Projective Urban Texturing

Yiangos Georgiou[1,2] Melinos Averkiou[1,2] Tom Kelly[3] Evangelos Kalogerakis[4]
University of Cyprus[1] CYENS CoE[2] University of Leeds[3] UMass Amherst[4]

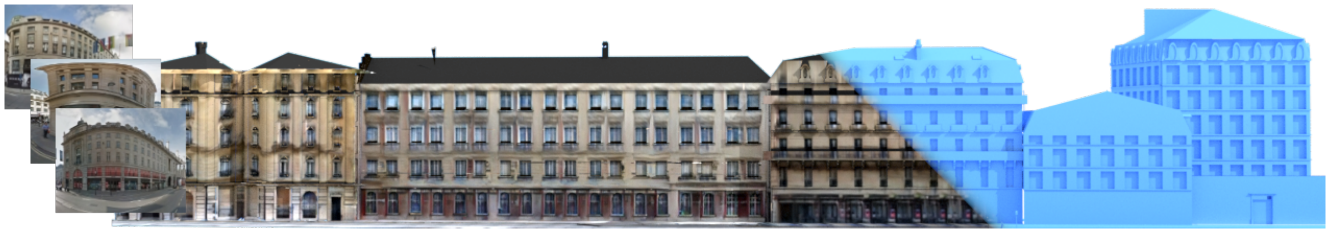


Figure 1: Our method takes as input a set of panoramic images (inset, left) representative of an urban environment (e.g., a London neighborhood) along with 3D untextured polygon meshes (blue, right), and it outputs textures for the meshes (centre) with a similar style to the input panoramic images.

Abstract

This paper proposes a method for automatic generation of textures for 3D city meshes in immersive urban environments. Many recent pipelines capture or synthesize large quantities of city geometry using scanners or procedural modeling pipelines. Such geometry is intricate and realistic, however the generation of photo-realistic textures for such large scenes remains a problem. We propose to generate textures for input target 3D meshes driven by the textural style present in readily available datasets of panoramic photos capturing urban environments. Re-targeting such 2D datasets to 3D geometry is challenging because the underlying shape, size, and layout of the urban structures in the photos do not correspond to the ones in the target meshes. Photos also often have objects (e.g., trees, vehicles) that may not even be present in the target geometry. To address these issues we present a method, called Projective Urban Texturing (PUT), which re-targets textural style from real-world panoramic images to unseen urban meshes. PUT relies on contrastive and adversarial training of a neural architecture designed for unpaired image-to-texture translation. The generated textures are stored in a texture atlas applied to the target 3D mesh geometry. To promote texture consistency, PUT employs an iterative procedure in which texture synthesis is conditioned on previously generated, adjacent textures. We demonstrate both quantitative and qualitative evaluation of the generated textures.

1. Introduction

Authoring realistic 3D urban environments is integral to city planning, navigation, story-telling, and real-estate. Although 3D meshes of cities with thousands or millions of polygons can be synthesized through numerous procedural modeling pipelines [41, 37, 30, 45, 12], they often lack texture information. To create vibrant and realistic virtual scenes, these meshes must be textured. Textures also provide many of the cues that we use to identify locations and purpose of buildings, including material and color. 3D meshes of cities can alternatively be reconstructed through mesh acquisition pipelines. However, texture generation for the acquired meshes may still be desirable especially when the captured textures have low quality and resolution (e.g., ones from aerial photogrammetric reconstruction), or when a different textural style is desired. Manually creating high-quality textures is time consuming – not only must the details of the textures match the geometry, but also the style within single structures, as well as between adjacent structures and their surroundings (e.g., between buildings, street-furniture, or sidewalks). An alternative is to programmatically generate textures (e.g., using shape grammars [38]), an approach which can create repeated features (e.g., windows, pillars, or lampposts) common in urban landscapes, but is time consuming to write additional such rules. This approach is often challenged by lack of texture variance (e.g., due to discoloration and aging of buildings).

We present Projective Urban Texturing (PUT), a method to create street-level textures for input city meshes. Following recent texture generation work, we use a convolutional neural network with adversarial losses to generate textures. However, the resolution of existing approaches is limited by available memory and they often result in numerous artifacts, as naïvely tiling network outputs creates discontinuities (seams) between adjacent tiles (see Figure 3). PUT iteratively textures parts of a 3D city mesh by translating panoramic street-level images of real cities using a neural network, and merges the results into a single high-resolution texture atlas. By conditioning the network outputs on previous iterations, we are able to reduce seams and promote consistent style with prior iterations.

A central decision when developing texturing systems is the domain to learn from. Learning to create textures in 3D object-space (e.g., over facades) [26, 29] can exploit semantic information. However, object-object interactions (e.g., between adjacent buildings or streets and buildings) are not modelled, mesh semantic information is rarely available, and mesh training data is rather sparse. In contrast, PUT learns in panoramic image-space, texturing multiple objects simultaneously and modeling their interactions consistently.

A particular advantage of panoramic image datasets is that they contain massive amounts of urban image data to train with. Advances in optics have given us low-cost commodity 360 degree panoramic cameras which capture omni-directional information quickly, while efforts such as Google's Street View have resulted in large panorama datasets captured in different cities [1]. These images are typically street-level – a single panorama captures multiple objects such opposing facades, overhanging features (e.g., roof eaves), and the street itself.

On the other hand, there are several challenges in retargeting these panoramic image datasets to textures of city meshes. Since panoramic images are captured from the real world, buildings are often occluded (e.g., by trees or vehicles), and contain camera artifacts (e.g., lens flare). Further, these image datasets do not contain corresponding 3D geometry. In fact, the underlying shape, size, and layout of the urban structures in the images can be quite different from the ones in the target meshes. Another challenge is to generate a single, consistent output texture atlas by training on an unorganized collection of images. PUT addresses these challenges by employing contrastive and adversarial learning for unpaired image-to-texture translation along with a texture consistency loss function. The generated textures share a similar style with the one in the training images e.g., training on paroramas from London neighborhoods yields London-like textures for target meshes (Figures 1, 5).

PUT offers the following contributions: i) an architecture that learns to generate textures over large urban scenes conditioned on meshes and previous results to promote texture consistency, ii) unpaired style transfer from panoramic image datasets to texture maps in the form of texture atlases.

2. Related Work

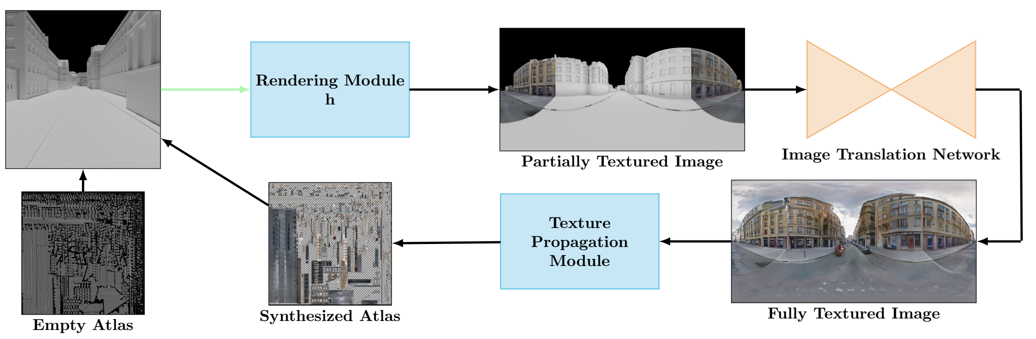
PUT is related to methods for 2D texture synthesis and texture generation for 3D objects as briefly discussed below.

Texture synthesis. Traditional learning-based methods generate textures with local and stationary properties [11, 49, 33, 10, 32]. Later work creates larger, more varied textures quickly [9], some of which are domain specific [8]. Procedural languages can create homogeneous textures for urban domains [38]; however, they lack realism since they do not model the heterogeneous appearance of surfaces due to weathering and damage. Such phenomena can be created using simulation or grammars [4, 5, 20, 34], yet these approaches require significant hand-engineering. With the advent of convnets (CNNs), data-driven texture synthesis has been achieved by optimization in feature space [16] or with GANs [53, 15]. Image-to-image translation with U-Nets [44, 26] has been demonstrated for style transfer [17]; these have been applied to various domains [18, 29, 52]. Processing panoramic images, such as street-view images, with CNNs has been explored for depth prediction [46] and image completion [47]. Approaches for unpaired image translation have also been developed using cycle consistency losses [54], which learn to translate from one image domain to another, and back again. These techniques has been extended to multi-domain [6, 25] and domain-specific problems [19, 48]. A more recent approach uses patch-wise contrastive learning [42]. We adopt this approach as our backbone and incorporate it within our iterative pipeline that conditions synthesis on input 3D meshes and previous texture results, along with a novel texture consistency loss.

Texturing 3D objects. Early approaches directly project photos onto reconstructed geometry [39, 2, 3]. This produces photorealistic results, but requires exact photo locations with consistent geometry (i.e., noise-free location data [31]) and unobstructed views (i.e., no occlusions by trees or vehicles [13]). 3D CNNs have been proposed to generate voxel colours [40] as well as geometry [51, 50], however such have high memory requirements and there are sparse training 3D object datasets with realistic textures. Attempts to overcome these limitations are ongoing, using a variety of approaches [28, 21, 23, 35, 43, 22]; results are generally low resolution or only in screen-space. GANs have been previously proposed to align and optimize a set of images for a scanned object [24] producing high quality results; however, they require images of a specific object. In our case, we leverage large 2D panoramic image datasets for generating textures for urban models without paired image input. A key difference over prior work is that our method learns textures by combining contrastive learning-based image synthesis [42] with a novel blending approach to wrap the texture over the geometry.

3. Method

Overview. Given untextured 3D geometry representing an urban environment as input, our pipeline (Figure 2) synthesizes a texture for it. We assume that the surface of the input 3D geometry is unwrapped (parameterized), such that UV coordinates assign 3D surface points to 2D locations



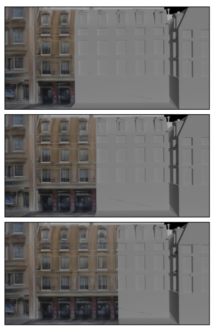


Figure 2: At each iteration our system creates a *partially textured image* by rendering the 3D geometry(top left) using the *synthesized atlas* containing texture from previous iterations. Mesh parts that have not been textured so far are rendered in grayscale. The *texture propagation module* uses the generated image to update the texture atlas used in the next iteration. A visualization of our iterative texturing synthesis output is displayed on a characteristic building facade on the right.

within a texture atlas (a 2D image map; Figure 2, bottom left). The input urban geometry may come from a variety of sources, such as procedural modeling (Figure 1, right) or photogrammetric reconstruction (Figure 7, right).

Our texture synthesis system is iterative, working sequentially with selected viewpoints along street centerlines: to begin each iteration, our *rendering module* creates a panoramic image representing the input 3D geometry from a viewpoint, including any previously generated texture (Figure 2). The rendered images are *partially textured*, since some parts of the 3D urban models may have associated texture synthesized from previous iterations, while others may not contain any texture at all. Untextured parts are rendered in grayscale, and textured parts in colour.

As each iteration continues, the partially textured image is translated into a fully textured RGB image through a neural network (Figure 2, *image translation* network). Our experiments found that passing the partially textured images through the network offered better performance, compared to passing completely untextured (grayscale) images, or using separate images, or channels, for the textured and untextured parts. The network is trained on a set of panoramic RGB images taken from streets of a particular city (e.g., London or New York) and a set of rendered images of urban geometry meshes. The two sets are unpaired; we do not assume that the urban geometry meshes have corresponding, or reference, textures. Such a level of supervision would require enormous manual effort to create training textures for each 3D urban model. Our system is instead trained to automatically translate the domain, or geographic "style", of the panoramic images of real-world buildings into textures that match the geometry "structure" of the input 3D urban models, and any prior textured parts.

At the end of the iteration, our *texture propagation module* updates the texture atlas using the output fully textured image from the network (Figure 2, *synthesized atlas*). After the final iteration, all viewpoints have been processed, the texture atlas is complete, and is used to render the urban environment, as shown in Figure 5.

Viewpoint selection. In a pre-processing step, we create paths along which we place viewpoints to sequentially capture the input geometry of the city blocks. The paths are formed by following the centerlines of all streets contained in the urban 3D scene (see Section 4). The paths trace the street from the start to the end, "sweeping" the buildings in each block on both sides of the road. Each viewpoint is placed at 5m horizontal intervals along the street centerlines with a vertical height of 2.5m. The horizontal intervals are empirically selected such that each rendered image has an overlap (about 25%) with the image rendered from the previous and next viewpoint. The vertical height is motivated by the fact that real-world car mounted cameras are placed at a height close to 2.5m, reducing the "domain gap" between our renderings and real-world panoramas taken from cars. The viewpoint (camera) up and forward axes are set such that they are aligned with the world upright axis and street direction respectively. As a result, the cameras "look" towards the buildings along both sides of the road.

Rendering. Given each viewpoint, each rendering is produced through equirectangular (panoramic) projection. As a result, the domain gap with real-world panoramas is decreased with respect to the projection type. The geometry is rendered with global illumination using Blender [7] at a 512x256 resolution. Any textured parts from previous passes incorporate RGB color information from the texture atlas, while for untextured parts, a white material is used, resulting in a grayscale appearance. Background (e.g., sky) is rendered as black. An example of the resulting "partially textured" image is shown in Figure 2.

Neural network. The input to our image translation network is a partially textured rendering (3x512x256) at each iteration. The neural network uses the architecture from Johnson et al. [27], also used in Contrastive Unpaired Translation (CUT) [42]. The network contains three convolution layers, 9 residual blocks, two fractionally-strided convolution layers with stride $1/2$, and a final convolution layer that maps the features to a RGB image (3x512x256). Since our input is a panoramic image, we use equirectangular convolution for the first convolutional layer of the network [14]. Equirectangular convolution is better suited for processing panoramic images and offers better performance in our experiments. As discussed in Section 4, we follow a training procedure and loss inspired by [42], yet with important differences to handle partially textured images and ensure consistency in the texture generated from different viewpoints.

Texture propagation. At each iteration, we use the generated image from the above network to update the texture atlas. For each texel t with center coordinates (u_t, v_t) in the atlas, we find the corresponding 3D point $\mathbf{p}_t$ on the input geometry. Then for this 3D point, we find its corresponding pixel location in the generated image at the current iteration i: $[x_{t,i},\, y_{t,i}] = \Pi_i(\mathbf{p_t})$, where Π_i is the equirectangular projection function used during the frame rendering of the current iteration. The color of the texel is transferred with the mapping: $\mathit{color}[u_t, v_t] = \mathbf{R}_i[x_{t,i},\, y_{t,i}]$, where $\mathbf{R}_i$ is the generated image from the network at the current iteration. However, this strategy can create artifacts since it will overwrite the color of texels that were updated from previous iterations, resulting in sharp discontinuities in the generated texture (Figure 3, left).

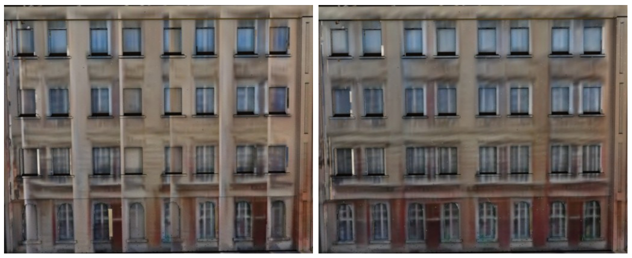


Figure 3: Texture w/o blending (left), w/ blending (right)

Instead, we follow a pooling strategy, where colors for each texel are aggregated, or blended, from different iterations. Specifically, at each iteration, the color of each texel is determined as a weighted average of pixel colors originating from images generated from previous iterations:

$$\mathit{color}[u_t, v_t] = \sum_{i \in V_t} w_{i,t} \mathbf{R}_i[x_{t,i},\, y_{t,i}] \tag{1}$$

where V_t is the set of iterations where the texel's corresponding 3D point $\mathbf{p}_t$ was accessible (visible) from the cameras associated with these iterations. The blending weights $w_{i,t}$ are determined by how close $\mathbf{p}_t$ was to the center of the projection at each iteration. The closer to the centre the point $\mathbf{p}_t$ was, the higher the weight was set for the color of its corresponding pixel at the generated image. Specifically, if $d_{i,t}$ is the distance of the point $\mathbf{p}_t$ to the projection centerline for the camera at iteration i, the weight of its corresponding pixel color from the generated image at this iteration was determined as $w_{i,t} = 1 - d_{i,t} / \sum_{i'} d_{i',t}$, where i' are iteration indices where the point was visible. The weights were further clamped to $[0.3, 0.7]$, then were re-normalized between [0, 1] to eliminate contributions from cameras located too far away from the point.

4. Training

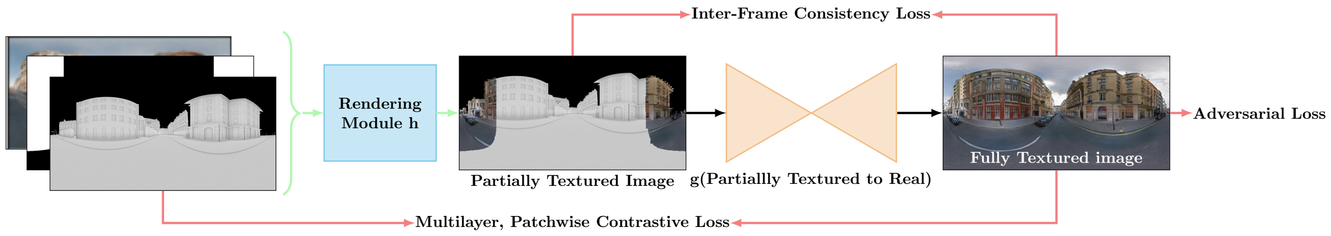
To train the image translation neural network of our pipeline, we require a set of training examples from both domains: rendered panorama images from the (untextured) 3D geometry, and real-world panoramic photographs. The two sets are unpaired, i.e., the real-world images have no geometric or semantic correspondences with rendered panoramas. We discuss the datasets used for these domains in the following paragraphs, then explain our training procedure.

Real-world panorama dataset. For real-world panoramas, we use the dataset from [36], with $29,414$ images from *London* and $6,300$ images from *New York*; these photos are taken from city streets with vehicle mounted cameras.

Rendered panoramas from 3D geometry. We demonstrate our system on two classes of 3D urban meshes - procedural and photogrammetrically reconstructed meshes. Procedural meshes are generated using the CGA language [37]; each generated scene has unique street graph, block-subdivision, and building parameters (e.g., style, feature size, height etc.). We found that the inclusion of details such as pedestrians, trees, and cars helped the multi-layer patch-wise contrastive loss used in our network to identify meaningful correspondences between the real and rendered panorama domains. We also export the street center-line data from the procedural models (no other "semantic" information or labels are used). We generated 10 scenes with 1600 viewpoints sampled from 18 random paths on streets of each scene. This resulted in $16K$ viewpoints in total, from which we rendered grayscale images of the input geometry using equirectangular projection. We use 9 scenes ($14.4K$ grayscale, rendered panoramic images) for training, and keep one scene for testing ($1.6K$ images). We note that the streets, building arrangements, and their geometry are unique for each street and scene, thus, the test synthetic renderings were different from the training ones.

The photogrammetric mesh dataset is from Google Earth. The 3D meshes are coarse and noisy (Fig. 7, right, blue mesh). They contain holes, vehicles, and trees. This contrasts the precise, high-detail, data in procedural meshes. Street centerlines are collected from OpenStreetMap. We use 2 scenes and $1,411$ rendered images. One is used for training and the other for testing.

Training Procedure. As mentioned above, since there are no target reference texture images for the input 3D geometry, we resort to a training procedure that uses the unpaired real-world RGB panoramas and our partially textured panorama renderings. Contrastive Unpaired Translation (CUT) [42] is a fast and flexible method to train im-



Inter-Frame Consistency Loss
Rendering Module h
Partially Textured Image
g(Partiallly Textured to Real)
Fully Textured image
Adversarial Loss
Multilayer, Patchwise Contrastive Loss

Figure 4: Training procedure: the green arrow shows the mapping h from a grayscale rendering of a training city block to the partially textured image. This mapping is performed in our rendering module that integrates previous texture information with the help of a binary mask indicating prior textured regions. The image translation network g maps the partially textured image to a fully textured one, mimicking real-world statistics with the help of an adversarial loss. The partially-textured image is compared to the fully textured image in the inter-frame consistency loss to discourage seams in the output texture.

age translation networks between unpaired domains. One potential strategy is to use CUT's losses and training as-is in our setting. However, this strategy disregards the need for consistency between generated textures at different iterations of our pipeline. Further, our image translation network processes partially textured images, which are different from both the domain of real-world panoramas and the domain of grayscale rendered images. Next, we explain how our training procedure promotes consistency and handles the different kinds of inputs required for our network.

Multi-layer Patch-wise Contrastive Loss. The goal of our training procedure is to learn the parameters of our image translation network $g : P \rightarrow R$ mapping the domain P of partially textured images to the domain R of fully textured ones mimicking real-world statistics. To associate input and output structure we use a multi-layer patch-wise contrastive loss between input and output. The association between images is achieved by the comparison of the stack of features for selected layers in the encoder network f.

Specifically, in our setting, given an untextured (grayscale) rendering S from our set of untextured renderings $\mathbf{S}$, we first convert it to a partially textured one by integrating previously textured parts indicated from a binary mask M (1 for previously textured parts, 0 for untextured ones). This conversion $h : S \rightarrow P$ is a fixed, parameter-free mapping implemented in our rendering module. Given a rendered image S and the generated image $g(h(S))$, we generate their stack of features $z = f(S)$ and $\hat{z} = f(g(h(S)))$. Pairs of patches from the rendered and generated images at the same location n are treated as positive, while pairs from different locations are treated as negative. A temperature-scaled cross-entropy loss H (with temperature $\tau = 0.07$) computes the probability of the positive example being selected over negatives [42]. The loss is summed over our dataset and samples:

$$\mathcal{L}_{contr_S} = \mathbb{E}_{S \sim \mathbf{S}} \sum_{l=1}^{L} \sum_{n=1}^{N_l} H(\hat{z_l^n}, z_l^n, z_l^{N_l/n}) \qquad (2)$$

where L is the number of selected layers and N_l is the index set of spatial locations in feature maps at each layer. As in CUT [42], to avoid unnecessary generator changes, a cross-entropy loss $\mathcal{L}_{contr_R}$ is also applied on patches sampled from the real images domain $\mathcal{R}$.

Adversarial losses. Apart from the contrastive losses, we use an adversarial loss to ensure that the fully textured images generated from the image translation network g share similar statistics with the real-world panoramas:

$$\begin{aligned}\mathcal{L}_{gan} &= \mathbb{E}_{R \sim \mathbf{R}}[log(d_r(R))] \\ &\quad + \mathbb{E}_{S \sim \mathbf{S}}[1 - log(d_r(g(h(S)))]\end{aligned}$$

where $\mathbf{R}$ is the training set of real-world panoramas, d_r is a discriminator network following the architecture of CUT applied to real images domain.

Inter-frame consistency loss. We introduce an inter-frame consistency loss to promote consistency in the generated textured images from our image translation network g across different iterations ("frames") of our pipeline. Given a fully textured image $g(h(S))$ generated from the image translation network during training, and a partially textured image $h(S)$ containing texture from prior iterations, we compare the textured regions in $h(S)$ with the corresponding generated regions. The comparison is performed with the help of the binary mask M containing 1s for the partially textured regions of $h(S)$ and 0s otherwise. Using the above-mentioned mappings, the loss is expressed as follows:

$$\mathcal{L}_{cons} = \mathbb{E}_{S \sim \mathbf{S}}[\| \ (g(h(S))) \odot M - h(S) \odot M \ \|_1] \qquad (3)$$

where $\odot$ is the Hadamard product. We discuss the effect of this loss in our experiments and ablation study.

Full Objective. The full objective is a weighted combination of the above losses:

$$\mathcal{L} = \lambda_1 \mathcal{L}_{gan} + \lambda_2 \mathcal{L}_{cons} + \lambda_3 \mathcal{L}_{contr_S} + \lambda_4 \mathcal{L}_{contr_R} \qquad (4)$$

where $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are hyper-parameters set to $1.0, 10.0, 0.5, 0.5$ respectively.

Implementation details. We use the Adam optimizer with learning rate $2 \cdot 10^{-4}$ to train the above architecture. We train the model for 200 epochs. During the first 150 epochs the learning rate is fixed, then it gradually decays for the last 50 epochs. All our code and datasets, including the rule-set and scripts for the procedural generation of the training and test scenes are available on our project page `https://ygeorg01.github.io/PUT/`.

5. Evaluation

We now evaluate our pipeline on test models generated with the process described in Section 4. As discussed earlier, no identical city blocks exist in our train and test meshes. We trained our translation network separately on the images datasets of *London* and *New York*.

Qualitative Evaluation. First, we evaluate the quality of our output textures by placing perspective cameras at the street level for each of our test meshes and rendering the meshes with texture atlases created by our method. Results can be seen in Figure 5, for the network trained on London data. Our method transfers the appearance of large structures such as multi-storey walls, streets, or sidewalks from real panoramas and often aligns texture details with geometric ones i.e., texture edges of windows and doors with geometry edges. A side-effect of our method is that shadows are sometimes translated to streets and sidewalks (Figure 5).

City style transfer. Our method can be trained on street-level panoramic images collected from different cities in the world to give urban 3D models the appearance of a city's distinctive style, as illustrated by our London and New York examples in Figure 6. The generated textured city blocks appear distinctively textured with our pipeline trained on New York versus the ones textured with London panoramas. For example, a red brick-like appearance is visible in the walls of the meshes textured by our network trained in New York panoramas; such appearance is common in New York buildings. In contrast, meshes textured by our network trained on London panoramas appear with white, yellow and brown distributions of color on their facades, a color distribution which is common around London.

Mesh type generalization. We also demonstrate our pipeline on a city 3D model from Google Earth, shown in Figure 7. Our method generates consistent facade textures, even for such challenging polygon soups that are noisy and do not contain any facade or window geometric information, as shown in the untextured rendering of the same model (Figure 7, right). However, the absence of accurate facade geometry limits the diversity of texturing.

Inter-frame consistency. We qualitatively evaluate the degree of consistency between consecutive output textures from our architecture in Figure 8. We display six consecutive intermediate panoramic outputs of our image translation network trained on the London dataset. We note that each consecutive output tends to be consistent with the preceding images. This helps to preserve some details in the generated textures, as seen at the top of Figure 8, which shows a close-up rendering of the textured 3D mesh on the right side of the street.

Quantitative evaluation. We first compare PUT to the *CUT* network when generating texture atlases. We note that *CUT* does not condition its image translation network on the partially textured image and does not use our inter-frame consistency loss. Both networks were trained on the London dataset. For fair comparison, we use the same equirectangular convolution for the first convolution layer of both architectures. Table 1 shows FID scores for panoramic renderings produced by PUT and *CUT* for our test procedural scene. Since a large area of panoramic images consists of road and sky with little or no variation in texture, we also evaluate the FID on cropped versions of these renderings that isolate the facades from the roads and sky; we call this variant score as "crop FID". We additionally show the performance of PUT and *CUT* using three different blending approaches: *no blend* where no blending is applied between frames, *average* blending which takes the average RGB values between frames and our texture propagation (without weights), *weighted* uses the weighted averaging scheme of Equation 1. Our model outperforms *CUT* regardless of the blending approach. Note that, our texture propagation approach based on our weighted averaging scheme outperforms the no-blend or average-blend baselines.

We also compare our model to *FrankenGAN* [29] using the author-provided trained model. Note that blending cannot be used in FrankenGAN since it generates textures for 3D objects individually (e.g., facade, window, or roof). FrankenGAN has the additional benefit of semantic labels for these objects. In contrast, we do not assume that such labels are available for test meshes. As shown in Table 1, our approach still outperforms FrankenGAN.

	No-Blend↓		**Average**↓		**Weighted**↓	
Method	full	crop	full	crop	full	crop
CUT	131.6	110.2	135.2	113.4	121.7	101.1
FrankenGAN	174.3	135.8	-	-	-	-
PUT	**128.2**	**95.6**	**132.6**	**97.3**	**115.4**	**86.3**

Table 1: FID comparisons between PUT and alternatives.

In Figures 9 and 10 we show qualitative comparisons between PUT and *CUT*. Figure 9 (top) shows failure cases of *CUT*, i.e. undesired tree artifacts on the facades which PUT decreases (bottom) by using the inter-frame consistency loss. Moreover, the style of the facade between the first and the third panorama produced by *CUT* differs in Figure 10 (top) in contrast to PUT which generates more consistent results (bottom).

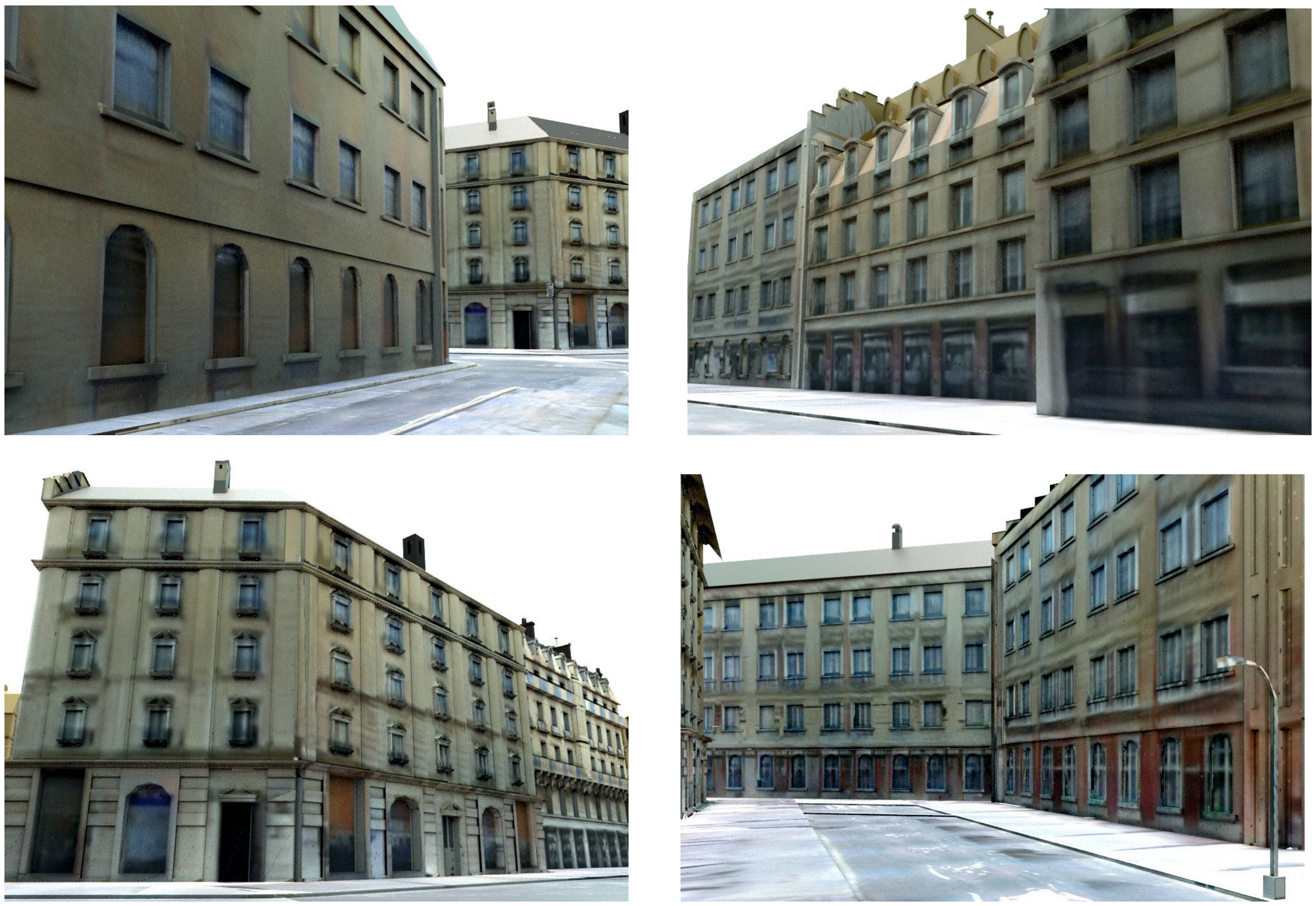


Figure 5: Renderings of our test scene textured by PUT trained on London panoramas. We show street-level renderings.

Finally, we performed a perceptual user study which showed that users preferred results generated from PUT much more compared to the results from *CUT*; see supplementary materials for additional details.

Ablation study. We also designed an ablation study to support the three main design choices for our network architecture, namely the use of the mask M in the inter-frame consistency loss (Equation 3), the use of equirectangular convolution for the first convolutional layer of the network [14], and the merging of the untextured (grayscale) and partially textured image to create 3-channel inputs.

Table 2 reports the "crop FID" scores for five variations of PUT produced by combination of the three design choices we made. Each row is a different model, incorporating different variations of the three choices. Comparing PUT 1 and PUT 2 we can see that our choice to merge untextured and partially textured images in a 3-channel input improves FID scores. Adding equirectangular convolution in the first convolution layer of PUT 3 slightly improves FID score, while incorporating the mask in the inter-frame consistency in PUT 4 gives a much larger improvement compared to PUT 2. Finally, using all three modifications results in the best FID score for the full model.

Model	Masked Cons.	Equir. Conv.	Gray+RGB	crop FID $\downarrow$
PUT 1	–	–	–	93.10
PUT 2	–	–	✓	92.47
PUT 3	–	✓	✓	92.23
PUT 4	✓	–	✓	87.56
full PUT	✓	✓	✓	**86.30**

Table 2: FID scores for PUT design choices. "Masked Cons." means whether we use the mask in the consistency loss, "Equir. Conv." means whether we use equirectangular convolution, "Gray+RGB" means whether we merge the grayscale rendering with the RGB partially textured image in a 3-channel image (or treat it as 4-channel image).

6. Limitations and Conclusion

We presented PUT, a method for texturing urban 3D scenes. Our method is able to texture large urban meshes by training on unpaired panoramic image and 3D geometry. One limitation of PUT is its inability to texture roofs; incorporating aerial images into our pipeline can help. Another limitation affecting the output texture resolution is that storing our large atlas is memory intensive. PUT can also be challenged by large domain gaps in the distributions of fea-

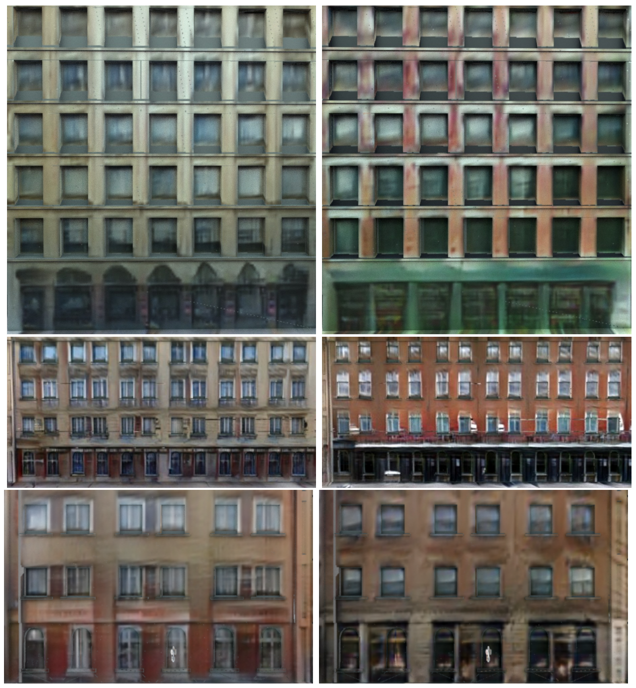


Figure 6: Street-level renderings of buildings in our test scene textured by our method trained on London images (left) and New York images (right).

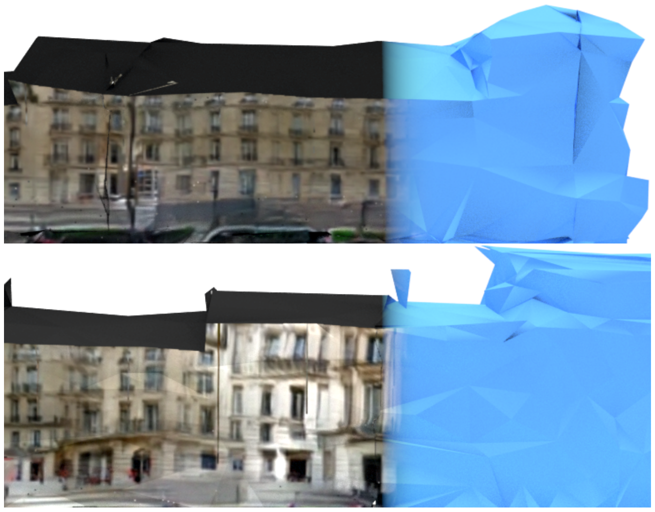


Figure 7: Our method can generate textures for challenging polygon meshes such as those from Google Earth. Here we show a street-level rendering of a Google Earth mesh textured by our method trained on London panoramas. The mesh geometry is noisy with non-planar facades lacking window and door cues (right). Still our method manages to create an approximate mesh texture with such cues.

tures between panoramic photos and geometry. A related issue is the reconstruction of artifacts due to lens aberrations and shadows in panoramas. In the future, we would like to allow users to explore the style-space of an urban environment by navigating a latent space. Another direction

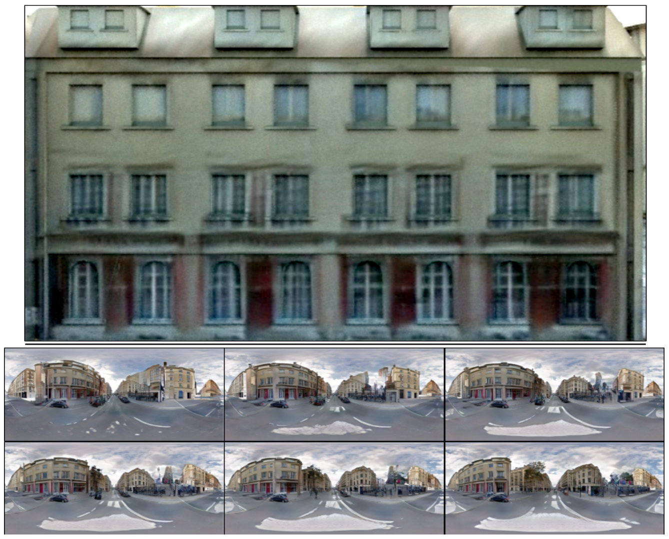


Figure 8: A sequence of six consecutive panoramic outputs of our network trained on London data (bottom). Notice how each consecutive output tends to be more consistent with the previous, which allows our texture propagation module to better preserve some details and better align features such as windows and doors on facade geometry (top).

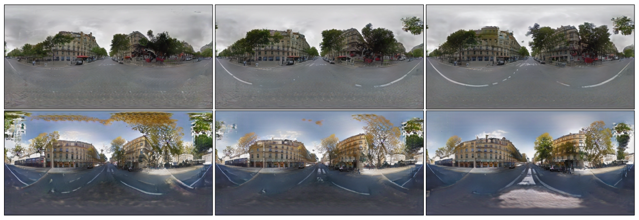


Figure 9: Tree artifacts appear on the facades in images generated from *CUT* (top row). Our inter-frame consistency loss reduces these artifacts (bottom row).

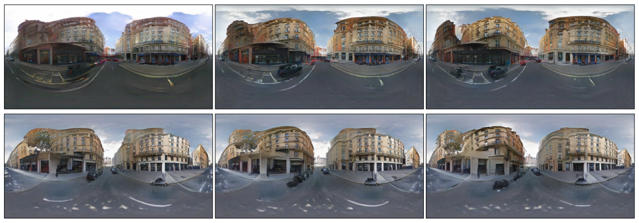


Figure 10: Style discontinuities occur between consecutive images generated from *CUT* (top row) in contrast with PUT which generates more consistent colors (bottom row).

is to increase our texture resolution to better portray fine details using super-resolution neural architectures.

Acknowledgements. Our work was funded by the ERDF and the Republic of Cyprus through the RIF (Project EXCELLENCE/1216/0352), the EU H2020 Research and Innovation Programme, and the Cyprus Deputy Ministry of Research, Innovation and Digital Policy (Grant 739578).

References

[1] Dragomir Anguelov, Carole Dulong, Daniel Filip, Christian Frueh, Stéphane Lafon, Richard Lyon, Abhijit Ogale, Luc Vincent, and Josh Weaver. Google street view: Capturing the world at street level. *Computer*, 43(6):32–38, 2010. 2

[2] Jan Böhm. Multi-image fusion for occlusion-free façade texturing. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 35(5):867–872, 2004. 2

[3] Alexander Bornik, Konrad Karner, Joachim Bauer, Franz Leberl, and Heinz Mayer. High-quality texture reconstruction from multiple views. *The journal of visualization and computer animation*, 12(5):263–276, 2001. 2

[4] Yao-Xun Chang and Zen-Chung Shih. The synthesis of rust in seawater. *The Visual Computer*, 19(1):50–66, 2003. 2

[5] Yanyun Chen, Lin Xia, Tien-Tsin Wong, Xin Tong, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Visual simulation of weathering by γ-ton tracing. In *ACM SIGGRAPH*, pages 1127–1133. 2005. 2

[6] 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 *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8789–8797, 2018. 2

[7] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Stichting Blender Foundation, Amsterdam, 2018. 3

[8] Dengxin Dai, Hayko Riemenschneider, Gerhard Schmitt, and Luc Van Gool. Example-based facade texture synthesis. pages 1065–1072, 2013. 2

[9] Soheil Darabi, Eli Shechtman, Connelly Barnes, Dan B Goldman, and Pradeep Sen. Image melding: combining inconsistent images using patch-based synthesis. *ACM SIGGRAPH*, 31(4):1–10, 2012. 2

[10] 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. 2

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

[12] Esri. Esri CityEngine 2020.1, 2020. 1

[13] Lubin Fan, Przemyslaw Musialski, Ligang Liu, and Peter Wonka. Structure completion for facade layouts. *ACM SIGGRAPH Asia*, 33(6), Nov. 2014. 2

[14] Clara Fernandez-Labrador, Jose M. Facil, Alejandro Perez-Yus, Cendric Demonceaux, Javier Civera, and Jose J. Guerrero. Corners for layout: End-to-end layout recovery from 360 images. 2019. 4, 7

[15] Anna Frühstück, Ibraheem Alhashim, and Peter Wonka. Tilegan: synthesis of large-scale non-homogeneous textures. *ACM SIGGRAPH*, 38(4):1–11, 2019. 2

[16] Leon Gatys, Alexander S Ecker, and Matthias Bethge. Texture synthesis using convolutional neural networks. In *NIPS*, pages 262–270, 2015. 2

[17] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016. 2

[18] Éric Guérin, Julie Digne, Éric Galin, Adrien Peytavie, Christian Wolf, Bedrich Benes, and Benoît Martinez. Interactive example-based terrain authoring with conditional generative adversarial networks. *ACM TOG*, 36(6):1–13, 2017. 2

[19] Xi Guo, Zhicheng Wang, Qin Yang, Weifeng Lv, Xianglong Liu, Qiong Wu, and Jian Huang. Gan-based virtual-to-real image translation for urban scene semantic segmentation. *Neurocomputing*, 2019. 2

[20] Tobias Günther, Kai Rohmer, and Thorsten Grosch. GPU-accelerated Interactive Material Aging. In Michael Goesele, Thorsten Grosch, Holger Theisel, Klaus Toennies, and Bernhard Preim, editors, *Vision, Modeling and Visualization*. The Eurographics Association, 2012. 2

[21] Peter Hedman, Julien Philip, True Price, Jan-Michael Frahm, George Drettakis, and Gabriel Brostow. Deep blending for free-viewpoint image-based rendering. *ACM TOG*, 37(6):1–15, 2018. 2

[22] Paul Henderson, Vagia Tsiminaki, and Christoph H Lampert. Leveraging 2d data to learn textured 3d mesh generation. pages 7498–7507, 2020. 2

[23] Philipp Henzler, Niloy J Mitra, , and Tobias Ritschel. Learning a neural 3d texture space from 2d exemplars. In *IEEE CVPR*, June 2019. 2

[24] Jingwei Huang, Justus Thies, Angela Dai, Abhijit Kundu, Chiyu Max Jiang, Leonidas Guibas, Matthias Nießner, and Thomas Funkhouser. Adversarial texture optimization from rgb-d scans. *arXiv preprint arXiv:2003.08400*, 2020. 2

[25] Xun Huang, Ming-Yu Liu, Serge Belongie, and Jan Kautz. Multimodal unsupervised image-to-image translation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 172–189, 2018. 2

[26] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *IEEE CVPR*, 2017. 2

[27] Justin Johnson, Alexandre Alahi, and Fei fei Li. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*, 2016. 3

[28] Hiroharu Kato, Yoshitaka Ushiku, and Tatsuya Harada. Neural 3d mesh renderer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3907–3916, 2018. 2

[29] Tom Kelly, Paul Guerrero, Anthony Steed, Peter Wonka, and Niloy J Mitra. Frankengan: guided detail synthesis for building mass models using style-synchonized gans. *ACM TOG*, 37(6):1–14, 2018. 2, 6

[30] Tom Kelly and Peter Wonka. Interactive architectural modeling with procedural extrusions. *ACM TOG*, 30(2):14, 2011. 1

[31] Bryan Klingner, David Martin, and James Roseborough. Street view motion-from-structure-from-motion. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 953–960, 2013. 2

[32] Chuan Li and Michael Wand. Approximate translational building blocks for image decomposition and synthesis. *ACM Transactions on Graphics (TOG)*, 34(5):1–16, 2015. 2

[33] Lin Liang, Ce Liu, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Real-time texture synthesis by patch-based sampling. *ACM Trans. Graph.*, 20(3):127–150, July 2001. 2

[34] Jianye Lu, Athinodoros S Georghiades, Andreas Glaser, Hongzhi Wu, Li-Yi Wei, Baining Guo, Julie Dorsey, and Holly Rushmeier. Context-aware textures. *ACM Transactions on Graphics (TOG)*, 26(1):3–es, 2007. 2

[35] Moustafa Meshry, Dan B Goldman, Sameh Khamis, Hugues Hoppe, Rohit Pandey, Noah Snavely, and Ricardo Martin-Brualla. Neural rerendering in the wild. In *IEEE CVPR*, pages 6878–6887, 2019. 2

[36] Piotr Mirowski, Andras Banki-Horvath, Keith Anderson, Denis Teplyashin, Karl Moritz Hermann, Mateusz Malinowski, Matthew Koichi Grimes, Karen Simonyan, Koray Kavukcuoglu, Andrew Zisserman, et al. The streetlearn environment and dataset. *arXiv preprint arXiv:1903.01292*, 2019. 4

[37] Pascal Mueller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. *ACM TOG*, 25(3):614–623, 2006. 1, 4

[38] Pascal Müller, Gang Zeng, Peter Wonka, and Luc Van Gool. Image-based procedural modeling of facades. *ACM SIGGRAPH*, 26(3):85, 2007. 1, 2

[39] Wolfgang Niem and Hellward Broszio. Mapping texture from multiple camera views onto 3d-object models for computer animation. In *Proceedings of the International Workshop on Stereoscopic and Three Dimensional Imaging*, pages 99–105, 1995. 2

[40] Michael Oechsle, Lars Mescheder, Michael Niemeyer, Thilo Strauss, and Andreas Geiger. Texture fields: Learning texture representations in function space. In *IEEE ICCV*, pages 4531–4540, 2019. 2

[41] Yoav I. H. Parish and Pascal Müller. Procedural modeling of cities. In *Proceedings of SIGGRAPH 2001*, pages 301–308, 2001. 1

[42] Taesung Park, Richarf Zhang Alexei A. Efros, and Jun-Yan Zhu. Contrastive learning for unpaired image-to-image translation. *ECCV*, 2020. 2, 3, 4, 5

[43] Amit Raj, Cusuh Ham, Connelly Barnes, Vladimir Kim, Jingwan Lu, and James Hays. Learning to generate textures on 3d meshes. In *IEEE CVPR Workshops*, June 2019. 2

[44] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 2

[45] Michael Schwarz and Pascal Müller. Advanced procedural modeling of architecture. *ACM TOG*, 34(4):107:1–107:12, 2015. 1

[46] Alisha Sharma and Jonathan Ventura. Unsupervised learning of depth and ego-motion from cylindrical panoramic video. *arXiv preprint arXiv:1901.00979*, 2019. 2

[47] Shuran Song, Andy Zeng, Angel X Chang, Manolis Savva, Silvio Savarese, and Thomas Funkhouser. Im2pano3d: Extrapolating 360 structure and semantics beyond the field of view. In *IEEE CVPR*, pages 3847–3856, 2018. 2

[48] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. *arXiv preprint arXiv:1808.06601*, 2018. 2

[49] Li-Yi Wei and Marc Levoy. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, page 479–488, USA, 2000. ACM Press/Addison-Wesley Publishing Co. 2

[50] Chao Wen, Yinda Zhang, Zhuwen Li, and Yanwei Fu. Pixel2mesh++: Multi-view 3d mesh generation via deformation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1042–1051, 2019. 2

[51] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *NIPS*, pages 82–90, 2016. 2

[52] X Zhang, C May, and D Aliaga. Synthesis and completion of facades from satellite imagery. 2020. 2

[53] Yang Zhou, Zhen Zhu, Xiang Bai, Dani Lischinski, Daniel Cohen-Or, and Hui Huang. Non-stationary texture synthesis by adversarial expansion. *ACM SIGGRAPH*, 37(4), July 2018. 2

[54] Zhu, Jun-Yan, Park Taesung, Isola, Phillip, and Efros Alexei A. Unpaired image-to-image translation using cycle-consistent adversarial networks. *IEEE ICCV*, 2017. 2