StreetGAN: Towards Road Network Synthesis with Generative Adversarial Networks

Stefan Hartmann Michael Weinmann Raoul Wessel Reinhard Klein
University of Bonn
Institute for Computer Science II
Friedrich-Ebert-Allee 144
Germany, 53113 Bonn
{hartmans,mw,wesselr,rk}@cs.uni-bonn.de

ABSTRACT

We propose a novel example-based approach for road network synthesis relying on Generative Adversarial Networks (GANs), a recently introduced deep learning technique. In a pre-processing step, we first convert a given representation of a road network patch into a binary image where pixel intensities encode the presence or absence of streets. We then train a GAN that is able to automatically synthesize a multitude of arbitrary sized street networks that faithfully reproduce the style of the original patch. In a post-processing step, we extract a graph-based representation from the generated images. In contrast to other methods, our approach does neither require domain-specific expert knowledge, nor is it restricted to a limited number of street network templates. We demonstrate the general feasibility of our approach by synthesizing street networks of largely varying style and evaluate the results in terms of visual similarity as well as statistical similarity based on road network similarity measures.

Keywords
deep learning, generative modeling, generative adversarial networks (GANs), road network generation.

1 INTRODUCTION

High-quality productions such as video games, simulations or movies rely on high-quality content including detailed virtual environments, buildings and realistic road networks. However, the manual design and modeling of such content from scratch is a time-consuming and tedious task. Therefore, the automation of high-quality content production has become an active line of research in recent years. Automatic content generation has been addressed using various approaches that follow the concepts of procedural modeling, inverse-procedural modeling or example-based modeling. While procedural approaches rely on manually designed grammar snippets and rule sets to derive geometric representations of buildings, plants or road networks, inverse procedural approaches try to infer the production rules from a given set of existing examples. In contrast, example-based approaches inspect small real-world examples and decompose them into a set of building blocks in an offline step. Novel content is then generated by custom tailored algorithms that reshuffle, recombine, and bend the content in order to statistically and perceptually match the style present in the examples. Furthermore, the potential of deep learning techniques for procedural content generation [23, 15, 32] has been investigated. These techniques based on convolutional neural networks (CNNs) have already been established as promising workhorses in other areas of computer graphics like texture synthesis [11, 20].

In this work, we propose a novel example-based approach for road network generation that leverages the potential of modern deep learning techniques. The input for our method is a road network patch extracted from *OpenStreetMap* (OSM). As the data is publicly available and maintained by a large community no further domain-specific expert knowledge for data preparation and/or annotation is required. Our method comprises three major components. The first component prepares and converts an input road network into a binary image, where the pixel intensities encode the presence or absence of roads. The second component trains a generative adversarial network (GAN) [14] on image patches extracted from the prepared road network image. The third step utilizes the GAN to synthesize arbitrary sized images that contain a rastered road network. In order to use the produced road network encoded in the image in GIS applications such as CityEngine[8], we extract the road graph and post-process it in a final step. The results shown in Section 5 illustrate that our approach is able to synthesize road networks that are visually similar when compared to the original road networks. Moreover, they also faithfully reproduce road network properties like city block area, compactness and block aspect ratio (see Section 5.2). The statistical evaluation furthermore shows that the major characteristics and the style of the road network present in the original networks can also be found in the synthesized results.

To the best of our knowledge, no other method for road network generation using a GAN approach has been published so far. However, we believe that this technique is particularly well-suited for the envisioned task, as due to their nature GANs try to learn the distributions that underlie a set of training images or patches. This might overcome the need for manual definition of rule sets and parameter tuning for procedural algorithms which can be tedious for non-expert users. In addition, such a technique might boost the expressiveness of custom-tailored example-based synthesis algorithms that is typically limited by the variation found within the input template.

2 RELATED WORK

The first part of this section reviews procedural and example-based approaches with a strong focus on road network generation. In the second part, we briefly review content generation approaches that leverage the power of deep learning techniques. As no approach for road network generation that leverages deep learning techniques has been presented so far, we instead review approaches that combine CNNs with procedural and data-driven content generation algorithms.

Procedural approaches: In a comprehensive survey on procedural and inverse procedural modeling, Smelik et al. [29] discuss different approaches and applications. In general, procedural methods rely on the use of manually defined or automatically determined rule sets for content generation. Such approaches have e.g. been followed by Parish and Müller [25], where *Open L-Systems* are used to procedurally grow road networks from an initial seed point. Galin et al. [10] generate procedural roads between two end points by computing the anisotropic shortest path incorporating the underlying terrain and user defined environmental cost functions. In Galin et al. [9], the focus is on generating road networks for inter-city connection. Benes et al. [2] grow street networks from multiple seed points. Each seed points represents an individual city, that is expanded by guiding the growth process with a virtual traffic simulation. The controllability of procedural road networks was improved by Chen et al. [4] using tensor and direction fields to guide the road network generator. Emilien et al. [6] focus on procedural generation of villages on arbitrary terrain. Their road network generator is custom tailored to courses of streets found in small villages.

Example-based approaches: In contrast to procedural approaches, example-based methods do not require an underlying rule set to generate content. Instead, they rely on analyzing the data such as the road network or a city layout in a pre-processing step to extract templates and/or statistical information. In Aliaga et al. [1], intersections are enriched with attributes such as intersection degree, street level, etc. A novel network is generated by employing a random walk using the attributes stored at junctions as guidance. Yang et al. [31] focus on the synthesis of suburban areas and districts. Starting from an empty domain they apply a recursive splitting technique based on either template matching followed by deformation, or streamline-based splitting using a crossfield. Emilien et al. [7] learn distributions from small patches of generated or manually modeled 3D content. The learned distributions are applied in a brushed-based modeling metaphor in order to steer the underlying procedural content generators that produce roads, foliage, trees or buildings. Nishida et al. [22] extract road patches from real road networks. From an initial seed point a road network is grown by attaching road patches to connector streets taking the terrain into account. In cases where no example-based growth is possible, statistical growing similar to [1] is employed.

Learning-based approaches: Emerging deep learning techniques have been used for procedural and data-driven content generation. In Yumer et al. [32], a low-dimensional generative model from high-dimensional procedural models incorporating shape features is learned. Novel models can then be generated by interpolating between points in a low-dimensional latent space enabling faster and more intuitive modeling of shape variations. Huang et al. [15] present a sketch-modeling system using CNNs. CNNs are trained on synthetic line drawings produced by procedural models with varying parameters. Novel shapes can then be generated by regressing the parameters according to a user provided sketch depicting the desired output. A similar approach was proposed by Nishida et al. [23] focusing on interactive modeling of buildings. The authors train CNNs for classifying the type of a rule snippet as well as for regressing their parameter sets from synthetic line renderings of the rule snippets. The user iteratively sketches building mass, roof, etc., and the CNNs are used to classify the resulting shapes and to infer their parameters. Ritchie et al. [27] focus on controlling procedural algorithms using neural networks. In particular, the neural network manipulates the next steps of the algorithm based on the content generated so far. Apart from the approaches that require the existence of procedural models/algorithms, pure image-based algorithms have been investigated for controlled content generation. Isola et al. [16] investigate GANs

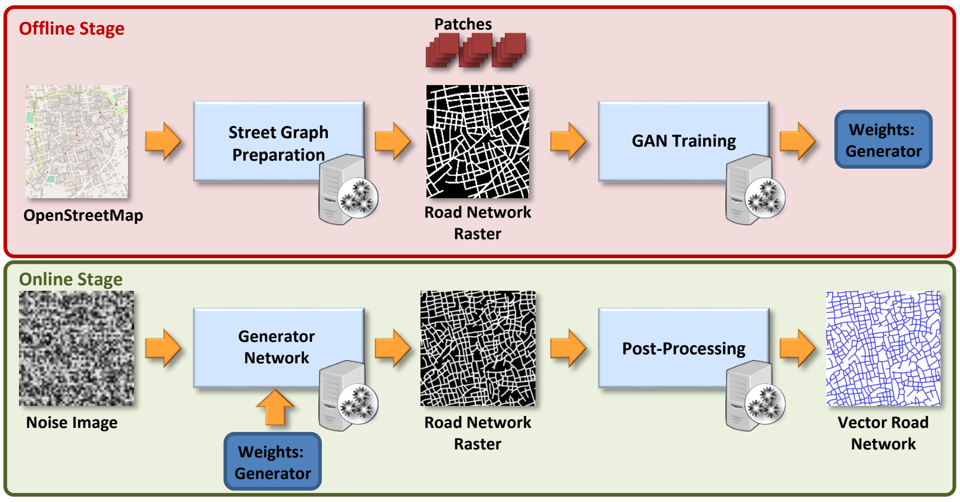


Figure 1: Our system is composed of two components. In an offline step, a road network patch taken from a real-world city is rastered into an image. The rastered road network is used to train a GAN and the generator weights are stored. In an online step, the trained model, i.e. the generator weights, are used to synthesize road network variations from images containing uniformly sampled noise. A clean graph is extracted from the produced image ready to use in GIS applications.

for transfer learning, i.e. they learn a mapping from one object representation into another such as *urban map to aerial image* or *sketch to image*. The authors show that GANs can be used to learn such a mapping without custom feature engineering. More recently, a texture synthesis approach utilizing GANs has been proposed by Jetchev et al. [17]. With their framework, they are able to synthesize textures of arbitrary size from spatial noise. This technique called *Spatial GAN (SGAN)*, a specialized GAN technique, serves as basis for our approach.

3 REVIEW OF GENERATIVE ADVERSARIAL NETWORKS

Before outlining our approach in Section 4, we provide a brief overview about generative adversarial networks (GANs) that we apply to generate road networks. GANs are a technique to learn a generative model based on concepts from game theory. The key ingredients of GANs are given by two players, a generator G and a discriminator D. The generator is a function $G_{\theta^G}(z): \mathbb{R}^d \to \mathbb{R}^{w\times h\times c}$ that takes as input a vector $z \in \mathbb{R}^d$ sampled from a d-dimensional prior distribution $p_z(z)$ such as a uniform distribution and uses the parameters $\theta^{(G)}$ to transform it into a sample image x'. The fabricated sample $x' = G(z)$ is an image $x' \in \mathbb{R}^{h\times w\times c}$, where w and h denote its width and its height and c denotes its channels. In contrast, the discriminator D is a function $D_{\theta^D}(x): \mathbb{R}^{w\times h\times c} \to \mathbb{R}$ that takes as input either an image patch x from the training set or a fabricated image x', and uses its parameters $\theta^{(D)}$ to produce a scalar that represents the probability that the investigated sample is a example x from training set, or a fabrication x' produced by G. The discriminator cost is accordingly given by

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2}\mathbb{E}_{x\sim p_{data}(x)} \log(D(x)) \\ -\frac{1}{2}\mathbb{E}_{z\sim p_z(z)} \log(1 - D(x'))$$

which is the standard cross-entropy for a binary classification problem. The discriminator tries to minimize $J^{(D)}(\theta^{(D)}, \theta^{(G)})$ while it controls only $\theta^{(D)}$, however, it also depends on the parameters $\theta^{(G)}$ of the generator. The term, $\mathbb{E}_{x\sim p_{data}(x)}[\log(D(x)]$, measures the skill of D to distinguish fabricated samples x' from real ones x that are produced by the data-generating distribution p_{data}. In contrast, $\mathbb{E}_{z\sim p_z(z)}[\log(1 - D(x')]$ measures the skill of G to fabricate examples, that are misclassified by D and thus considered as real examples. In the previous terms, $\mathbb{E}$ represents the expectation value of the *log*-probability, which in practice boils down to the arithmetic mean of the *log*-probabilities computed using the samples of the current training iteration. The cost function of G is given by $J^{(G)}(\theta^{(D)}, \theta^{(G)}) = -J^{(D)}(\theta^{(D)}, \theta^{(G)})$ and its goal is to maximize D's error on the fabricated examples x'. As both cost functions follow different goals and compete against each other, the underlying problem is described as a game between the two players [13, 12]. One strategy to solve the described problem is in terms of a zero-sum game also called *minimax* game. The game is accordingly described by the objective

$$\arg\min_{\theta^{(G)}} \max_{\theta^{(D)}} \; -J^{(D)}(\theta^{(D)}, \theta^{(G)})$$

where $-J^{(D)}(\theta^{(D)}, \theta^{(G)})$ represents the discriminator's pay-off. The overall goal of such a game is to minimize the possible loss for a worst case maximum loss. In particular, this is realized by performing a minimization in an outer loop, while performing a maximization in an inner loop. We refer the reader to a recent tutorial by Goodfellow [12] for additional details.

In practice, G and D are represented as neural networks and training them is similar to finding the *Nash* equilibrium of the described *minimax* game played by G and D. A *Nash* equilibrium in such a context can be described as a parameter state $(\theta^{(D)}, \theta^{(G)})$ that is a local minimum of $J^{(D)}$ and a local minimum of $J^{(G)}$. In order to keep the problem tractable, G and D are trained in an alternating fashion instead of using the nested loops as described above. Furthermore, G's cost function $J^{(G)}(\theta^{(D)}, \theta^{(G)})$ is changed to $-\frac{1}{2}\mathbb{E}_{z \sim p_z(z)} \log(D(x'))$. The term in the original cost function $-\frac{1}{2}\mathbb{E}_{z \sim p_z(z)} \log(1 - D(x'))$ would lead to vanishing gradients during the training, when D successfully rejects examples fabricated by G. Instead of previously minimizing the *log*-probability that the sample x' is classified as fabricated, the new goal of the generator G is now to maximize the *log*-probability that D performs a wrong classification. As noted in [12], that change enables both G and D to produce strong gradients during the final training.

For the modified game and its training this particularly means that in one iteration D is trained, while in the next iteration G is trained. As we search a local minimum for D and G, the parameters of current component are updated in each iteration using stochastic gradient descent. When G is trained, its parameters are tuned towards the production of samples x' that are indistinguishable from the real training data and thus to fool the discriminator D. In contrast, when D is trained its parameters are tuned to improve D's skill to discriminate the fabricated samples x' from the real samples x. For additional details about the theoretic background we refer the interested reader to [12, 17].

So far, when the GAN is trained using neural networks, no well-founded theory about the determination of the success of training procedure of a GAN can be found in literature. Therefore, it is necessary to visually check generated samples and to capture the weights θ^G of G that fabricate visually pleasing outputs. Note, there is no need to capture θ^D because after the training D can be omitted and only the generator G is necessary to produce new samples [13, 12, 17].

4 ROAD NETWORK SYNTHESIS USING GAN

In this Section, we outline our street network generation approach by providing an brief description of its major components.

4.1 Pipeline Overview

In order to successfully apply a GAN model to street network data in vector representation as provided by *OpenStreetMap* (OSM), we developed three components to approach this task (see Figure 1). In an offline step, a sample map from OSM is used to create an image that contains a raster representation of the street network (see Section 4.2). The produced image is used to extract a set of randomly chosen image patches of size $n \times n$. These patches contain subparts of the initial road network and are used to train the GAN (see Section 4.3). Afterwards, novel road network patches can be generated from a d-dimensional z with samples drawn from $p_z(z)$. z serve as input for the generator network G that maps them to a grayscale image $x \in \mathbb{R}^{w \times h \times 1}$ representing a rastered road network. The resulting road network is encoded by pixel intensities (cf. Section 4.4) and the discrete representation is transformed into a graph-based one in a post-processing step.

4.2 Road Network Preparation

We use publicly available community mapping data from OpenStreetMap (OSM) [24] datasets in which road networks are represented as piecewise-linear polylines, that are attached a *highway* label in order to distinguish them from other structures like buildings, rivers, and parks. Among other polylines in an OSM dataset roads can be identified by the label *highway* attached them. Each road is assigned a specific *highway* type representing its category. For all our examples, we extract roads from the OSM dataset that have one of the following *highway*-categories: **motorway, primary, secondary, tertiary, residential, living_street, pedestrian**. The raw road network extracted from OSM is represented as vector data in geo-coordinates. As well-established CNN pipelines require images as input, we transform the road network into a raster representation. First, we project the geo-coordinates to WGS84, which is a well-established coordinate projection that transforms geo-coordinates given in *Latitude/Longitude* to meters. Next, we scale the road network that each pixel in the rastered representation represents an area of 3×3 meters. Finally, we raster the road segments as lines with a width of 15 pixels using the line rasterization routine of OpenCV [3] to produce a binary image, in which white pixels now represent the presence of roads while black pixels represent the absence of roads. Please note that we inverted the colors in the Figures shown in the paper.

4.3 Training Procedure

We train the GAN on images patches with fixed size of $n \times n$ pixels extracted from the image containing the rastered road network. In order to provide a suitable large training set and to enable the network to capture the local statistics well, we perform the training on images patches. A well-established approach for training GANs is an iterative and alternating training procedure. This means that G and D are trained in an alternating fashion every second iteration. In the step when G is trained, it takes as input a set $Z = \{z_0, \ldots, z_k\}$. As described in Jetchev et al. [17], that serves as basis for our training, each z_i is a tensor $z_i \in \mathbb{R}^{m \times n \times d}$ where at each spatial location a d-dimensional vector drawn from p_z it used. The z_i and is then transformed by G into a grayscale image $x'_i = G(z_i)$ of size $x'_i \in \mathbb{R}^{n \times n \times 1}$. When D is trained, a set $X = \{x_0, \ldots, x_k\}$ of k image patches $x_i \in \mathbb{R}^{n \times n \times 1}$ and the set $X' = \{x'_0, \ldots, x'_k\}$ generated by G in the previous step serve as input. The samples $x \in X$ are extracted from random locations inside the training image. We refer the reader to the work by Jetchev et al. [17] for additional details about the training procedure. Please note that we use only a single image, that provides patches for the training procedure, but using multiple different images would be possible and would increase the examples in the training set.

4.4 Road Network Post-Processing

In order to use the resulting road network in GIS applications or in a road network analysis task, we need to transform grayscale image intensities to a road network graph. For this purpose, we apply a post-processing to the synthesized images.

Image post-processing: The grayscale images produced by the generator network contain pixel intensities in the range $[0, 255]$ (see Figure 4). In a first step, we threshold the gray values at 127 in order to produce a binary image where pixels set to *true* represent the presence and non-set pixels represent the absence of road. Applying the threshold might produce undesirable isolated pixels and also small cusps along road regions. In order to get rid of these artifacts, we apply a morphological erosion operation. However, the produced result might still contain small holes or road regions that do not touch within a radius of up to five pixels. In order to close such small gaps, we apply five steps of morphological dilation. For all morphological operations, we use a 3×3 structuring element with the shape of a cross. The obtained initial road network, however, contains road regions that are too thick to extract an initial road graph. Therefore, we thin out the result to extract a skeleton from the cleaned binary image using the algorithm from Zhang et al. [33].

Road graph construction: We utilize the pixel-skeleton from the previous step to construct an initial graph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ representation of the synthesized road network, where $\mathscr{V}$ are its vertices and $\mathscr{E}$ are its edges. In order to construct $\mathscr{G}$, we add a node V_i to $\mathscr{V}$ for each of the skeleton pixels. Next, we examine the 8-neighborhood of each V_i in the image. For each skeleton pixel V_j inside the 8-neighborhood of V_i, we add an edge $E_{ij} = (V_i, V_j)$.

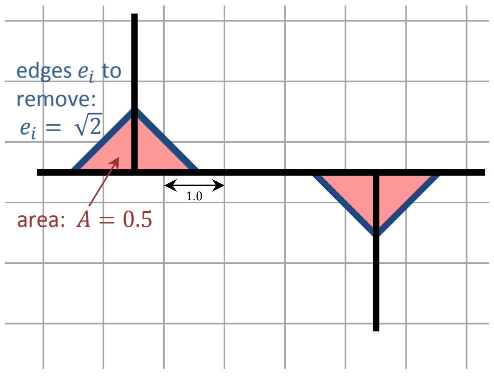


Figure 2: Block artifacts resulting from graph construction

Cityblock cleanup: The graph construction from the pixel skeleton produces regions within the road network that have a very small area of 0.5 square pixels (see Figure 2), which are removed in a first step. The regions within a road network graph are typically called city blocks. Strictly speaking, a city block is a region within the graph $\mathscr{G}$, that is enclosed by a set of road segments and might contain dead-end street segments. In order to identify these small regions, we first compute all the city blocks of the graph. As the graph is a directed graph and embedded in $\mathbb{R}^2$, the city blocks can be computed by determining the minimal cycles of the graph by computing loops of edges. Next, we filter out blocks with an area of 0.5 pixels. These artifact blocks can be removed by identifying and removing their longest edge, which has a length of $\sqrt{2}$.

Road courses smoothing: Another artifact produced by constructing $\mathscr{G}$ from the image raster are jagged edges. In order to smooth these in the final graph, we extract a set of street chains $S = \{S_i\}$ from the graph. Each $S_i = \{V_0, \ldots, V_n\}$ consists of n nodes, while the degrees of V_0 and V_n are constrained by $deg(V_0) \neq 2$ and $deg(V_n) \neq 2$. From each of the S_i, a polyline $P_i = \{p_0, \ldots, p_n\}$ with n positions is built. A smoothed version of the positions can be obtained by applying 5 steps of local averaging $\hat{p}_i = \frac{1}{4}p_{i-1} + \frac{1}{2}p_i + \frac{1}{4}p_{i+1}$ to the p_i's with $i \in [1, n-1]$, and replacing the original p_i's with their smoothed version $\hat{p}_i$. Finally, we additionally straighten the road courses, by removing superfluous nodes using the Douglas Peucker simplification [5]. We allow to remove nodes that deviate up to 3 meters from a straight line. The last step removes short dead-end street chains with an total length of less than 25 meters.

5 CASE STUDY: ROAD NETWORK TYPES

In order to showcase the versatility of our road network synthesis approach, we evaluate our approach on a set of challenging test cases. We composed a collection of real-world as well as synthetic road network examples (see Figure 3 a)-d)). The real-world examples were taken from OSM, while the synthetic ones were taken from [21]. For all the examples shown in here, we used a patch size of 321×321 (cf. Figure 9) pixel during the training procedure, because that size captures the local structures found in the our test road networks. Furthermore, we used only a single road network image from which patches were extracted. We synthesized two examples for each road network shown in Figure 3 a)-d). In our evaluation, we investigate the visual appearance of the generated results and analyse the similarity in terms of road networks measures such as area, compactness and aspect ratio of the city blocks by comparing the resulting distributions.

5.1 Visual Evaluation

Irregular: Synthetic As a first test case, we considered a synthetic road network (see Figure 3a). The major characteristics of this road network are blocks of different sizes with and without dead-ends, and similar sized blocks, that form small groups. Nearly all blocks have a rectangular shape except for a few exceptions. Figure 4 (a) and (b) show road networks generated by GAN model after passing our post-processing pipeline. It can be noticed, that the generated results contain blocks similar in shape and size when compared to the original network. Notice, that the results even contain the small groups of nearly square shaped blocks that are present in the original network. Larger road courses are present in the examples, although they have a curvature different to that in the original network.

Irregular: San Marco Next, we evaluated a street network patch from a village in Italy (see Figure 3b). A major characteristic of that network is its large amount of small city blocks in comparison to only a few larger ones. Generated samples of this network type are depicted in Figure 4c and 4d. Both samples contain a significant number of small blocks when compared to the number of medium sized and large city blocks. It is also noticeable that smaller blocks are located next to each other. Furthermore, the result contains large scale structure such as connected road courses that separate groups of smaller blocks. Another produced sample visualized using CityEngine can be seen in Figure 12.

Irregular: Berlin In contrast to the previous example, the next network shown in Figure 3c is composed of a significant amount of larger, mainly square or rectangular shaped blocks. Only a few blocks are irregularly shaped and contain dead-ends. The generated samples shown in Figure 4e and 4f contain a significant amount of nearly square shaped blocks and rectangular shaped blocks. It can be recognized that the generated networks also contain irregularly shaped blocks and even L-shaped blocks not being present in the examples.

Suburban: Synthetic Next we show results generated from a synthetic network of a suburban region with structures mainly found in suburban regions of the US (see Figure 3d). A major property of such network types is the presence of curved road courses. Our produced results shown in Figure 4g and 4h contain these typical curved roads shapes.

5.2 Statistical Evaluation

Apart from the visual comparison of the results, we performed an evaluation of graph measures computed on the synthesized road networks and the original road networks. These considered measures include the *cityblock area*, the *compactness*, i.e. the ratio between block area and its minimal bounding box, and the *city block aspect ratio*, i.e. the ratio between the shorter and the longer side of the minimal bounding box.

Irregular: Synthetic Figure 5 compares the graph measures between the synthetic irregular network shown in Figure 3a with the ones obtained from our synthesized results. While the distributions of the block area and the compactness have a similar shape, the aspect ratio distribution varies as the generated result contains much more variation of rectangular shaped blocks than the original road network.

Irregular: San Marco In this result (see Figure 3b), the distributions of block area, aspect ratio and compactness are similar (see Figure 6). The resulting network mostly consists of small city blocks as illustrated by the block area distribution. Both the original and the generated road network contain a significant amount of nearly rectangular blocks (see compactness). As the aspect ratios within the generated network are also similar, thus, learned model has captured the properties of the original network.

Irregular: Berlin In Figure 7, we illustrate the distributions for the Berlin example shown in Figure 3c. While the block area and the aspect ratio of the blocks found in the generated example tend to be similar, the compactness varies more than in the previous examples. As the streets in the produced network are not perfectly straight anymore, the compactness of the blocks deviates from being nearly 1.0.

Suburban: Synthetic For suburban networks such as the one shown in Figure 3d, the distributions of block area and aspect ratio differ, while especially the aspect ratios within the generated network have a few

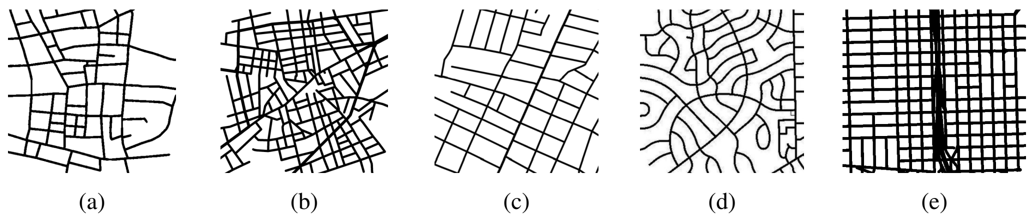


Figure 3: Overview of the different road network styles used in our case study: (a) Synthetic irregular, (b) Cellino San Marco irregular, (c) Berlin irregular, (d) Synthetic suburban, (e) Portland with highway ramps

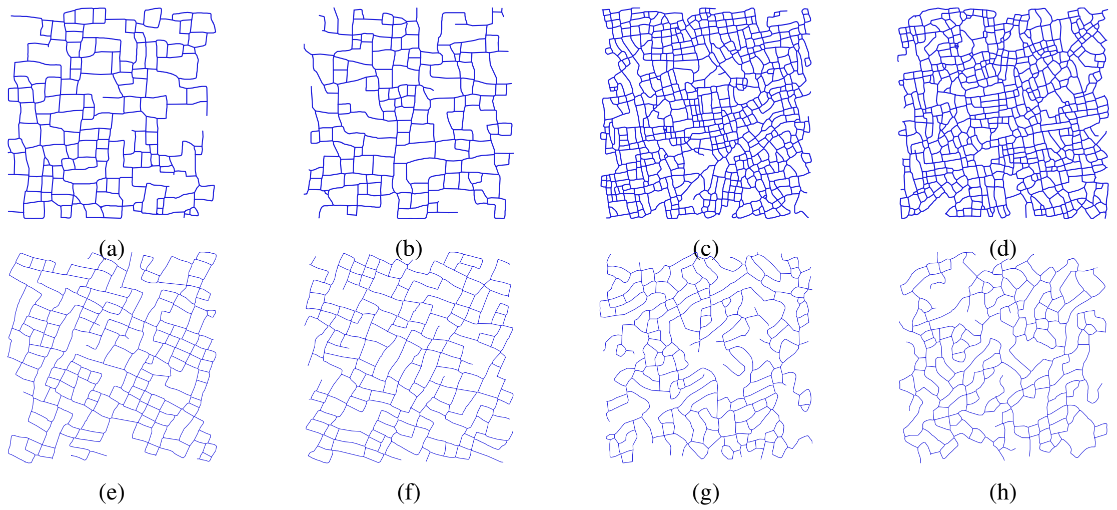


Figure 4: Different samples fabricated by the generator learned from the synthetic irregular example shown in Figure 3. Synthetic irregular (a) and (b), Cellino San Marco (c) and (d), Berlin irregular (e) and (f), Synthetic suburban (g) and (h). The results are discussed in detail in Section 5.

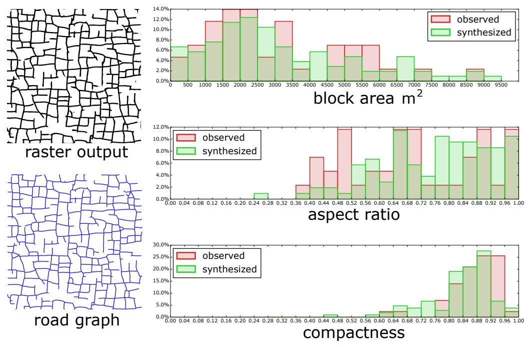


Figure 5: Statistical evaluation of Synthetic irregular

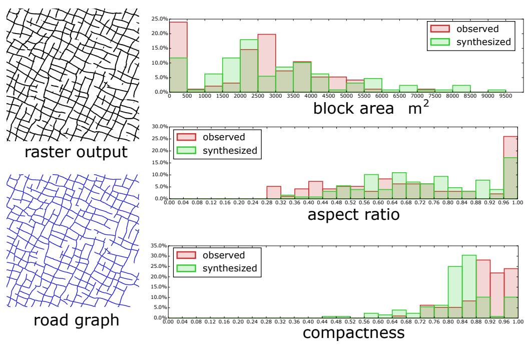


Figure 7: Statistical evaluation of Berlin irregular

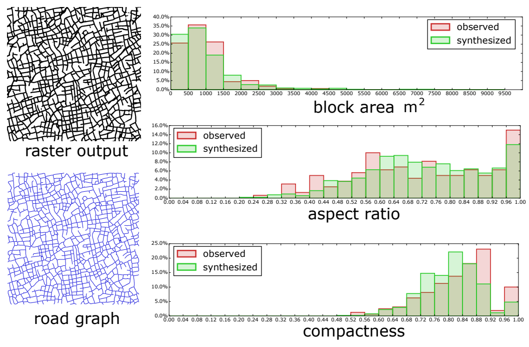


Figure 6: Statistical evaluation of Cellino San Marco

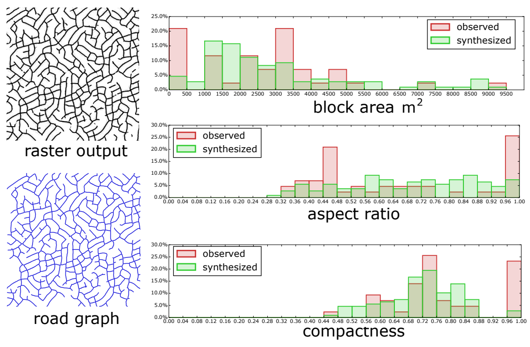


Figure 8: Statistical evaluation of synthetic suburban

spikes (cf. Figure 8). However, at a larger scale the overall shape of the distribution is similar. As this road network type contains large-scale structures such as curved roads that pass through the whole network the chosen context size cannot capture these, thus, the generated network will suffer from these missing global properties. This leads to a structurally different generated road network which is reflected by the distributions of the different graph measures.

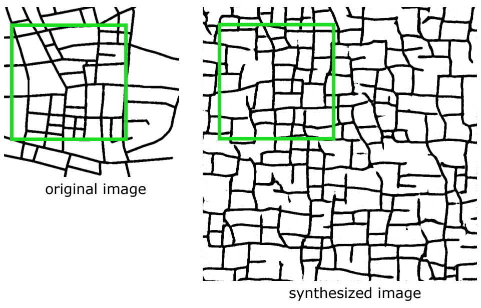


Figure 9: Illustration of the context size using during the training stage. Left: Original image with overlaid extent of the training image. Right: Generated sample with an overlay of the training image size.

5.3 Limitations

Large-scale structures and ramps. We noticed that our approach cannot successfully handle road network patches that contain highway ramps and networks that contain street lanes that are located very close to each other, as illustrated in the road network example taken from Portland (see Figure 3e). When the road network is rastered nearby lanes will be merged with and form even thicker lanes. If highway ramps are present, additional pixel blobs are introduced as illustrated in the synthesized example shown in Figure 10. It can be noticed that the grid-like road pattern is faithfully reproduced. However, due to the thick lanes and the limited context size (see Figure 9) the highway structures present in the training data cannot be recovered successfully. Instead, thick road structures occur on the left border (cf. green arrows) and blob shaped artifacts are scattered over the synthesized example (cf. region surrounded by green ellipses). When the post-processing is applied, these artifacts will be alleviated, however, irregularly shaped blocks will be present in the final road network.

Deadend roads. All the synthesized examples contain much more dead-ends when compared with the number of dead-ends present in their corresponding original road network. This might be due to the patch-based training procedure. Each patch that is used for training typically contains virtual dead-end street segments that abruptly end at the patch boundary.

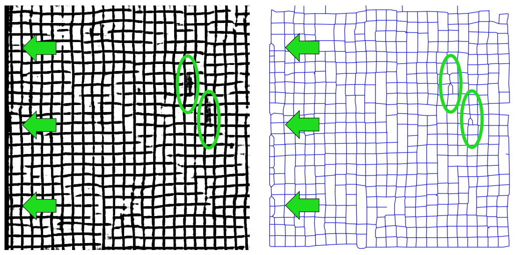


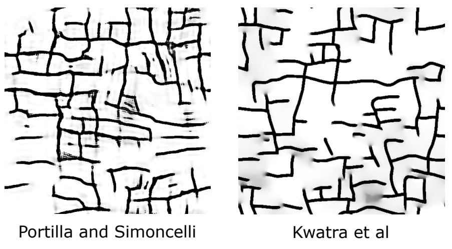
Figure 10: In case of nearby located highway lanes and highway ramps, the GAN fails to capture these properties. This leads to blob-like artifacts in the generated samples.

5.4 Street Network Generation with Texture Synthesis Techniques

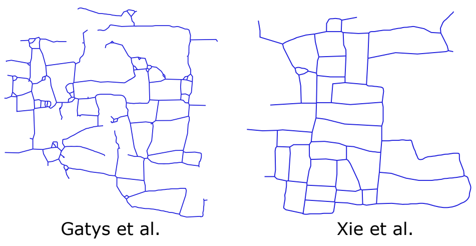
We complete our case study with a brief evaluation of texture synthesis algorithms for street network generation. In particular, we synthesize road networks with patch-based texture synthesis algorithms such as method of Portilla and Simoncelli [26] and Kwatra et al. [19]. Furthermore, we evaluate recent CNN based texture synthesis algorithms, specifically the method of Gatys et al. [11] and the Generative ConvNet technique proposed by Xie et al. [30] for road network generation. Figure 11a illustrates results from Portilla and Simoncelli on the left-hand side and results from the method of Kwatra on the right-hand side. For both algorithms a variation of the irregular road network shown in Figure 3a was synthesized. As it can be clearly noticed, these methods are not able to produce large scale structures such as city blocks. Furthermore, both algorithms have problems in consistently producing connected road courses. CNN-based algorithms are able to produce large scale structure as illustrated in Figure 11b. The road network produced by Gatys et al. [11], however, lacks visual similarity to the original network. In contrast, the approach by Xie et al. [30] is able to produce a visual similar road network and captures that properties found in the original road network.

6 IMPLEMENTATION DETAILS

Our algorithms are implemented in Python and we used the GAN implementation of [17] as a basis for learning the different road network models. However, we changed the original implementation in order to consistently support single channel images. The GAN model for the different road networks is trained on a single NVidia TitanX (Pascal). Each epoch takes 100 iterations with a batch size of 64 and takes about 90 seconds to compute. We trained all the models for at least 100 epochs and decided from a visual examination of samples taken from various epochs which model to choose. The overall training is done in an offline step that takes up to 3 hours. The single



(a) Patch-based texture synthesis methods



(b) CNN based texture synthesis methods

Figure 11: Evaluation of texture synthesis algorithms using the irregular road network illustrated in Figure 3a. In 11a the result of patch-based synthesis algorithms i.e. the approach of Portilla and Simoncelli and the method of Kwatra et al. are illustrated. These methods suffer from producing larger scale structures such as city blocks and connected road structures. In contrast 11b illustrates result from modern CNN based methods, i.e. the algorithm Gatys et al. and the method Xie et al. are able to reproduce connected structure and even city blocks, however, they are only able to produce images of fixed that need additional resizing.

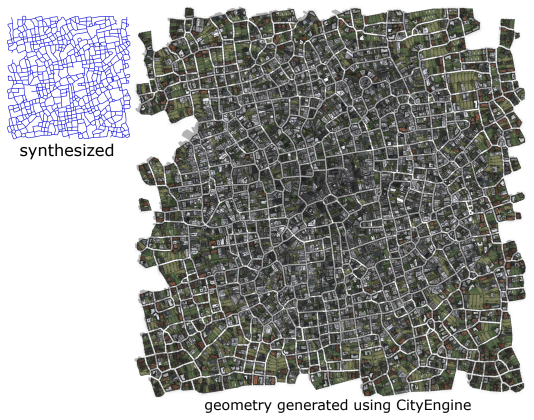


Figure 12: The resulting road network is directly usable in urban planning tools such as CityEngine.

steps of the online synthesis steps takes up to a few seconds. In more detail, the generation of a sample of size 769×769 pixels produced from a tensor $z \in \mathbb{R}^{25\times25\times100}$ sampled from $p_z(z)$, takes on average 0.08 seconds on a single NVidia TitanX (Pascal). The post-processing steps are performed on a Intel Core-i7 5820K, with only a single core in use. Each step takes: for *graph construction*: $1.5s$, for *block computation*: $1.6s$, for *simplification*: $2.0s$ and for *deadend removal*: $0.02s$ in average.

7 CONCLUSION

We have investigated the suitability of GANs for road network synthesis. In order to make it possible to train GAN on road network data we developed a preprocessing step. A post-processing step enabled us to extract a graph-based representation. Our results have demonstrated that GANs are able to produce novel road network patches, that are structurally sound and visually similar, when compared to the input network. Furthermore, we substantiated our results by a statistical evaluation of different road network measures such as *city block area*, *city block compactness*, and *city block aspect ratio*.

During the evaluation of our pipeline, we identified several limitations. First, structures like roundabouts, highway ramps and also roads that are very close to each other are not sufficiently captured during the training. This means that roundabouts or highway ramps cannot be successfully synthesized with our approach. Second, currently we consider all *highway* categories as part of the same street level. We did not succeed in learning models for different street levels, thus, we decided to perform the experiments using only a single street level (cf. Section 4.2). Typically, a road network naturally splits into multiple street levels such as *major* and *minor* roads. Thus, it is necessary to perform an in-depth evaluation of multiple street levels in future work. Furthermore, large-scale road courses are typically present in every road network. Although, these structures are rudimentary present in the synthesized examples shown in Section 5, our post-processing step lacks an additional step to consistently enforce such large-scale structures. One possibility to address this issue, would be fitting curves to road individual courses and enforcing global constraints such as parallelism. Another limitation is that we have only limited control over the output of the generator. In real road networks the road courses are specifically planned to fulfil specific requirements regarding landuse or terrain. Furthermore, the urban planner might also incorporate existing objects into its road design decisions. As our approach is a very first step towards using GANs for road network generation, we did not incorporate such external constraints. However, such constrains are necessary to steer the output of the generator G and leave this for future work as it would exceed the scope of the paper.

There are several interesting directions for future work. First, we would like to add attribute layers e.g. density maps, landuse maps, terrain maps etc.

in order to condition the learning process. This would make it possible to improve controllability of the generator network. Second, we would like to investigate further steps in order to train a GAN model that is able to synthesize multiple street levels. The post-processing needs to be extended to reproduce large-scale structures so that a fair comparison to existing example-based or procedural algorithms for road network generation can be given. Third, we would like to extend our approach and investigate the suitability of GANs to generate building footprints given a predefined city block shape. Finally, we would like to extend the road network generation in terms of a growing based road generation system. Apart from using GANs for urban structures we might also investigate their use for feature map generation for texture synthesis algorithms such as [28, 18].

8 REFERENCES

[1] Daniel G. Aliaga, Carlos A. Vanegas, and Bedřich Beneš. Interactive example-based urban layout synthesis. In *ACM TOG SIGGRAPH Asia*, pages 160:1–160:10, 2008.

[2] Jan Beneš, Alexander Wilkie, and Jaroslav Křivánek. Procedural modelling of urban road networks. *Computer Graphics Forum*, 2014.

[3] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[4] Guoning Chen, Gregory Esch, Peter Wonka, Pascal Müller, and Eugene Zhang. Interactive procedural street modeling. In *ACM transactions on graphics (TOG)*, volume 27, page 103. ACM, 2008.

[5] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[6] Arnaud Emilien, Adrien Bernhardt, Adrien Peytavie, Marie-Paule Cani, and Eric Galin. Procedural generation of villages on arbitrary terrains. *The Visual Computer*, 28(6-8):809–818, 2012.

[7] Arnaud Emilien, Ulysse Vimont, Marie-Paule Cani, Pierre Poulin, and Bedrich Benes. Worldbrush: Interactive example-based synthesis of procedural virtual worlds. *ACM Transactions on Graphics (TOG)*, 34(4):106, 2015.

[8] Esri Inc. Cityengine, 2017.

[9] Eric Galin, Adrien Peytavie, Eric Guerin, and Bedrich Benes. Authoring Hierarchical Road Networks. *Computer Graphics Forum*, 30(7), 2011.

[10] Eric Galin, Adrien Peytavie, Nicolas Marechal, and Eric Guerin. Procedural Generation of Roads. *Computer Graphics Forum (Proc. of Eurographics)*, 29(2):429–438, 2010.

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

[12] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. pages 2672–2680. Curran Associates, Inc., 2014.

[15] Haibin Huang, Evangelos Kalogerakis, ME Yumer, and Radomir Mech. Shape synthesis from sketches via procedural models and convolutional networks. *IEEE Transactions on Visualization and Computer Graphics*, 2016.

[16] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arxiv*, 2016.

[17] Nikolay Jetchev and Roland Bergmann Urs, Vollgraf. Texture synthesis with spatial generative adversarial networks. pages 2672–2680. Curran Associates, Inc., 2016.

[18] Alexandre Kaspar, Boris Neubert, Dani Lischinski, Mark Pauly, and Johannes Kopf. Self tuning texture optimization. In *Computer Graphics Forum*, volume 34, pages 349–359. Wiley Online Library, 2015.

[19] Vivek Kwatra, Irfan Essa, Aaron Bobick, and Nipun Kwatra. Texture optimization for example-based synthesis. *ACM Transactions on Graphics (ToG)*, 24(3):795–802, 2005.

[20] Chuan Li and Michael Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks. In *European Conference on Computer Vision*, pages 702–716. Springer, 2016.

[21] TANVI MISRA. X-ray your city's street network, 2017.

[22] G Nishida, I Garcia-Dorado, and DG Aliaga. Example-driven procedural urban roads. In *Computer Graphics Forum*. Wiley Online Library, 2015.

[23] Gen Nishida, Ignacio Garcia-Dorado, Daniel G Aliaga, Bedrich Benes, and Adrien Bousseau. Interactive sketching of urban procedural models. *ACM Transactions on Graphics (TOG)*, 35(4):130, 2016.

[24] Foundation OpenStreetMap. Openstreetmap, 2017.

[25] Yoav I. H. Parish and Pascal Müller. Procedural modeling of cities. *ACM TOG (Proceedings of SIGGRAPH)*, 19, 2001.

[26] Javier Portilla and Eero P Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *International journal of computer vision*, 40(1):49–70, 2000.

[27] Daniel Ritchie, Anna Thomas, Pat Hanrahan, and Noah D Goodman. Neurally-guided procedural models: learning to guide procedural models with deep neural networks. *arXiv preprint arXiv: 1603.06143*, 2016.

[28] Roland Ruiters, Ruwen Schnabel, and Reinhard Klein. Patch-based texture interpolation. *Computer Graphics Forum (Proc. of EGSR)*, 29(4):1421–1429, June 2010.

[29] Ruben M. Smelik, Tim Tuenel, Rafael Bidarra, and Bedrich Benes. A survey on procedural modelling of virtual worlds. *Computer Graphics Forum*, pages n/a–n/a, 2014.

[30] Jianwen Xie, Yang Lu, Song-Chun Zhu, and Ying Nian Wu. A theory of generative convnet. *arXiv preprint arXiv:1602.03264*, 2016.

[31] Yong-Liang Yang, Jun Wang, Etienne Vouga, and Peter Wonka. Urban pattern: Layout design by hierarchical domain splitting. *ACM TOG (Proceedings of SIGGRAPH Asia)*, 32, 2013.

[32] M. E. Yumer, P. Asente, Mech R., and L. B. Kara. Procedural modeling using autoencoder networks. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pages –. ACM, 2015.

[33] TY Zhang and Ching Y. Suen. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3):236–239, 1984.