade details, and window panes. These are used to generate the detailed mass model. First, geometry for details is generated procedurally based on the label maps. For details such as window sills and ledges, we use simple extrusions, while balconies and chimneys are generated with small procedural programs to fit the shape given by the label map.

To apply the generated textures to the detailed mass models, UV maps are generated procedurally along with the geometry. In addition to textures, we also define building materials based on the label maps. Each label is given a set of material properties: windows, for example, are reflective and have high glossiness, while walls are mostly diffuse. To further increase the fidelity of our models, textures and label maps are used to heuristically generate normal maps. The intensity of the generated texture is treated as a height field that allows us to compute normals. While this does not give us accurate normals, it works well in practice to simulate the roughness of a texture. Per-label roughness weights ensure that details such as glass panes still remain flat. Finally, generated window textures and label maps are merged back into the facade textures; an example is given in Figure 11.

5.5 User interface

Our system contains a complete framework for interactively using FrankenGAN. A user may select an urban city block, specify a distribution, and then the system adds the additional geometry and textures to the 3D view. At this point, the user can edit semantic details (such as window locations), while seeing texture updates in real time. Of note is our interface to build our joint distributions (Figure 12), which continually shows the user new examples drawn from the current distribution. The accompanying video demonstrates the user interface.

6 RESULTS

We evaluate our method on several scenes consisting of procedurally generated mass models. We qualitatively show the fidelity of our

Fig. 12. The distribution designer UI (see supplemental video). Given a style distribution (a), the system continuously shows evaluations of that distribution (c). By clicking on an image, the user can see the network inputs (b). Different networks can be selected (d). The style distribution for any network is a Gaussian mixture model that may have multiple modes (e), the mean of which is given by an exemplar image.

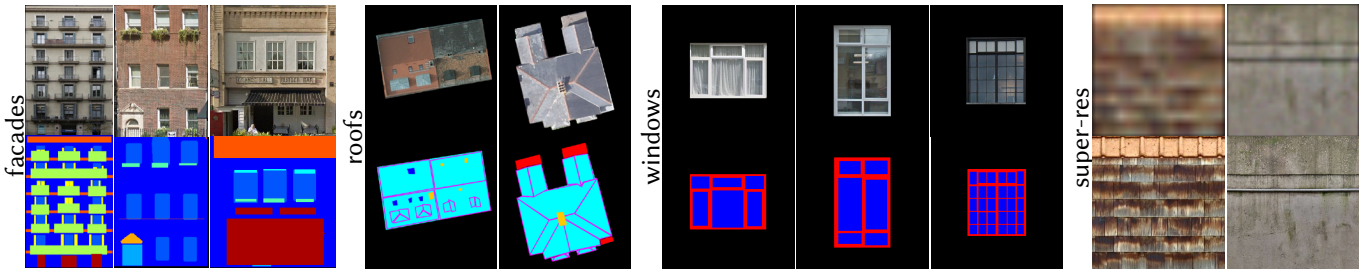

Fig. 13. Datasets used to train our GANs. We use four datasets of labeled images, a few examples are shown here.

output and the effects of style and scale control. Style and scale control are also evaluated quantitatively with a perceptual study, and we provide comparisons to existing end-to-end networks, both qualitatively and quantitatively through another perceptual study.

6.1 Datasets and Training Setup

Each GAN in our framework performs an image-to-image transformation that is trained with a separate dataset of matched image pairs. Matched images for these pairs are obtained from three datasets:

The *facade* dataset consists of the CMP dataset [Tyleček and Šára 2013] and a larger dataset of labeled facades that has not yet been released, but that has been made available to us by the authors. The combined dataset contains 3941 rectified facades with labels for several types of details, including doors, windows, window sills, and balconies. We further refined this dataset by removing heavily occluded facades and by annotating the height of a typical floor in each facade to obtain the real-world scale that our GANs are conditioned on, as described in Section 4. From this dataset, we create matched pairs of images for each GAN in the facade chain.

The *roof* dataset consists of 585 high-quality roof images with labeled roof area, ridge/valley lines, pitched windows and chimneys. The images are part of an unreleased dataset. We contracted professional labellers to create high-quality labels. From this dataset, we created matched pairs of images for the two GANs in the roof chain.

The *window* dataset contains 1376 rectified window images with labeled window areas and glass panes. These images were obtained from Google Street View, and high-quality labels were created by professional labellers. From this dataset, we created matched pairs of images for the two GANs in the window chain. Examples from all datasets are shown in Figure 13.

In addition to these three datasets, the super-resolution GANs were trained with two separate datasets. These datasets were created from a set of high-quality roof/wall texture patches that was downloaded from the internet, for example, brick patterns or roof shingles, and blurring them by random amounts. The networks are then trained to transform the blurred image to the original image. We found that it increased the performance of our networks to add in a second texture at a random location in the image. This accounts for changes in the texture over a facade or roof that occur in natural images.

Table 1. GAN statistics: the size of the training data (n), resolution (in pixels squared), number of epochs trained, and if the network takes style as input.

	n	resolution	epochs	style
roof labels	555	512	400	yes
roof textures	555	512	400	yes
facade window labels	3441	256	400	yes
facade textures	3441	256	150	yes
facade full labels	3441	256	335	no
window labels	1176	256	200	yes
window textures	1176	256	400	yes
facade super-resolution	2015	256	600	yes
roof super-resolution	1122	256	600	yes

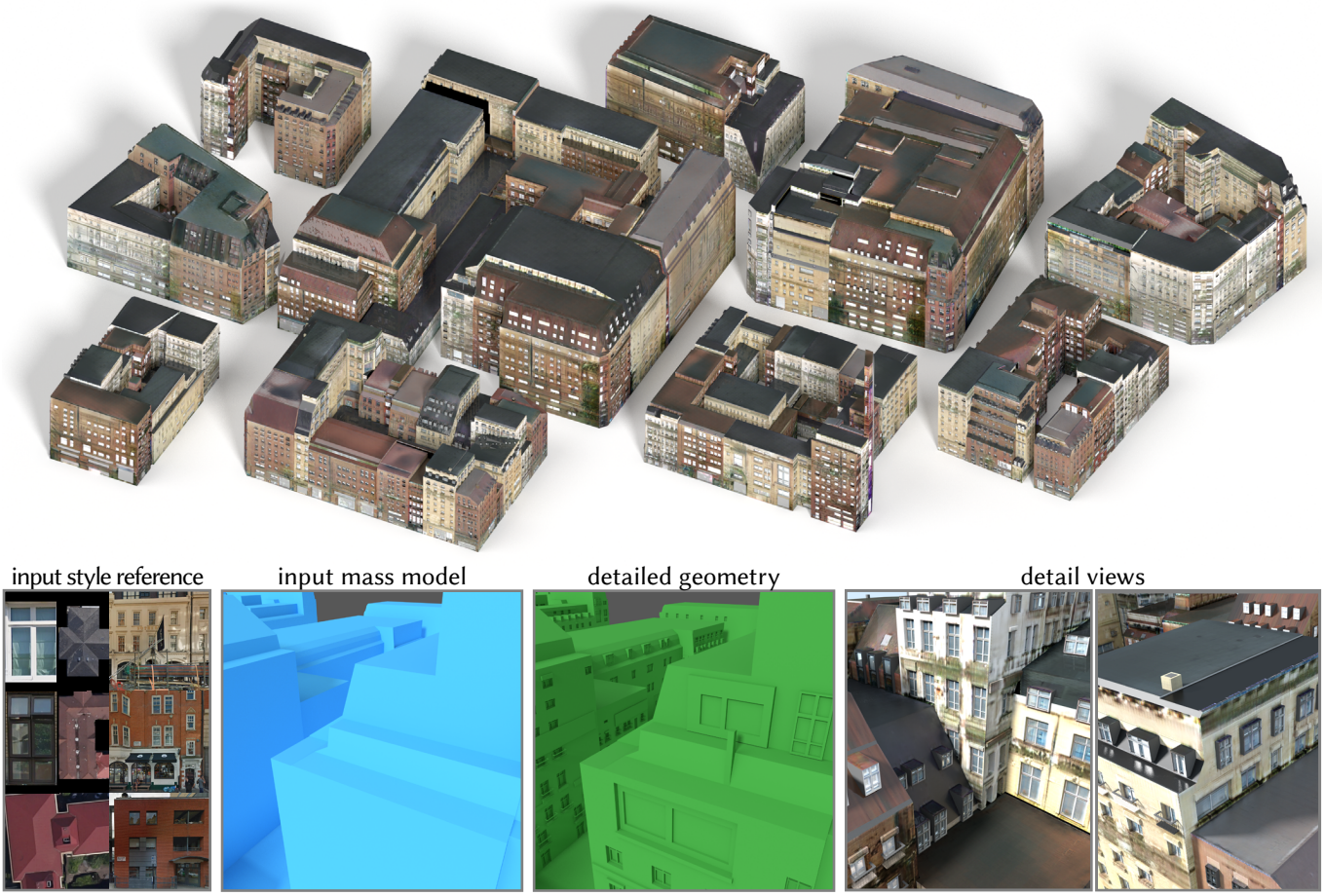

Fig. 14. Detailed London area. The output of our method is shown on top, and the input style images at the bottom left. This is followed, from left to right, by close-ups of the input mass models, detail geometry generated by our method, and two detailed views of the generated model using super-resolution textures.

To train each GAN, we alternate between discriminator optimization steps that minimize Eq. 2 and generator/encoder optimization steps that minimize Eq. 7. The optimization is performed with Adam [Kingma and Ba 2015]. The weights $(\lambda_{\text{GAN}}, \lambda_{\text{L1}}, \lambda_{\text{KL}}, \lambda_{\text{LR}})$ in Equation 7 are set to $(1, 10, 0.01, 0.5)$. A large λ_{L1} encourages results that are close to the average over all training images. This helps to stabilize training for textures. For label map generation, however, there is usually one dominant label, such as the wall or the roof label, and a large $L1$ loss encourages the generation of this label over other labels. Lowering the $L1$ loss to 1 improves results for label maps. Statistics for our GANs are summarized in Table 1.

At test time, we evaluate our networks on mass models created through procedural reconstruction from photogrammetric meshes [Kelly et al. 2017]. Facades and roofs on these mass models are labeled, and roof ridges and valleys are known, providing all necessary inputs for FrankenGAN: facade masks and coarse roof label maps. Note that it obtaining these inputs automatically from any type of reasonably clean mass model, is also feasible.

6.2 Qualitative Results

We show the quality of our results with one area of Madrid and one area of London, both spanning several blocks. As our reference style, we take images of facades, windows and roofs, some of which are taken from our dataset, and some from the web. Figures 1 and 14 show the result of detailing the London scene. Reference style textures and input mass models are shown on the left, our detailed result on the right. We produce varied details that are not completely random, but guided by the style given as input. In this example, several sets of style textures are used. Note the varied window layouts and textures: each building has unique layouts and textures that are never repeated.

Figure 15 shows our results of detailing the Madrid scene. Reference style images and a detailed view of the input mass models are shown on the left; our output is shown in the center and on the right, including a detailed view of the generated geometry. Note several modes of matching styles in the result, including red buildings with black roofs and yellow buildings that often have several ledges.

Statistics for the London and Madrid scenes are shown in Table 2. Each of the scenes has 10 blocks and contains an average of 21 buildings (the number of buildings equals the number of roofs). All of the generated details, including their textures, are unique in the entire scene. In the London scene, we generate approximately 860 Megapixels of texture and 2.68 million triangles; in the Madrid scene

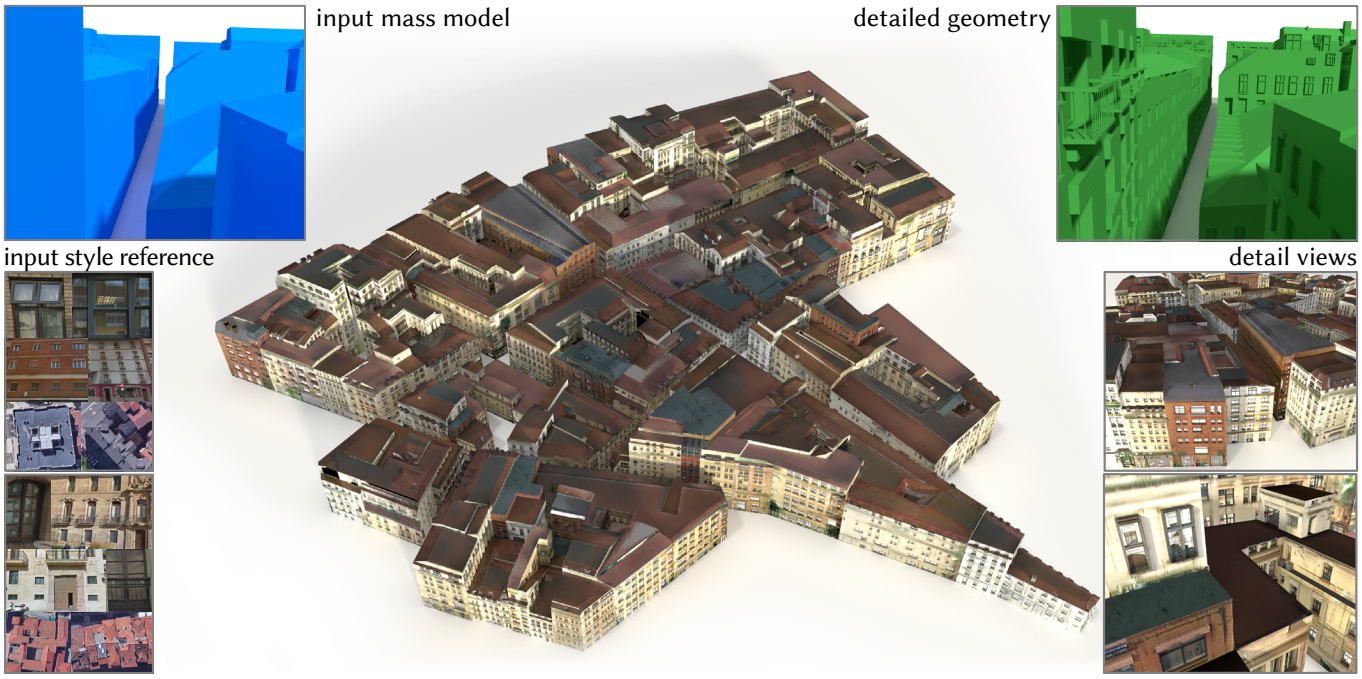

Fig. 15. Detailed Madrid area. Input style images and mass models are shown on the left, an overview of our output in the center, and close-ups of our output and the generated geometry (green), showing details like balconies and window moldings, on the right. Lower right panel uses super-resolution textures.

Table 2. Statistics for the London and Madrid scenes. We show the number of roofs, facades and windows in each block, as well as the time taken to generate the block.

London				
block	roofs	facades	windows	time(s)
1	29	145	1075	490
2	20	204	541	351
3	15	87	536	222
4	25	133	1040	400
5	47	243	2107	809
6	27	171	1622	597
7	10	65	1144	403
8	7	40	559	199
9	8	42	786	271
10	26	158	1566	577
total	214	1288	10976	4322

Madrid				
block	roofs	facades	windows	time(s)
1	22	146	773	315
2	20	110	559	239
3	17	103	471	219
4	12	67	399	166
5	7	66	230	102
6	22	116	758	291
7	25	139	571	292
8	22	125	611	255
9	35	240	1219	495
10	37	222	738	350
total	219	1334	6329	2722

we generate approximately 560 Megapixels and 1.17 million triangles. The time taken to generate the scenes on a standard desktop PC with an Intel 7700k processor, 32 GB of main memory, and an NVidia GTX 1070 GPU is shown in the last column of Table 2. The FrankenGAN implementation has two modes - when minimizing GPU memory, a single network is loaded at a time, and all applicable tasks in a scene are batch processed; when GPU memory is not a constraint, the whole network can be loaded at once. These modes use 560 MB and 2371 MB of GPU memory, respectively, leaving sufficient space on the graphics card to interactively display the textures. These figures compare favourably with larger end-to-end networks that require more memory [Karras et al. 2018].

6.3 Qualitative Comparisons

In this section, we qualitatively evaluate the style control of FrankenGAN and compare our results to two end-to-end GANs: Pix2Pix [Isola et al. 2017] and BicycleGAN [Zhu et al. 2017b]. Similar quantiative evaluations and comparisons, based on two perceptual studies, are provided in the next section.

To evaluate the effects of style control on our outputs, we compare four different types of style distributions applied to similar mass models in Figure 16. On the left, a constant style vector is used for all houses. This results in buildings with similar style, which may, for example, be required for row houses or blocks of apartment buildings. A fully random style is obtained by sampling the style prior, a standard normal distribution. The diversity in this case approximates the diversity in our training set. Using a Gaussian mixture model (GMM) as the style distribution, as described in Section 5.1, gives more control over the style. Note the two modes of white and red buildings in the third column, corresponding to two reference style images. The style for different building properties, such as roof and facade textures, is sampled independently, resulting in randomly re-mixed property styles, both white and red facades, for example, are mixed with black and red roofs. When specifying multiple sets of GMMs, one set is randomly picked for each house, allowing the user to model dependencies between property styles, as shown in the last column.

Examples of different super-resolution styles are shown in Figure 17. Since a low-resolution texture contains little information about fine texture detail, like the shingles on a roof or the stone texture on a wall, there is a diverse range of possible styles for this fine texture detail, as shown in the middle and right building.

Figure 18 shows a qualitative comparison to Pix2Pix and BicycleGAN. We trained both end-to-end GANs on our facade and roof datasets, transforming empty facade and roof masks to textures in one step. As discussed in Section 4, Pix2Pix, like most image-to-image GANs, has low output diversity for a given input image. In

Fig. 16. Different types of style distributions. From left to right, a constant style is used for all houses, a style chosen randomly from the style prior, a style sampled independently for all building properties, and a dependently sampled property style. Note that in the last two columns, there are several separate modes for properties, such as wall and building color, that are either mixed randomly for each building (third column) or sampled dependently (last column).

our case, the input image, being a blank facade or roof rectangle, does not provide much variation either, resulting in a strong mode collapse for Pix2Pix, shown by the repeated patterns such as the 'glass front' pattern across multiple buildings. Like FrankenGAN, BicycleGAN takes a style vector as input, which we set to a similar multi-modal distribution as in our method. There is more diversity than for Pix2Pix, but we observe inconsistent scale and style across different buildings, or across different facades of the same building, less realistic window layouts, and less diversity than in our results. Splitting the texture generation task into multiple steps allows us to provide more training signals, such as an explicit ground truth for the window layout, without requiring a very large network, regularize the intermediate results of the network and then use the intermediate results as label maps that can be used to generate geometric detail and assign materials. Additionally, giving the user more precise control over the style results in more consistent details across a building or parts of buildings. This allows us to generate more diverse and realistic building details. In the next section, we quantify the comparison discussed here with a perceptual study.

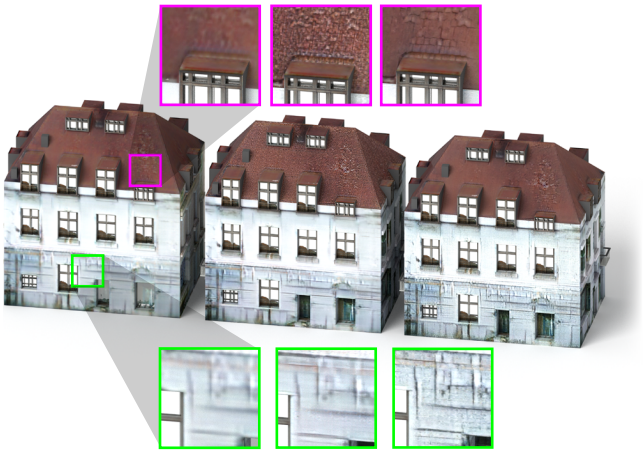

Fig. 17. Different super-resolution styles. The original low-resolution facade and roof textures are shown on the left; the middle and right buildings show two different super-resolutions styles, resulting in different textures for the roof tiles or stone wall.

6.4 Perceptual Studies

We performed two perceptual studies to quantify the following questions:

(A) *How visible are the effects of style and scale control?*
(B) *How does the realism of FrankenGAN compare with that of Pix2Pix [Isola et al. 2017] and BicycleGAN [Zhu et al. 2017b]?*

To investigate question A, we tested if participants could reliably tell which of two generated facades was guided by a given reference style facade. The style and scale was randomized for the other facade. Question B was investigated by comparing facades generated by FrankenGAN and one of the other two methods side-by-side and asking participants to compare the realism of the two facades. We performed both studies in Amazon Mechanical Turk (AMT). Screenshots of both studies are shown in Figure 19; the assignment of images to the left and right positions was randomized.

For study A, we created 172 triples of facades, each consisting of a reference facade, a facade with style and scale given by the reference facade, and a third facade with randomized style. Participants were asked which of the two facades was more similar to the reference. Each triple was evaluated by an average of 11 participants, and a total of 116 unique participants took part in the study. Figure 19 (green vs. blue) shows the average probability of a participant choosing either the style-guided facade or the facade with randomized style as more similar to the reference facade. Our results show that style-guided facade was consistently chosen, implying good visibility of style and scale control.

The second study was performed in two parts. In the first part, we compare our method to Pix2Pix, and in the second part to BicycleGAN. Users were shown one facade generated with FrankenGAN and one facade with the other method, trained end-to-end to transform facade masks to facade textures. Each pair was evaluated by an average of 17.6 and 17.5 users, for Pix2Pix and BicycleGAN, respectively, and there were 86 unique participants in the study. Results are shown in Figure 19 (red vs. blue and yellow vs. blue). FrankenGAN was chosen as more realistic in 66.6% of the comparisons with Pix2Pix and 57.6% of the comparisons for BicycleGAN. 95% confidence intervals are shown as small bars in Figure 19. Note that this advantage in realism is in addition to the advantage of of having fine-grained style and scale control and obtaining label maps as intermediate results that are necessary to generate 3D details.

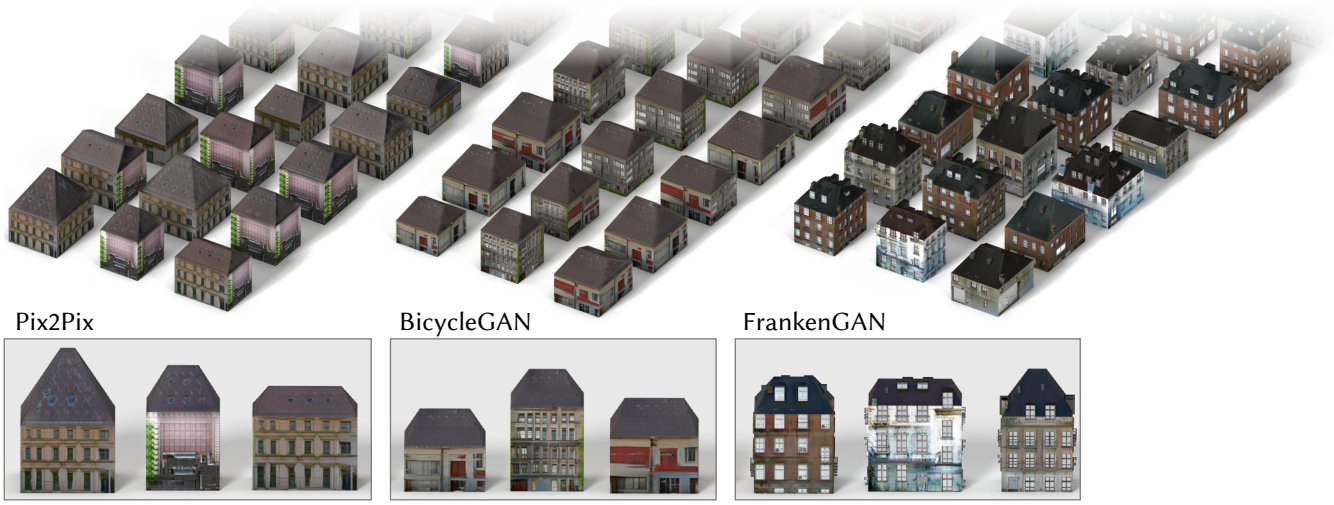

Fig. 18. Qualitative comparison with end-to-end GANs. The left column shows results of Pix2Pix trained to transform empty facade and roof masks to textures. The middle column shows BicycleGAN trained similarly, while the last column shows our method. Note how Pix2Pix suffers from mode collapse, while BicyleGAN has less realistic window layouts and lacks scale and style consistency. FrankenGAN provides better style control and our approach of splitting up the problem into multiple steps opens up several avenues to increase the realism of our models.

6.5 Limitations

There are also some limitations to our framework. To replicate the framework from scratch requires extensive training data. Similar to other GANs, our results look very good from a certain range of distances, but there is a limit to how close it is possible to zoom in before noticing a lack of details or blurriness. This would require the synthesis of displacement maps and additional material layers in addition to textures. Data-driven texturing is inherently dependent on representative datasets. For example, our facade label dataset had missing windows due to occlusions by trees and other flora. We thus occasionally see missing windows in our results. We chose not to fill in windows in the regularisation stage. The facade-texturing network then learned to associate these missing windows with flora, and dutifully added green "ivy" to the building walls (Figure 20, left). Finally, our system uses a shared style to synchronize the appearance of adjacent building facades. However, this compact representation does not contain sufficient detail to guarantee seamless textures at boundaries (Figure 20, right).

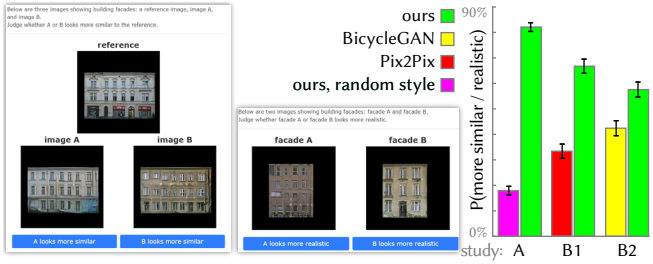

Fig. 19. Perceptual studies comparing our method with/without style guidance (left) and to Pix2Pix [Isola et al. 2017] and BicycleGAN [Zhu et al. 2017b] (middle). The average probability for each method of being judged more similar to the reference or more realistic by the study participants is shown on the right. The black bars are 95% confidence intervals.

7 CONCLUSION

We presented FrankenGAN, a system for adding realistic geometric and texture details on large-scale coarse building mass models guided by example images for style. Our system is based on a cascaded set of GANs that are individually controlled by generator vectors for style synchronization. We evaluated our system on a range of large-scale urban mass models and qualitatively evaluated the quality of the detailed models via a set of user studies.

The most direct way to improve the quality of output generated by FrankenGAN is to retrain the GANs using richer and more accurately annotated facade and roof datasets and also to extend the system to generate street textures and furniture, as well as trees and botanical elements. This is a natural improvement as many research efforts are now focused on capturing high quality facade data with improved quality annotations. A more non-trivial extension is to explore a GAN-based approach that directly generates textured 3D geometry. The challenge here is to find a way to compare rendered procedural models with images based on photographs that also includes facade structure rather than just texture details. Finally, it would be interesting to combine our proposed

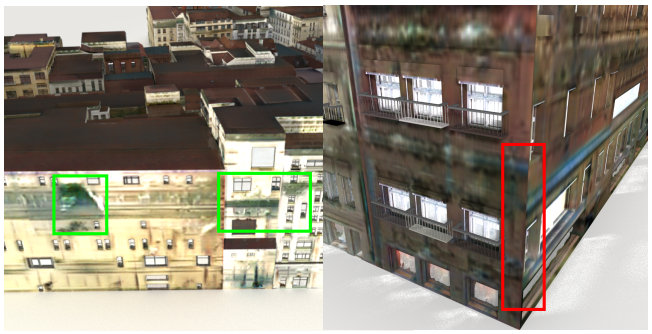

Fig. 20. Limitations. Green: the system is prone to generating green vegetation over missing windows. Red: texture discontinuities at boundaries.

approach to author both realistic building mass models along with their detailing directly from rough input from the user in the form of sketches [Nishida et al. 2016] or high-level functional specifications [Peng et al. 2016].

ACKNOWLEDGMENTS

This project was supported by an ERC Starting Grant (SmartGeometry StG-2013-335373), KAUST-UCL Grant (OSR-2015-CCF-2533), ERC PoC Grant (SemanticCity), the KAUST Office of Sponsored Research (OSR-CRG2017-3426), Open3D Project (EPSRC Grant EP/M013685/1), and a Google Faculty Award (UrbanPlan).

REFERENCES

Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein GAN. (2017). arXiv:1701.07875

Fan Bao, Michael Schwarz, and Peter Wonka. 2013. Procedural Facade Variations from a Single Layout. *ACM TOG* 32, 1 (2013), 8.

M. Berger, J. Li, and J. A. Levine. 2018. A Generative Model for Volume Rendering. *IEEE TVCG* (2018), 1–1.

David Berthelot, Tom Schumm, and Luke Metz. 2017. BEGAN: Boundary Equilibrium Generative Adversarial Networks. (2017). arXiv:1703.10717

Martin Bokeloh, Michael Wand, Hans-Peter Seidel, and Vladlen Koltun. 2012. An Algebraic Model for Parameterized Shape Editing. *ACM TOG* 31, 4 (2012).

A. Cohen, A. G. Schwing, and M. Pollefeys. 2014. Efficient Structured Parsing of Facades Using Dynamic Programming. In *IEEE PAMI*. 3206–3213.

Minh Dang, Duygu Ceylan, Boris Neubert, and Mark Pauly. 2014. SAFE: Structure-aware Facade Editing. *CGF* 33, 2 (2014), 83–93.

Emily L Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. 2015. Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks. In *NIPS*. 1486–1494.

Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. 2016. Adversarial Feature Learning. In *ICLR*.

Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Alex Lamb, Martin Arjovsky, Olivier Mastropietro, and Aaron Courville. 2016. Adversarially Learned Inference. In *ICLR*.

Victor Escorcia, Fabian Caba Heilbron, Juan Carlos Niebles, and Bernard Ghanem. 2016. Daps: Deep action proposals for action understanding. In *ECCV*. Springer, 768–784.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Nets. In *NIPS*. 2672–2680.

Swaminathan Gurumurthy, Ravi Kiran Sarvadevabhatla, and R. Venkatesh Babu. 2017. DeLiGAN : Generative Adversarial Networks for Diverse and Limited Data. In *CVPR*.

Stefan Hartmann, Michael Weinmann, Raoul Wessel, and Reinhard Klein. 2017. StreetGAN: Towards Road Network Synthesis with Generative Adversarial Networks. *ICCV*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *CVPR*. 770–778.

Martin Ilčík, Przemyslaw Musialski, Thomas Auzinger, and Michael Wimmer. 2015. Layer-Based Procedural Design of Façades. *CGF* 34, 2 (2015), 205–216.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. 2017. Image-to-Image Translation with Conditional Adversarial Networks. *CVPR* (2017).

Justin Johnson, Alexandre Alahi, and Fei fei Li. 2016. Perceptual Losses for Real-Time Style Transfer and Super-Resolution. In *ECCV*.

Angel X. Chang Kai Wang, Manolis Savva and Daniel Ritchie. 2018. Deep Convolutional Priors for Indoor Scene Synthesis. *ACM TOG* (2018).

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. 2018. Progressive Growing of GANs for Improved Quality, Stability, and Variation. In *ICLR*.

Tom Kelly, John Femiani, Peter Wonka, and Niloy J. Mitra. 2017. BigSUR: Large-scale Structured Urban Reconstruction. *ACM SIGGRAPH Asia* 36, 6, Article 204 (2017).

D.P. Kingma and M. Welling. 2014. Auto-Encoding Variational Bayes. In *ICLR*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*. 1097–1105.

Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. 2016. Autoencoding beyond pixels using a learned similarity metric. In *ICML*, Vol. 48.

Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. 2017. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. In *CVPR*.

Jinjie Lin, Daniel Cohen-Or, Hao Zhang, Cheng Liang, Andrei Sharf, Oliver Deussen, and Baoquan Chen. 2011. Structure-Preserving Retargeting of Irregular 3D Architecture. *ACM TOG* 30, 6 (2011), 183:1–183:10. https://doi.org/10.1145/2070781.2024217

Jonathan Long, Evan Shelhamer, and Trevor Darrell. 2015. Fully convolutional networks for semantic segmentation. In *CVPR*. 3431–3440.

Andjelo Martinovic and Luc Van Gool. 2013. Bayesian Grammar Learning for Inverse Procedural Modeling. In *CVPR*. 201–208.

Markus Mathias, Andelo Martinović, and Luc Gool. 2016. ATLAS: A Three-Layered Approach to Facade Parsing. *IJCV* 118, 1 (May 2016), 22–48.

Pascal Mueller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. 2006. Procedural Modeling of Buildings. *ACM TOG* 25, 3 (2006), 614–623.

Gen Nishida, Ignacio Garcia-Dorado, Daniel G. Aliaga, Bedrich Benes, and Adrien Bousseau. 2016. Interactive Sketching of Urban Procedural Models. *ACM TOG* (2016).

Chi-Han Peng, Yong-Liang Yang, Fan Bao, Daniel Fink, Dong-Ming Yan, Peter Wonka, and Niloy J Mitra. 2016. Computational Network Design from Functional Specifications. *ACM TOG* 35, 4 (2016), 131.

Daniel Ritchie, Ben Mildenhall, Noah D. Goodman, and Pat Hanrahan. 2015. Controlling Procedural Modeling Programs with Stochastically-Ordered Sequential Monte Carlo. *ACM TOG* 34, 4 (2015), 105:1–105:11. https://doi.org/10.1145/2766895

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, Xi Chen, and Xi Chen. 2016. Improved Techniques for Training GANs. In *NIPS*. 2234–2242.

Michael Schwarz and Pascal Müller. 2015. Advanced Procedural Modeling of Architecture. *ACM TOG* 34, 4 (2015), 107:1–107:12. https://doi.org/10.1145/2766956

Jost T. Springenberg. 2016. Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Network. In *ICLR*.

Andreas Stolcke and Stephen Omohundro. 1994. Inducing Probabilistic Grammars by Bayesian Model Merging. In *Proceedings of ICGI-94*. 106–118.

Jerry O. Talton, Yu Lou, Steve Lesser, Jared Duke, Radomír Měch, and Vladlen Koltun. 2011. Metropolis Procedural Modeling. *ACM TOG* 30, 2 (2011), 11:1–11:14.

Jerry O. Talton, Lingfeng Yang, Ranjitha Kumar, Maxine Lim, Noah D. Goodman, and Radomír Měch. 2012. Learning Design Patterns with Bayesian Grammar Induction. In *Proceedings of UIST '12*. 63–74.

O. Teboul, I. Kokkinos, L. Simon, P. Koutsourakis, and N. Paragios. 2013. Parsing Facades with Shape Grammars and Reinforcement Learning. *IEEE PAMI* 35, 7 (July 2013).

Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. 2015. Learning spatiotemporal features with 3d convolutional networks. In *ICCV*.

Luan Tran, Xi Yin, and Xiaoming Liu. 2017. Disentangled Representation Learning GAN for Pose-Invariant Face Recognition. In *CVPR*.

Radim Tyleček and Radim Šára. 2013. Spatial Pattern Templates for Recognition of Objects with Regular Structure. In *Proc. GCPR*. Saarbrucken, Germany.

Carlos A. Vanegas, Ignacio Garcia-Dorado, Daniel G. Aliaga, Bedrich Benes, and Paul Waddell. 2012. Inverse Design of Urban Procedural Models. *ACM TOG* 31, 6 (2012), 168:1–168:11. https://doi.org/10.1145/2366145.2366187

V. S. R. Veeravasarapu, Constantin A. Rothkopf, and Visvanathan Ramesh. 2017. Adversarially Tuned Scene Generation. (2017).

Chengde Wan, Thomas Probst, Luc Van Gool, and Angela Yao. 2017. Crossing Nets: Combining GANs and VAEs With a Shared Latent Space for Hand Pose Estimation. In *CVPR*.

David Warde-Farley and Yoshua Bengio. 2017. Improving Generative Adverarial Networks with Denoising Feature Matching. In *CVPR*.

Peter Wonka, Michael Wimmer, François Sillion, and William Ribarsky. 2003. Instant Architecture. *ACM TOG* 22, 3 (2003), 669–677.

Raymond Yeh, Chen Chen, Teck-Yian Lim, Mark Hasegawa-Johnson, and Minh N. Do. 2016. Semantic Image Inpainting with Perceptual and Contextual Losses. (2016). arXiv:1607.07539

Yi-Ting Yeh, Katherine Breeden, Lingfeng Yang, Matthew Fisher, and Pat Hanrahan. 2013. Synthesis of Tiled Patterns using Factor Graphs. *ACM TOG* 32, 1 (2013).

Junbo Zhao, Michael Mathieu, and Yann LeCun. 2017. Energy-based generative adversarial network. In *ICLR*.

Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A. Efros. 2016. Generative Visual Manipulation on the Natural Image Manifold. In *ECCV*.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. 2017a. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. *ICCV* (2017).

Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A Efros, Oliver Wang, and Eli Shechtman. 2017b. Toward Multimodal Image-to-Image Translation. In *NIPS*.