CO – 313

COMPUTER GRAPHICS PROJECT REPORT

Non-Algorithmic Fantasy Map Generation

RAMANSH GROVER | RACHIT SAKSENA
(2K18/CO/281)   (2K18/CO/265)

Under the supervision of
DR. S.K. SAXENA

Contents

1. Objective (Motivation) 2
2. Related Works (Literature Review)..2
2.1. Procedural Content Generation 2
2.2. Deep Generative Networks 4
3. Methodology (Contributions) 6
3.1. Data Acquisition 6
3.2. Formulation...7
3.3. Experimentation..8
3.4. Visualisation 8
3.5. Conclusions & Future Work..9
4. Source Code 10
5. References..10

1. Objective (Motivation)

In Computer Graphics and Virtual Environment Development, a large portion of time is devoted to creating assets – maps being one of them – as they usually form the basis of these large graphical worlds. Traditionally, the “procedural” generation of maps is accomplished by employing smartly designed (but handcrafted) algorithms that can create impressive texture renders that look visually stunning, however, can appear somewhat simple and repetitive due to mathematical constraints and hence, require human intervention.

So far, maps have been procedurally generated through a host of algorithms designed to mimic real-life cartography. Given the advent of Deep Learning, it would only seem like a relevant and exciting endeavour to leverage the power of generative networks (such as Generative Adversarial Networks (Goodfellow et al., 2014)) to learn algorithms to automatically generate maps, without the need to manually write algorithms to do so. Therefore, We aim to create a fantasy map generator utilising GANs to the most of their potential for effectively automating the generation process (without human involvement) and developing non-algorithmic map generators. This project is publicly available on Github[1].

2. Related Works (Literature Review)

2.1. Procedural Content Generation

Procedural Content Generation (PCG) in video games and Computer Graphics Literature is defined as the algorithmic generation of content intended to increase replay value through interleaving the gameplay with elements of unpredictability. It is based on the idea of “building-as-you-are-going” and recent noteworthy video game titles, whose premises are almost entirely based on procedural world generation, have utilised and dominated the video game markets because of the large expanses they are able to generate abiding by this very notion. Some examples include ‘Minecraft’, a game where the player can explore a vast open world sandbox whose environment is entirely based on voxels (‘volumetric pixels’), allowing the player to manipulate their surroundings (i.e. dig tunnels, build walls) and explore interesting terrains (such as beaches, jungles, caves, etc.) and 'No Man’s Sky’, featuring 18 quintillion life-size planets, each relying heavily on procedural generation algorithms in order to determine planet size, gravity, flora-fauna characteristics, etc.

PCG has been around since the 1980s, when algorithms were specifically designed to synthesize real-life entities. One of its use cases involves designing smart algorithms to generate height maps which act as meshes and which are rendered as phenomenal 3-D virtual environments. Some prominent examples include the use of Perlin Noise (Perlin, 1985) and the Diamond Square Algorithm (Fournier et al., 1982). Perlin Noise generates a greyscale image of a height map from an induced noise source (with intensities proportional to heights above sea level), which, when rendered in 3D as a mesh, produces a map. It is arguably the most versatile at generating procedural height values, with the ability to continuously produce smooth transitions of height values at any point on a surface, albeit quite fast, using Perlin Noise entirely on its own can synthesise non-realistic and considerably simple height maps, due to the repeating non-structured noise features.

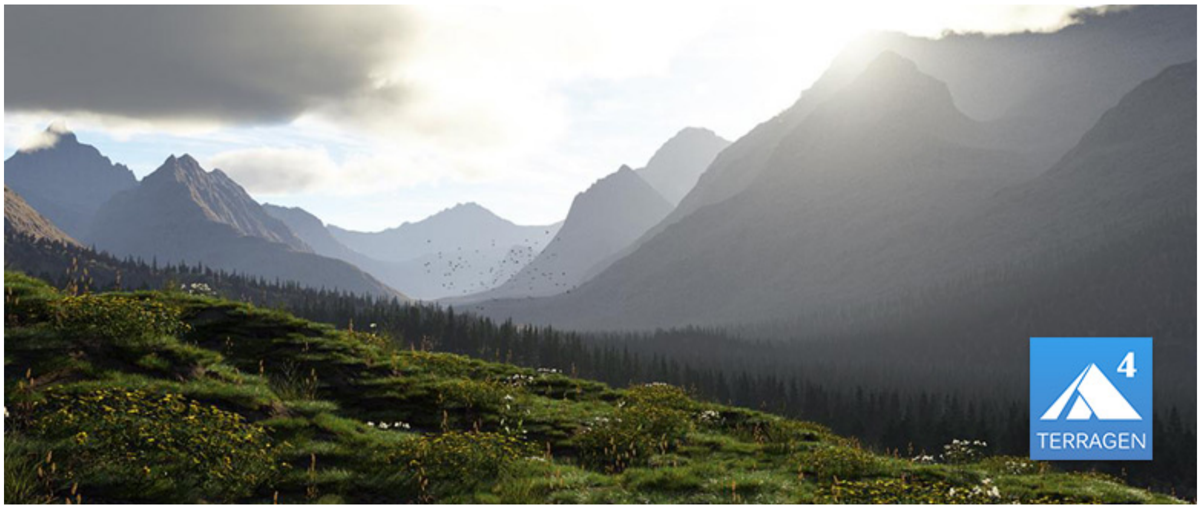

Figure 1: 3-D Environment Procedurally Rendered by Terragen, a proprietary software developed by Planetside Software LLC

Usually Fractal Brownian Motion (FBM) (Mandelbrot and Ness, 1968) is used to control the intensity of differing noise layers, sampling and averaging points from Perlin Noise over multiple levels of varying height and frequency creates a height map of increasingly detailed but less noticeable features. On the other hand, the diamond-square is a more algorithmic approach which avoids the need for a noise based reliance. By averaging points continuously between certain seeded points on a grid, this method can produce highly detailed fractal terrains that can mimic mountain ranges within the real world. This type of algorithm can generate non-uniform distributed structures, though with an element of randomness within the seeded points, can be equally as difficult to control the resulting outputs. In contrast, modern softwares such as Terragen, L3DT, Bryce, and E-on Vue feature a host of sophisticated algorithms which let the user decide the kind of terrain they desire, (e.g. mountains, lakes, valleys), and while these can produce very impressive renders (Figure 1).

Although the generated content is of a higher quality, all these methods are human aided, 'tailoring' algorithms to generate height maps and the algorithms are burdened with the added expense of human supervision and the lack of direct control over the ability to create terrain-based variants on a target objective. Certain game developers have expressed the need for additional tools for generating terrain. The team behind Hell blade, at a Keynote talk at the IGGI conference on games 2018 (Antoniades, 2018), expressed the need for more interesting and interactive world map environments that can be generated with little input through follow up questions to their talk. Another example is an author of a highly detailed 2D map generation tool called Here Dragons Abound expressing concerns with the level of "interesting" generations Perlin Noise is capable of producing, with users soon becoming bored of the repetitive outputs that the noise based algorithms can generate.

2.2. Deep Generative Networks

Deep Learning or Deep Neural Networks allow the processing of data through multiple defined layers (a neural network) of varying computational functions that learn to create a representation of the input data using backpropagation to update weights at nodes to reduce the loss between true and predicted values.

The recent explosion of deep learning has affected a large portion of technology industries, with applications in the games field looking at a varying degree of generation methods. Generative Adversarial Networks (GANs) provide a unique framework that utilizes two deep neural networks: a generator (G) network which attempts to capture the distribution of the training data, mapping this on to an input of latent noise, and a discriminator (D) network that will estimate the probability of its input being from the original training data or from the generators “recreated/forged" output, essentially a discrete multilayer perceptron classifier. The power of adversarial networks lies in being able to mimic any distribution of training data from a nearly endless amount of domain areas. Mapping these learned weights within the generator network back on to a latent vector of noise, upscaling the size with each layer of the network, to finally produce an output based on the trained network’s unsupervised inputs.

Variational Autoencoders, a type of unsupervised learning method, has been used to generate levels (Jain et al., 2016), visually changing graphics outside of game engines using convolutional filters to change the colours to something more understandable to the user (Guzdial et al., 2017). Also improved texturing of 3D environments with the use of GANs (Klein et al., 2017). Lastly a study on level generation was conducted using GANs (Volz et al., 2018). Based on these examples there is an obvious push for more content being generated using these unsupervised methods, requiring little input once a model has been fully trained, though there is little exploration of using machine learning methods to generate the terrain for 3D environments.

A similar task and the original idea of generating terrains using GANs from regions of the planet was first mentioned in (Beckham and Pal, 2017) where the authors attempted to synthesize extremely high-resolution terrain and heightmap data provided by the NASA ‘Visible Earth’ project utilizing multiple state-of-the-art generative networks in a two staged pipeline consisting of a deep convolutional GAN (DCGAN) (Radford et al., 2015) along with ‘pix2pix’ GAN (Isola et al., 2016) which inferred the respective texture map of random 512px crops of terrain data. (Wulff-Jensen et al., 2018) attempted a similar feat using DCGANs with multiple state of the art deep learning optimizations, where they attempted to generate outputs similar to the Alps imagery. While (Spick et al., 2019) attempted to refine this approach on heightmaps with a region-specific training objective with Spatial GANs (Jetchev et al., 2016) we strictly followed the one defined by the (Beckham and Pal, 2017) as we generated the equivalent texture maps as well.

3. Methodology (Contributions)

3.1. Data Acquisition

For the purpose of this project, we decided to utilise heightmap data generated from a Procedural Map Generator as the scope of this exploratory experiment is to examine the capabilities of Deep Generative Networks when contrasted with existing Procedural Map Generators. Moreover, it seemed more appropriate as there already exists literature with variations of (real-life) high resolution NASA heightmap and terrain data. We opted for Azgaar's Fantasy Map Generator which is based and inspired off of Martin O'Leary's Generating Fantasy Maps, Amit Patel's Polygonal Map Generation for Games, and Scott Turner's Here Dragons Abound and provides a wide array of customisable options. We generated and fetched about 6000 simple height maps and their equivalent texture maps (as adding rivers, provinces, and other terrain parameters complicated the problem) with an automated fetch script. As expected of noise-based algorithmic procedural map generators, we observed similar recurrent iterations of maps generated after every few epochs which contributes to the specificity and limitation of the dataset used. Some sample datapoints can be seen in (Figure 2) and the generated dataset can be found here.

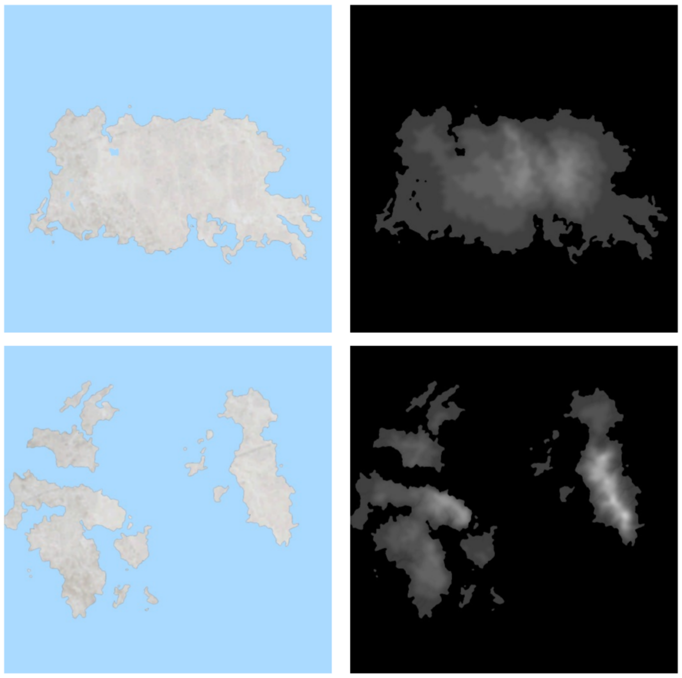

Figure 2: 512px Texture Maps (Left) and their respective Height Maps (Right)

3.2. Formulation

Suppose, $z' \sim p(z)$ is a k-dimensional sample we draw from the prior distribution. $x' = G_h(z')$ the heightmap which is generated from z', and $y' = G_t(x')$ is the texture generated from the corresponding heightmap.

We can think of this process as being comprised of two GANs: the 'DCGAN' which generates the heightmap from noise, and 'pix2pix' GAN, which (informally) refers to conditional GANs for the equivalent image-to-image texture translation. If we denote the DCGAN generator and discriminator as $G_h(\bullet)$and $D_h(\bullet)$ respectively (where the 'h' in the subscript denotes 'heightmap'), then we can formulate the training objective as:

$$\min_{G_h} \; l(D_h(G_h(z')), 1)$$
$$\min_{D_h} \; l(D_h(x), 1) + l(D_h(G_h(z')), 0)$$

where l is a GAN-specific loss, e.g. binary cross-entropy for the regular GAN formulation, and squared error for LSGAN (Mao et al., 2016). We can write similar equations for the pix2pix GAN, where we now have $G_t(\bullet)$ and $D_t(\bullet, \bullet)$ instead (where 't' denotes 'texture'), and instead of x and x', we have y (ground truth texture) and y' (generated texture), respectively. Note that the discriminator in this case, D_t, actually takes two arguments: either a real heightmap / real texture pair (x, y), or a real heightmap / generated texture pair (x, y'). Also note that for the pix2pix part of the network, we can also employ some form of pixel-wise reconstruction loss to prevent the generator from dropping modes. Therefore, we can write the training objectives for pix2pix as such:

$$\min_{G_t} \; l(D_t(x, G_t(x')), 1) + \lambda d(y, G_t(x'))$$
$$\min_{D_t} \; l(D_t(x, y), 1) + l(D_t(x, G_t(x')), 0)$$

where $d(\bullet, \bullet)$ can be some distance function such as L_1 or L_2 loss, and λ is a hyperparameter denoting the relative strength of the reconstruction loss. We set λ = 100 and use L_1.

3.3. Experimentation

As a first step, we train a DCGAN which maps samples from the prior $z' \sim p(z)$ to a generated heightmap $x' = G_h(z')$. While we experienced some issues with training stability we were able to generate heightmaps that were somewhat faithful to the original images. Apart from the aforementioned stability issues, the generated heightmaps also exhibited the presence of small-scaled artifacts which we tackled by adding a slight blur to the final images via a Gaussian kernel convolution. This served to smooth out the weird artifacts generated by the DCGAN G_h and smoothened out the heightmaps for visualization (as discussed in the next section). A pix2pix GAN G_t was jointly trained in conjunction to synthesize the equivalent texture map.

3.4. Visualisation

The maps generated in the previous part are then processed and 3D visualized in this step. First, the height map and the texture map are obtained. Both the maps are then converted to grayscale using the average grayscale formula:

$$g_r = (r + g + b)/3$$

where,

g_r is the value of red, green and blue for new grey pixel.

r is the value of red in original pixel.

g is the value of green in original pixel.

b is the value of blue in green pixel.

From these grey pixel values, height (intensity) values are obtained by scaling them to get a value between 0 and 1. As mentioned in the previous section, a slight blur is applied with a Gaussian Kernel Convolution on the final heightmaps to accommodate for any sharp / rigid edges and irregularities, making the final render smoother and continuous. We use the height values obtained from the grayscale image as the z coordinate to plot the 3D map. Finally, the respective texture mesh is placed over the 3D Visualization and the final 3D map is then saved as an HTML file for viewing (Figure 3).

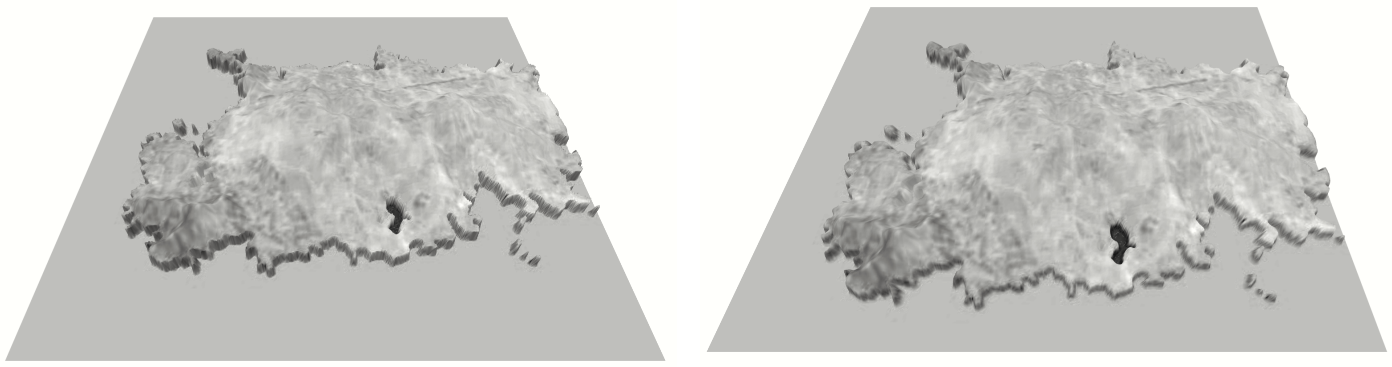

Figure 3: HTML Visualization / Final 3-D Map Render (Original – Left; Smoothened – Right)

3.5. Conclusions & Future Work

This concept and its application has the ability to change how designers of 3D environments create assets, reducing the time involvement of generating base regions for 3D environments. Or at a different level removing designers altogether, where a push of a button can see countless useable output regions being shown to the development team, seamlessly integrating it with other generated regions in continuous open world environments that generate data on the fly.

This method provides benefits over commonly used techniques such as Perlin noise and Diamond Square lack, such as the ability to generate height maps from specific input images of desired features with nearly zero human interaction. While the repeating features that can occur within Perlin noise are reduced in this method through the learning of larger scaled regions that contain more features, with the limitations of repeating features being controlled by the depth of the input tensor of noise to the generator. Hence, we can conclude that this project is a step towards non-algorithmic procedural map generation given synthetic fantasy map data.

As of this moment, there is no reliable source for hand-crafted fantasy map data which is partly why we opted for a pre-built generator to generate our dataset. If we happen to find one in the future, we would definitely like to observe results after training our model on that newly formed dataset.

In the future, we would like to integrate and experiment with SGANs similar to (Spick et al., 2019) into our pipeline.

Another neat addition to this idea would be the addition of a segmentation pipeline to classify different parts of the terrain, e.g. biomes, provinces, states, etc. This effectively serves as a layer of metadata that can be leveraged to add interesting detail in the terrain. For example, if the segmentation identifies a certain region in the generated terrain as ‘jungle’, the 3D game engine (or renderer) can automatically populate that region with trees and plants (This is called a ‘splatmap’ in Computer Graphics literature).

4. Source Code

This Project (including implementation details, dependencies, and execution instructions) is publicly available on Github.

5. References

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, & Yoshua Bengio. (2014). Generative Adversarial Networks.

Perlin, K. (1985). An Image Synthesizer. In Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques (pp. 287–296). Association for Computing Machinery.

Fournier, A., Fussell, D., & Carpenter, L. (1982). Computer Rendering of Stochastic Models *Commun. ACM, 25*, 371-384.

Mandelbrot, B. and van Ness, J.W. (1968) Fractional Brownian Motions, Fractional Noises and Applications. SIAM Review, 10, 422-437. http://dx.doi.org/10.1137/1010093

T. Antoniades, “Iggi 2018 conference key note,” Sep 2018.

AboundDragons, “Perlin Noise, Procedural Content Generation, and Interesting Space.” https://heredragonsabound.blogspot.com/2019/02/perlin-noiseprocedural-content.html, 2019.

R. Jain, A. Isaksen, C. Holmgard, and J. Togelius, “Autoencoders for ˚level generation, repair, and recognition,” in Proceedings of the ICCC Workshop on Computational Creativity and Games, 2016.

M. Guzdial, D. Long, C. Cassion, and A. Das, “Visual procedural content generation with an artificial abstract artist,” in Proceedings of ICCC Computational Creativity and Games Workshop, 2017.

J. Klein, S. Hartmann, M. Weinmann, and D. L. Michels, “Multiscale terrain texturing using generative adversarial networks,” in 2017 International Conference on Image and Vision Computing New Zealand (IVCNZ), pp. 1–6, IEEE, 2017.

V. Volz, J. Schrum, J. Liu, S. M. Lucas, A. Smith, and S. Risi, “Evolving mario levels in the latent space of a deep convolutional generative adversarial network,” in Proceedings of the Genetic and Evolutionary Computation Conference, pp. 221–228, ACM, 2018.

Christopher Beckham, & Christopher Pal. (2017). A step towards procedural terrain generation with GANs. https://arxiv.org/pdf/1707.03383.pdf

A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” arXiv preprint arXiv:1511.06434, 2015.

Isola, Phillip, Zhu, Jun-Yan, Zhou, Tinghui, and Efros, Alexei A. Image-to-image translation with conditional adversarial networks. CoRR, abs/1611.07004, 2016. URL http://arxiv.org/abs/1611.07004.

A. Wulff-Jensen, N. N. Rant, T. N. Møller, and J. A. Billeskov, “Deep convolutional generative adversarial network for procedural 3d landscape generation based on dem,” in Interactivity, Game Creation, Design, Learning, and Innovation, pp. 85–94, Springer, 2017.

Ryan J. Spick, Peter Cowling, & James Alfred Walker (2019). Procedural Generation using Spatial GANs for Region-Specific Learning of Elevation Data*2019 IEEE Conference on Games (CoG)*, 1-8.

N. Jetchev, U. Bergmann, and R. Vollgraf, “Texture synthesis with spatial generative adversarial networks,” arXiv preprint arXiv:1611.08207, 2016.

Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, & Stephen Paul Smolley. (2017). Least Squares Generative Adversarial Networks.

1 https://github.com/ramanshgrover/Procedural-Map-Generation