Procedural Generation of Roads with Conditional Generative Adversarial Networks

Lin Ziwen Kelvin and Bhojan Anand

November 25, 2020

Procedural Generation of Roads with Conditional Generative Adversarial Networks

Lin Ziwen Kelvin
Computer Science Department
National University of Singapore
Singapore
e0014961@u.nus.edu

Bhojan Anand
Computer Science Department
National University of Singapore
Singapore
banand@comp.nus.edu.sg

***Abstract*—Procedural terrain generation refers to the generation of terrain features, such as landscaping, rivers or road networks, through the use of algorithms, with minimal input required from the user. In the process of game development, generating terrain is often an important part of the game development process. Traditional generation methods are often too time consuming especially with larger terrain maps. On the other hand, procedural methods that generate terrain automatically often do not have much user control over the output. We explore the usage of conditional generative adversarial networks in the creation of road maps, as well as the application of such road maps in the creation of game levels in game development engines such as Unreal Engine 4.**

***Keywords*-Computing methodologies; Machine learning approaches; Computing methodologies; Image processing;**

I. Introduction

Procedural Terrain Generation (PTG) refers to the generation of terrain using algorithms, specifically with little to no human input. This is in contrast to traditional terrain creation, where the landscaping was sculpted by hand, and the secondary features are added on manually. PTG allows for terrain and levels to be generated dynamically without needing humans to create the entire landscape by hand.

Road networks are frequently used in games in some form or another. In city-based games such as Sims or Grand Theft Auto, the game takes place within cities, thus road networks directly influence player navigation. In other games such as Player Unknown's Battleground, road networks allow for safer transport of players, and can potentially affect player strategy through usage of chokepoints. Outside of games, road generation may be of use in urban planning, such as for predicting traffic movements and future urban development. However, to the best of our knowledge, most road network generation methods do not have much customizability, and users are often unable to control the types of roads being generated. We thus explore the usage of conditional Generative Adversarial Networks (cGAN) in creating a framework that allows users to generate road networks that can be used for game development.

We have implemented a framework with a graphical user interface that allows users to provide an initial sketch diagram of primary roads, and the interface outputs a road network based on the initial sketch that additionally contains secondary roads. These road networks can be further exported for use in game engines such as Unreal Engine 4 to create landscapes that are pre-painted with textures corresponding to such road networks.

II. Related Work

Shortest-path algorithms were used by [1] and [2] to generate one single road, while [3], and [4] made use of recursive sub-division to partition an area into a complete road map.

Growth-based algorithms were proposed by [5] and [6], where an initial road network is provided, and a city comprising of the road networks is generated by considering the surrounding terrain. Patch-based algorithms where road patches are selected from an initial list, warped, and added to the generated road network were proposed by [7] and [8]. Generative Adversarial Networks (GANs) were used by [9] in the generation of landscape heightmaps, by using satellite images as the training dataset. In addition, the predicted texturing of the generated terrain for the given heightmap is also generated, which can be imported into the Unity Engine to create landscapes.

[10] proposed using GANs in the generation of road networks. The resulting GAN from [10] however makes use of random noise in generating new tiles, as the training is carried out on a standard GAN. This means that users would either need to provide their own noise input, or rerun the generation several times due to the lack of control, making them similar to traditional PTG algorithms in output controllability.

Conditional GANs were used by [11] in the generation of landscapes. Unlike [9], [11] uses heightmap data from the United State Geological Survey Earth explorer. [11] also features a real-time user sketching tool, that allows users to generate terrain in real time. This framework can be used in the generation of game terrain as it allows for a high degree of customizability.

III. Outline of Approach

Figure 1 describes the overall pipeline used to generate a full road network, given an initial sketching of primary

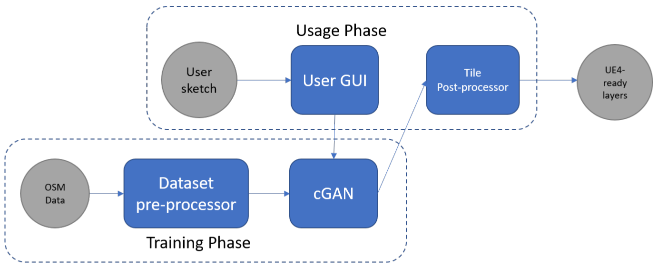


Figure 1: Overall Roadmap Generation Framework

roads. It consists of a data pre-processor, a conditional Generative Adversarial Network, a simple graphical user interface to input the initial network, and a post-processor that formats the generated network for use in game engines such as Unreal Engine 4.

A. *Dataset Pre-processor*

OpenStreetMap tiles were selected as the dataset for training the cGAN. OSM map tiles take the format of a 256 x 256 PNG image file, where each pixel is coloured to match the corresponding features present. OSM map tile generation is governed by user-defined rulesets, which dictate which and how features are generated.

The reasons for choosing OSM street tiles as the training dataset are as follows:

1) Tile data is based on real-life street directories. Terrain that is generated using such data is likely to be realistic and accurate with respect to the data provided.
2) Tile data is available easily online, making it easy to train the cGAN using tile data from different regions, such as America, Asia, or Europe.
3) Tile data is highly customizable. Using stylesheets and user-defined rulesets, the user can pick and choose the type of features to be rendered on each tile. This allows for the cGAN to learn how to generate different types of tiles, by using different rulesets.

The dataset pre-processing first involves removing irrelevant details, such as text and landmark icons by using a ruleset. Using this ruleset, only water bodies, forests, primary roads, secondary roads and buildings are generated, and the background is coloured as white. The primary roads are coloured as red, secondary roads as blue, and all buildings coloured as black, in contrast to the default colouring style used by OSM. This colouring was used, as in the default colouring scheme, several features shared similar colours, and the resulting cGAN was unable to distinguish between them clearly, thus generating incohesive results.

The tiles are next generated using Maperitive, a free software that allows all the tiles to be generated in a mbtiles format file. While it is possible to extract PNG files directly using self-made scripts, such scripts involve querying the OSM database directly, which is both time-consuming and cannot be customized using rulesets. The resulting mbtiles file is subsequently broken down into PNG files using the mbutil library.

The process is then repeated with a ruleset that does not generate any secondary roads or buildings. Thus, for every tile in the map, two tiles are generated, one that contains only the primary roads, hence referred to as the base tile, and one that contains primary roads, secondary roads, and buildings, hence referred to as the target tile.

B. *cGAN Architecture and Training*

The architecture for the cGAN comprises of a discriminator network d, and a generator network g. The cGAN is implemented using the Keras and TensorFlow libraries, and is largely adopted from the Pix2Pix network proposed by [12].

The discriminator network d takes in a $< generatedtile, targettile >$ pair and assigns it an accuracy score from 0 to 1. This score is based on how similar the target tile is to the training dataset in general. It is implemented as a standard Convolutional Neural Network, where a 4 x 4 sized window is used as a filter. MaxPooling and BatchNormalization is carried out on each layer. To prevent overfitting, a dropout rate of 0.4 is applied in the first Conv2D layer, and the binary cross-entropy is used for the loss function.

The generator network takes in a base tile i, and outputs a generated tile $g(i)$. The generator is implemented as an encoder-decoder network with skip connections, with a 4 x 4 sized window as a filter. Each encoder block consists of a Convolutional layer with BatchNormalization, and each decoder block consists of a transposed Convolutional layer. Dropout is applied in the decoder blocks to prevent overfitting.

```
for training cycles do
    x = select random batch;
    y = target output(x);
    out = generator(x);
    realScore = trainDiscriminator(x, y, targetScore = 1);
    fakeScore = trainDiscriminator(x, out, targetScore = 0);
    updateModel(concat(realScore, fakeScore));
end
```

In each training cycle, a batch of data is randomly selected from the training dataset, and first passed to the generator network. The resulting output tiles are concatenated with the base tiles, and passed to the discriminator. As these images are generated, the target score for these images would be 0. After this, the base image along with its target image are sent to the discriminator for grading, with a score of 1 as the target score. The resulting scores are then fed to the generator network as its loss function, as the generator aims to maximise s from the discriminator.

This is carried out for 5000 training cycles. At every 100 cycles, the model is evaluated by testing it against a fixed test set, and the model set is saved. Saving the model set in intervals allows us to backtrack to an earlier model, in case overfitting occurs.

C. Graphical Interface and Post-Processing

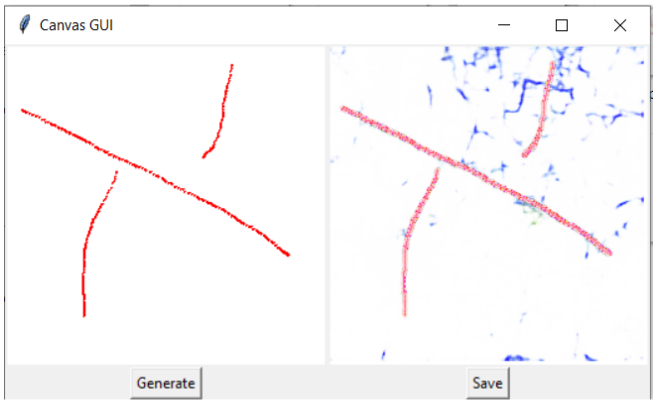


Figure 2: GUI for user-defined roads

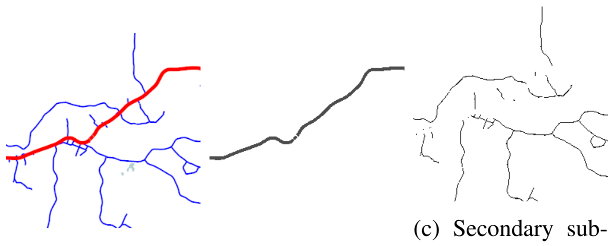
To allow for game developers to create their own road networks, we also provide a simple graphical user interface, implemented using the tkinter library. The GUI comprises of a drawing canvas, where the user uses their mouse to draw the initial primary road network. The user can then select the 'Generate' option to see what kind of road network the cGAN produces, and if they are satisfied with the outcome, save the generated tile for further use, be it to use as a reference image, or for use in Unreal Engine 4.

The user may choose to convert a generated tile from the GUI for use in Unreal Engine 4. Landscapes in Unreal Engine 4 can comprise of different weight layers for different material types. For example, one landscape can consist of a grass texture layer, a primary road texture layer (asphalt), a secondary road texture (dirt) layer, and building location layers (gravel). The engine allows for user-imported PNG files to populate the weight layers. The post-processor thus takes in the generated tile from the cGAN, and splits it into 3 different subtiles, each encompassing different aspects, namely the primary road, secondary road, and building locations from the tile. These subtiles are created by isolating the respective colours of each feature from the tile. Finally, the subtiles are converted into grayscale and saved.

Figure 3 shows the generated subtiles for a given base image. The user can then use these subtiles as the weight layer in UE4 landscape creation.

IV. Results

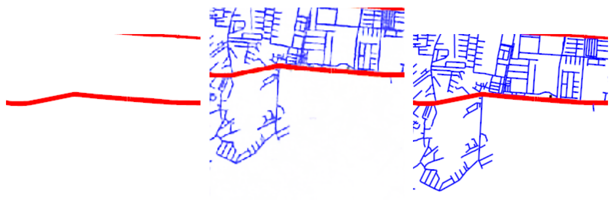
Our results show that a cGAN can be used to effectively generate full road networks from an initial sketch.



(a) Base Image (b) Primary subtile (c) Secondary subtile

Figure 3: Subtiles created from post-processor

To evaluate the performance of the cGAN, testing data was provided in the form of input tiles that were drawn from the same region as the dataset, but not used for training, as seen in Figure 4. The base tiles for the testing data were fed into the generator network, and the resulting generated tile was compared to the ground truth, which is the target tile.



(a) Base Image (b) Generated Image (c) Ground Truth

Figure 4: Generation of tiles

Using the graphical interface, user-drawn sketches were input to simulate the user drawing, and the subsequent weight layers were input into a fresh Unreal Engine 4 level asset. Using this landscape, foliage was planted into the map, and cuboid polygons were placed into spots designated as building locations, represented by the orange textures, to simulate the placement of buildings.

From this and other free assets provided in Unreal Engine 4, a simple driving simulator was created using the generated landscape as a basis. A sample screenshot from the driving simulator is shown in Figure 5.

We also compare the results from the tile generator framework to existing work, namely those proposed by [10] and [6]. We compare our results with the results by [10] as they use a similar approach by using GANs on the same dataset of OSM tiles. While the work proposed by [6] does not make use of GANs in any way, the algorithm proposed by [6] takes in a pre-existing primary road network and generates secondary roads on this initial road network. This makes it similar to the objective of this report, which is to allow for greater user control over the output from tile generation. As seen from Figure 6, One noticeable difference is that [6] takes in account of nearby water bodies and is able to generate bridges minimally to work around the terrain, which cannot be done by tiles generated from our

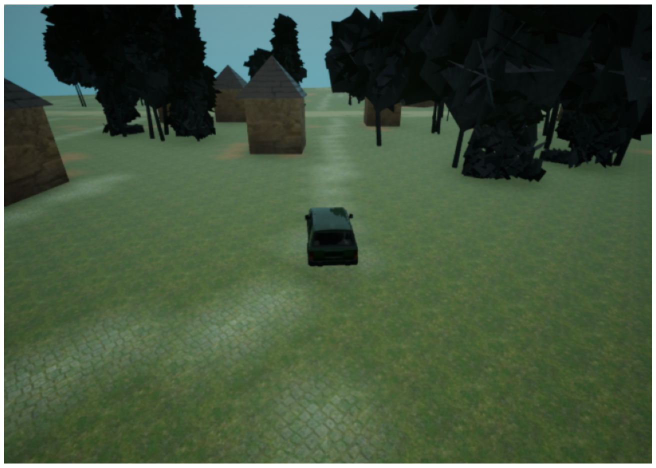
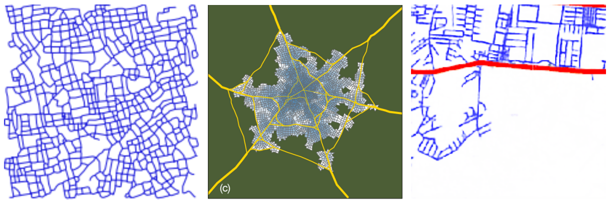


Figure 5: Driving Simulator created in Unreal Engine 4 using the generated tiles



(a) StreetGAN tiles (b) Generated city (c) Generated Tiles

Figure 6: Comparison of generated tiles between different algorithms. (a) from [10], (b) from [6]

framework, as the cGAN was not trained on tiles with water bodies. [10] made use of road styles from multiple regions in their work, while our framework is specific to one region. Thus, [10]'s work is able to generate tiles on a more general need, while we believe that our framework can generate tiles on a more specific need, such as if the tiles are to be generated with a style from a specific region.

V. Conclusion and Future Work

By using a Conditional Generative Adversarial Netowrk, we were able to create entire road networks from a simple initial input of primary roads. Users can also make use of a GUI to provide their own input sketches to generate their own tiles. These generated tiles were able to be adapted for use in Unreal Engine 4 in the creation of game landscapes.

For future work, the cGAN can be further trained on datasets with elevation data, as well as datasets with water bodies present. The presence of water bodies would affect how roads are generated in real life, and as such, would allow for even more realistic generation. Water bodies such as lakes and rivers are also often featured in games as a natural obstacle, and as such, allowing the user to define their own water bodies would allow for a wider range of tiles to be generated.

References

[1] E. Galin, A. Peytavie, N. Marchal, and E. Gurin, "Procedural generation of roads," *Computer Graphics Forum*, vol. 29, no. 2, p. 429438, 2010.

[2] C. Martek, "Procedural generation of road networks for large virtual environments," Master's thesis, Rochester Institute of Technology, 2012.

[3] R. Sharma, "Procedural city generator," *2016 International Conference System Modeling & Advancement in Research Trends (SMART)*, 2016.

[4] D. Gonzlez-Medina, L. Rodrguez-Ruiz, and I. Garca-Varea, "Procedural city generation for robotic simulation," *Advances in Intelligent Systems and Computing Robot 2015: Second Iberian Robotics Conference*, p. 707719, 2015.

[5] Q. Yu and A. Steed, "Example-based road network synthesis.," in *Eurographics (Short Papers)*, pp. 53–56, 2012.

[6] J. Beneš, A. Wilkie, and J. Křivánek, "Procedural modelling of urban road networks," in *Computer Graphics Forum*, vol. 33, pp. 132–142, Wiley Online Library, 2014.

[7] E. Teng and R. Bidarra, "A semantic approach to patch-based procedural generation of urban road networks," in *Proceedings of the 12th International Conference on the Foundations of Digital Games*, pp. 1–10, 2017.

[8] G. Nishida, I. Garcia-Dorado, and D. G. Aliaga, "Example-driven procedural urban roads," in *Computer Graphics Forum*, vol. 35, pp. 5–17, Wiley Online Library, 2016.

[9] C. Beckham and C. Pal, "A step towards procedural terrain generation with gans," *arXiv preprint arXiv:1707.03383*, 2017.

[10] S. Hartmann, M. Weinmann, R. Wessel, and R. Klein, "Streetgan: Towards road network synthesis with generative adversarial networks," 2017.

[11] É. Guérin, J. Digne, E. Galin, A. Peytavie, C. Wolf, B. Benes, and B. Martinez, "Interactive example-based terrain authoring with conditional generative adversarial networks," *Acm Transactions on Graphics (TOG)*, vol. 36, no. 6, p. 228, 2017.

[12] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017.