

# 3D Environment Generation Using Conditional Adversarial Networks

Tim Whitaker  
Colorado State University

December 9, 2019

## 1 Abstract

We introduce a method and application for generating 3D environments using conditional generative adversarial networks. The outputs from our networks are able to be imported into popular three dimensional modeling software like Unreal, Unity and Blender. We've created a simple web based interface to allow a user to control the output of the generated environment by painting with predefined brushes that represent terrain varieties: mountains, hills, terraces, plains, and rivers.

## 2 Introduction

This project explores the usage of conditional generative adversarial networks to synthesize 3 dimensional environments.

The motivations for this project are to make it easier for game, world or environment designers to create realistic environments procedurally while giving the designer more control over the results.

We aim to create a simple and easy to use GUI interface for a user to paint a 2 dimensional semantically significant image map. We then pass the user created image map through a generative adversarial network trained on elevation data gathered from the Earth in order to generate a realistic terrain heightmap that approximately represents the user defined map. The resulting heightmap produced from the neural network can then be uploaded into any of the popular 3 dimensional rendering softwares in-

cluding Unity, Unreal, Blender or World Machine.

This approach offers a promising avenue to co-creative design. Humans working together with machine learning programs could allow for more creative results with reduced cost and effort for the designer. The flexibility of being able to tweak and tune the training set also allows for the possibility of adapting this approach to content of many different kinds, not just 3d environments.

## 3 Background

### 3.1 Procedural Content Generation

Procedural content generation is an algorithmic and stochastic process used in art, games, music and film. As a paradigm for content generation, it is used to increase replayability, variability, flexibility and reduce production cost, effort and storage space [9]. These algorithms are crafted by hand and they require a lot of tuning by a developer to ensure that the random perturbations result in usable content.

Procedural content generation has been used in production applications since 1980 with the dungeon crawler Rogue demonstrating its effectiveness. [10] Procedural generation has been a staple in gaming since and today it continues to be featured in some of the top games.

The addition of machine learning to procedural generation is a new topic in research. Most projects are using 2 dimensional platformers, but real time strategy and 3 dimensional environments are also be-

ing explored. This field of research has its own challenges, particularly the comparability of results, but efforts are being made to improve benchmarks and datasets [9].

### 3.2 Generative Adversarial Networks

Neural networks allow for a powerful model of learning patterns from complex datasets. Generative adversarial networks work by synthesizing novel data according to patterns it learns in the training set. They work by training two neural networks in competition with each other, a generator and a discriminator. The generator network generates new data instances and the discriminator classifies the results of the generator according to how authentic the generated images are given some training distribution. These two networks train in tandem, and over time, the generator gets good at producing novel images that look like they belong to the training set.

We evaluate two implementations of a specialized form of generative adversarial networks designed to perform conditional image to image translation. The two models we are using are called Pix2Pix and CycleGAN.

Both Pix2Pix and CycleGAN are conditional networks based on the deep convolutional generative adversarial network architecture proposed by Radford et al. in 2015 [6].

The objective function of a conditional generative adversarial networks can be expressed as:

$$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{x,y}[\log(D(x, y))] + \mathbb{E}_{x,z}[\log(1 - D(x, G(x, z)))]$$

Where  $G$  is the generator network that is attempting to minimize this objective and  $D$  is the discriminator network that tries to maximize it.

#### 3.2.1 Pix2Pix

Pix2Pix introduces a generalized approach to image to image translation problems. Their model learns not only a mapping between input and output images, but it also learns a loss function between the

two image datasets. This allows this model to be used for a variety of image tasks that would originally need very different model and loss configurations [2].

Pix2Pix found it beneficial to add a traditional loss term as well, namely L1 distance. The discriminators job is the same as a traditional GAN, but the generator now is tasked with fooling the discriminator as well as producing results near the ground truth output.

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[||y - G(x, z)||_1]$$

The addition of that loss term can be tuned with a parameter  $\lambda$ . This combined objective function is used for training Pix2Pix.

$$G^* = \arg \min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G)$$

#### 3.2.2 CycleGAN

CycleGAN introduces cycle consistency loss to conditional generative adversarial networks. It works by measuring the loss of a mapping from one distribution to another and then back to the original distribution again. This approach allows for good generalization and the advantage of not needing specifically paired images in the training distributions. This can save a lot of time in building large datasets, but it can lose out on the specificity offered by Pix2Pix [12].

Where pix2pix trains one GAN (one generator and one discriminator), CycleGAN trains two, (Generator A and B and Discriminator A and B). learning the mapping between the two GANs is the trick to getting unpaired image translation to work. CycleGAN introduces a cycle consistency loss. The idea is that for each image in the  $x$  domain, the image translation cycle should be able to bring the  $x$  back to the original image,  $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$ .

$$\mathcal{L}_{cyc}(G, F) = \mathbb{E}_{x,y}[||F(G(x)) - x||_1] + \mathbb{E}_{x,z}[||G(F(y)) - y||_1]$$

$$\begin{aligned}\mathcal{L}(G, F, D_X, D_Y) = & \mathcal{L}_{GAN}(G, D_Y, X, Y) \\ & + \mathcal{L}_{GAN}(F, D_X, Y, X) \\ & + \lambda \mathcal{L}_{cyc}(G, F)\end{aligned}$$

### 3.3 Related Work

#### 3.3.1 Nvidia’s GauGAN

GauGAN was the inspiration behind the semantic user map. This research demonstrated an application that took a 2 dimensional semantic image drawn by a user, and translated it into a photorealistic image. This project also used Pix2Pix as a base for their network architecture, but they also introduce a novel normalization layer called spatially-adaptive normalization. The idea is that normal layer normalization tends to wash out semantic meaning from an image. By considering the spatial properties, you can keep the semantic information in tact while still performing normalization. [3]. Nvidia hosts an excellent demo of the project at <http://nvidia-research-mingyuliu.com/gaugan>.

#### 3.3.2 A Step Toward Procedural Terrain Generation with GANs

This paper had a very similar idea to this project. They use a standard generative adversarial network to synthesize terrain heightmaps with a model trained on Earth data. They use NASA’s visible earth project for their training image data at a resolution of 1km per pixel. This project appears to be quite introductory and they lay out a good overview of how a standard GAN could be implemented to generate environments. We expand on their work by introducing the conditional GAN and integrating co-creative design by including a translation from a user defined segmentation map. [1]

## 4 Methodology

### 4.1 User Interface Application

The user interface is a web based javascript application that allows a user to paint a semantically significant image using predefined brushes. We include tools to change the brush size and export or clear the image. The canvas size is 512 x 512 pixels, and these images are resized automatically when passed to our neural network.

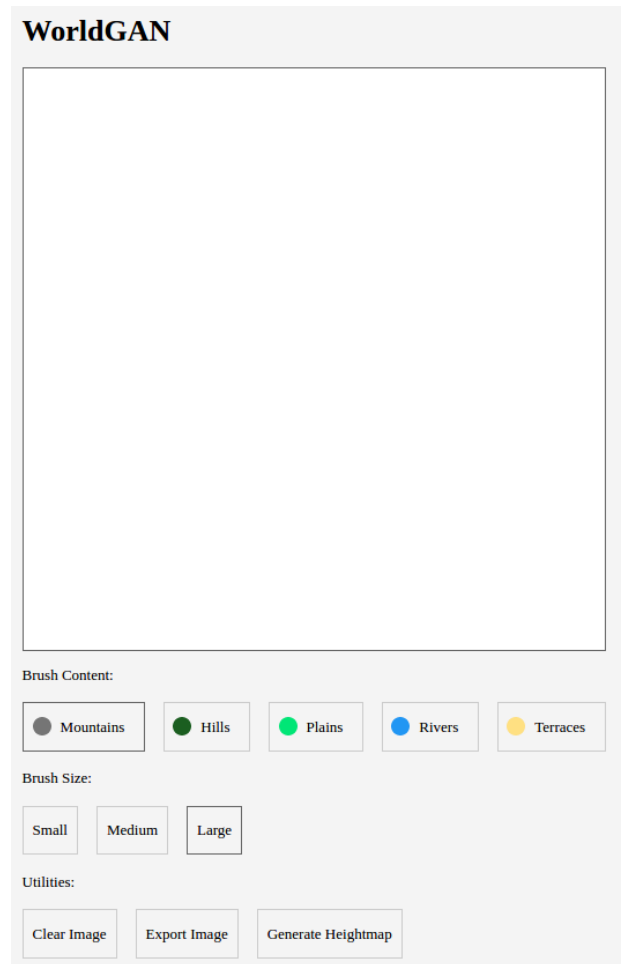


Figure 1: The user interface application

The backend is powered by a Flask API. We set up

an endpoint to be able to evaluate a drawn image on our trained models. In order to pass the usermap and generated heightmap back and forth between the client and the server, we use base 64 encoding and decoding on the image. This way we can save the images to the file system and have a local process run the neural network and return the generated heightmap. This direct access to the file system has security concerns, so it's only recommended to run the app locally.

## 4.2 The Dataset

Generative adversarial networks need a good amount of training data to output good results and a key objective of this project is to generate realistic results. This leads to an interesting challenge of putting together a realistic yet sizable dataset. In order to achieve this, we use terrain data of the earth as opposed to fictional or manufactured terrain data. In the next two sections we go into the methodology behind building the dataset.

### 4.2.1 Training Data

We use an interactive open source map tool that visualizes elevation data gathered from Mapzen's global elevation service [7].

We then built a program that controls a headless browser using an open source library developed by Google called puppeteer. This script loads a random coordinate, checks if the coordinate is over land and if so, then we save an image of the elevation data at that coordinate. We resize the saved image to be 512 x 512 pixels.

We then manually build the usermap training set using our user interface application. For each image in the heightmap training set, We paint a usermap to represent that heightmap, using the semantically significant brushes that most closely match the terrain (mountains, hills, plains, rivers, and terraces). This process is a subjective one and all usermaps were painted by myself. This does bias the model to produce images according to how I read the heightmaps.

After this process we are left with 258 paired

usermaps and heightmaps. We group all of these together into a batch to be trained on.

Creating the usermaps by hand is quite time consuming, and I wanted to experiment with models trained on a large dataset. We accomplish this by augmenting our first batch with simple transformations in order to artificially increase the number of images we can train on. Image augmentations are a well researched and effective approach to improving neural network performance [5].

We performed 5 operations on each usermap and heightmap. We mirror horizontally, flip vertically, rotate 90 degrees, rotate 180 degrees, and rotate 270 degrees. All of the resulting images are gathered into a second batch and we end up with 1548 paired images (a 5x increase).

Before training the networks, we need to perform one more preprocessing step. Pix2Pix requires paired images to be combined into a single side by side image. We stack each pair of usermap and heightmap and save these images into separate batches specifically for Pix2Pix. Each resulting image is 1024 x 512 pixels. CycleGAN does not need this preprocessing step and can be trained on images in different directories.

### 4.2.2 Test Data

The test data is a collection of 10 usermaps I drew that are designed to be representative of a variety of environments a user might draw. These 10 environments include mountainous regions, hilly regions, flat regions and combined regions.

## 4.3 Training The Models

We trained a total of 4 models. Pix2Pix and CycleGAN on both the original and augmented batch of training data. The training was performed on my desktop GPU enabled PC. The operating system is Ubuntu 18.04. We train on a Nvidia GTX 1080ti GPU with 11 GB of memory, an Intel i7 8700k processor, a 500GB solid state hard drive, and 32 GB of RAM.

Both models are built on top of the PyTorch framework [4] and are trained for 200 epochs. We use the

open source repo implemented by the Jun-Yan Zhu, who published the research on both models [11].

#### 4.3.1 Pix2Pix

Our pix2pix model contained a total of 57.183 million parameters. The generator network contains 54.414 million parameters and the discriminator contains 2.769 million parameters. The original batch of data took approximately 45 minutes to train. The augmented batch took approximately 4 hours to train.

#### 4.3.2 CycleGAN

Our cyclegan model contained a total of 28.286 million parameters. Generator A and Generator B each contain 11.378 million parameters. Discriminator A and Discriminator B each contain 2.765 million parameters. The original batch of data took approximately 4 hours to train. The augmented batch took approximately 24 hours to train.

### 4.4 Visualizations Using 3D Software

The resulting heightmaps can be imported into popular 3d modeling software including Unity, Unreal, Blender, and World Machine. For the purposes of visualizing these results, and because of the limited support for Linux in all of the previously listed softwares, we visualize results using a demo application called Fractscape. Fractscape includes elevation based texture mapping which happens to work very well for visualizing our terrain elevation data. Unity, Unreal, Blender and World Machine have a different workflow for painting textures, so using the generality of Fractscape works best for our visualizations [8].

## 5 Results

Subjective results of networks show that pix2pix performs much better than cyclegan. For the specific task of segmented image translation, paired training data allows the network to learn the meaning of the segmentations much better. Cyclegan had issues of keeping the semantic meaning in tact. Even though

it does appear to generate some of the patterns correctly, it's often mapped to the wrong brush or user defined section.

Both models trained on the augmented batch produce much higher fidelity images. Both pix2pix models produce usable and realistic terrain.

We also find that the model does not perform very well on usermaps with rivers. We found that rivers tend to produce noisy results and sometimes the rivers aren't defined enough to be noticeable in the results. I think this can be solved with more training data including rivers. Due to the nature of how the training data was gathered, it's likely that we didn't get enough examples, and due to the subjective nature of myself reading the earth data and drawing the usermaps myself, I could have read them wrong and not painted them as detailed as they needed to be.

Terraces also had a lack of training data. Due to the specificity of their appearance, they were able to produce better results than rivers. The models will likely struggle on terraces with odd shapes.

Mountains, hills, and plains all performed very well. There was plenty of samples in the training for these features and the areas for each tend to be large. This allowed the networks to do a great job of modeling these terrain features.

### 5.1 User Preference Survey

We also created a small user survey to measure user preference between the 4 trained models. The survey was conducted through google forms and sent to the students in CSCI 567 (3D User Interfaces) through the piazza message board. I took a sample of 5 user maps, and for each one, I produced the output for each model and listed those as multiple choice answers.

## 6 Conclusion

### 6.1 Future Work

Expanding this idea to include texture maps could be an interesting direction to go. Using this same method, it should be possible to translate a given

heightmap into a texturemap. This could allow the designer to have control over generated styles like barren desert scenes vs dark horror scenes vs fantastical colorful scenes. Gathering training data of texturemaps would likely be the bottleneck.

Modifying the network to perform real time generation would be an interesting addition. This would give the user a more interactive experience in designing the environments. Current latency with this implementation is not fast enough so new network architectures or approaches would be needed.

Tunable parameters would also be an interesting addition. Giving the user even more control over the results would lead to a better experience. I can imagine parameters like intensity, noise, smoothness could result in a lot of usable outcomes.

Expanding the training set with different means. Needing to draw all the usermaps for the training set is time consuming and error/bias prone. Coming up with a programmatic way to create the usermap training set could have a large impact on the quality of the results.

## References

- [1] Christopher Beckham and Christopher Pal. A step towards procedural terrain generation with gans, 2017.
- [2] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2016.
- [3] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization, 2019.
- [4] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.
- [5] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning, 2017.
- [6] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.
- [7] Peter Richardson. Tangrams heightmapper, 2016.
- [8] Star Scene Software. Fractscape, 2006.
- [9] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgrd, Amy K. Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. Procedural content generation via machine learning (pcgml), 2017.
- [10] Wikipedia contributors. Procedural generation — Wikipedia, the free encyclopedia, 2019. [Online; accessed 9-December-2019].
- [11] Jun-Yan Zhu. Pytorch pix2pix and cyclegan, 2017.
- [12] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2017.


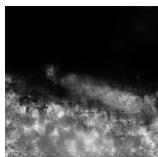
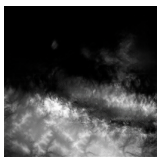
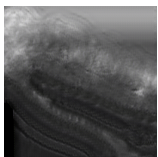
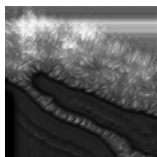

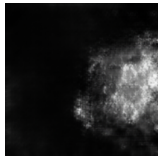
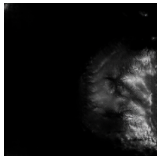
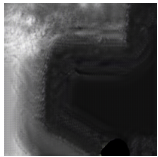
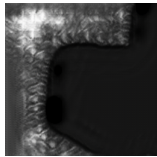


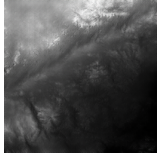
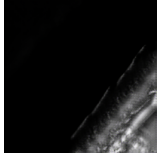
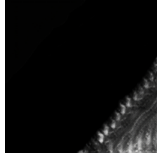

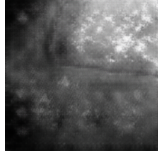
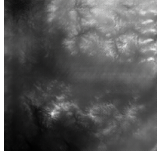
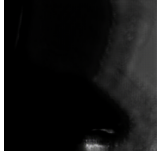


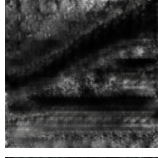
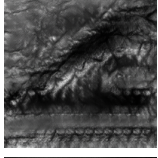
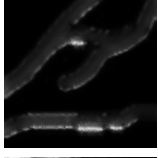
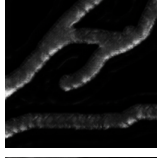

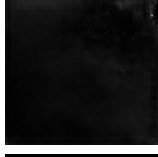
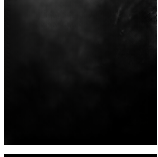
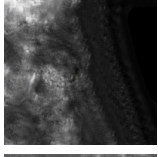
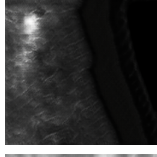


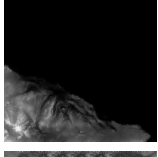
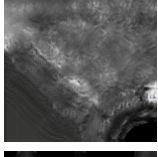
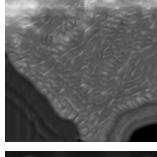

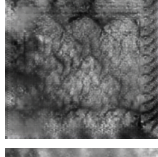
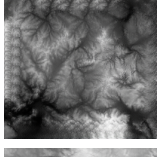
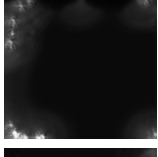
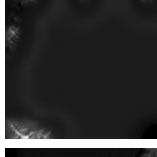

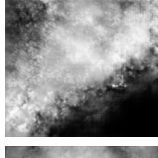
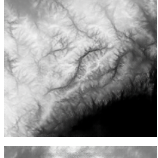
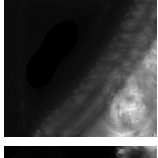
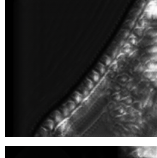

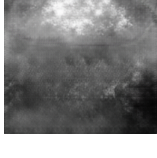
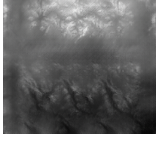
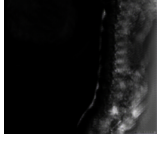

Test Image	Original Pix2Pix	Augmented Pix2Pix	Original CycleGAN	Augmented CycleGAN
				
				
				
				
				
				
				
				
				
				

Table 1: Test images and the outputs from all of our models