3D Environment Generation Using Conditional Adversarial Networks

Tim Whitaker
Colorado State University

December 8, 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 including Unity, Unreal, Blender or World Machine.

3 Background

This project explores the new paradigm of 3 dimensional procedurally generated content via machine learning.

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 storage space [6].

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. We evaluate two models of conditional generative adversarial networks to achieve image to image translation.

We are going to 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 based on the deep convolutional generative adversarial network architecture proposd by Radford et al. in 2015 [4].

The objective function of a conditional GAN 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 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 [1].

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 λ. 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 generativeadversarial 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 [7].

CycleGAN introduces a cycle consistency loss to their objective function. 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, $x \rightarrow G(x) \rightarrow F(G(x)) \approx x$.

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

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

3.3 Related Work

Nvidia GauGAN.

4 Methodology

In this section we go over all of the pieces of work that went in to this project.

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.

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.

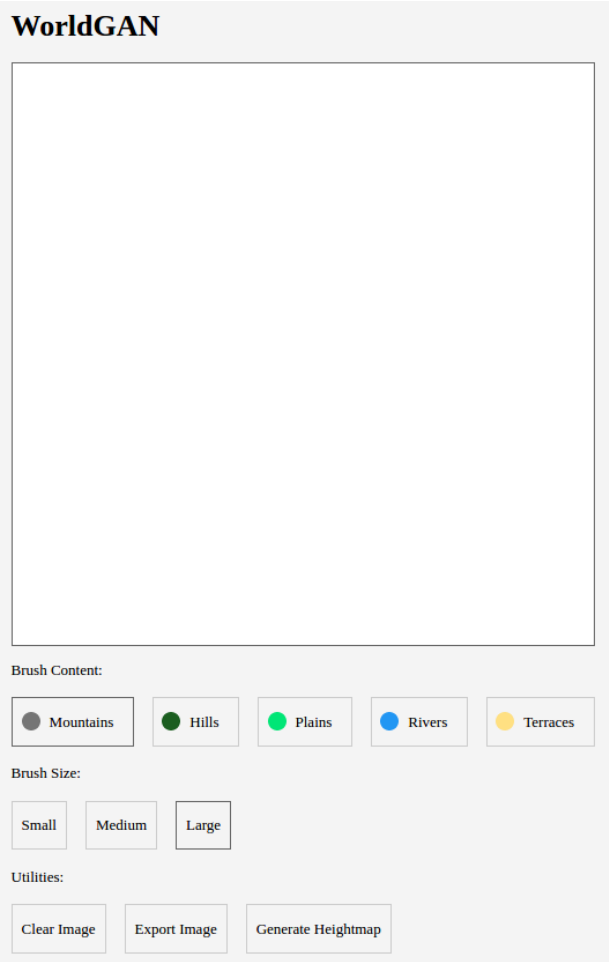

Figure 1: The user interace application

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 [5].

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 appoach to improving neural network performance [3].

We performed 5 operations on each usermap and heightmap. We mirror horizontally, flip vertically, rotate 90 derees, 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 spcifically 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 direc-

tories.

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 relevant specifications include an 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 [2]. Both models are trained for 200 epochs.

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.

5 Results

5.1 Visualizations Using 3D Software

6 Conclusion

6.1 Future Work

References

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

[2] 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.

[3] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning, 2017.

[4] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.

[5] Peter Richardson. Tangrams heightmapper, 2016.

[6] 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.

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