Procedural Generation using Spatial GANs for Region-Specific Learning of Elevation Data

Ryan J. Spick
Department of Computer Science
University of York, York, UK
rjs623@york.ac.uk

Peter Cowling
Department of Computer Science
University of York, York, UK
peter.cowling@york.ac.uk

James Alfred Walker, *Senior Member, IEEE*
Department of Computer Science
University of York, York, UK
james.walker@york.ac.uk

***Abstract*—Heightmap generation is currently a tedious topic with the majority of generation using Perlin noise which forms a reliable, but sometimes repetitive output. In this paper, a method of generating height maps from real-world digital elevation data taken from specific regions of the planet is proposed. Raw elevation data sourced from NASA's SRTM (30m) data set is transformed into a height map format, this data is then passed into a two type unsupervised model. The method uses a type of generative adversarial network to learn the spatially-invariant features within the input regions. Producing a network model that can output an extensive amount of varying, but visually and structurally similar height maps to that of the input regions. The visual validity of outputs from the network was tested using data from 262 human participants, with over 90.15% of generated samples being correctly assigned to the original input data with a significance of P <0.001.**

***Index Terms*—Deep Learning, Generative Adversarial Network, Height Maps, PCG.**

I. Introduction

Heightmaps provide a lightweight method of displacing 3D mesh terrains into a desired structure within graphical environments such as; games, simulations, scientific renders etc. Many modern open world games (*Skyrim*, *Fallout*) compress their map data into a heightmap image, applying some interpolation of values to smooth the differences between two height points and increase detail when the scene is rendered. Usually, height maps take the format of a raster image where the image forms a 2D array of points; for simplicity these images usually only use 1 colour channel forming a grayscale image. Each point represents a corresponding point in virtual space that can displace the terrain equal to the intensity of the pixel's colour.

Currently, height maps are largely generated via procedural noise based systems such as Perlin noise [1], real-world height data [2] or a more algorithmic approach like the diamond square algorithm [3]. Perlin noise 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, though using Perlin noise entirely on its own can create non-realistic height maps, due to the repeating non-structured noise features. Fractal Brownian Motion (FBM) [4] is a method of controlling 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. The diamond-square algorithm [3] 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 terrain outputs. None of these methods provides direct control over the ability to create variant procedural terrain based on a target objective without the need for human supervision.

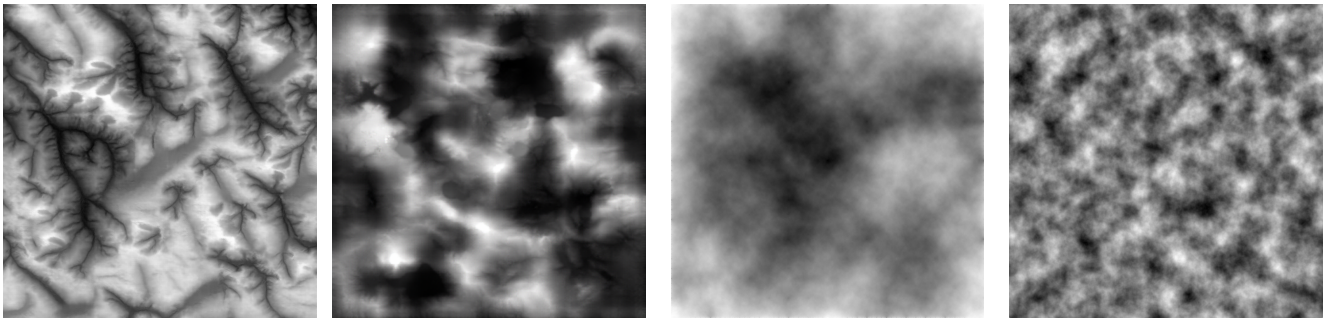
Certain developers of games have expressed the need for additional tools for generating terrain. The team behind *Hellblade*, at a Keynote talk at the IGGI conference on games 2018 [5], expressed the need for more interesting and interactive world map environments that can be generated with little input through follow up questions to their talk. Another example is an author of a highly detailed 2D map generation tool called *Here Dragons Abound* [6] 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.

The ability to generate terrains of a specific type could provide artists with a base to add the structures and areas necessary for a game environment. While at a broader level the generation tool could entirely replace creations of height maps through automated learning of regions that the creators want the game to exist within. The method proposed is a novel approach to generating heightmaps learnt through a generative deep learning model which can output similar regions to that which they were trained on, essentially creating an environment where users can specify what their procedurally generated terrains should resemble.

The structure of the paper is as follows; Background work is discussed in Section II. Followed by an overview of data collection and cleaning in Section III. Section IV outlines the process of obtaining and training network paramaters. With results shown in Section V and across Figs. 4, 6, 7 and 8.

This work was supported by the EPSRC Centre for Doctoral Training in Intelligent Games & Games Intelligence (IGGI) [EP/L015846/1] and the Digital Creativity Labs (digitalcreativity.ac.uk), jointly funded by EPSRC/AHRC/Innovate UK under grant no. EP/M023265/1.

978-1-7281-1884-0/19/$31.00 ©2019 IEEE



(a) *Proposed* Spatial GAN (b) Deep Convolutional GAN (c) Diamond-Square Algorithm (d) Fractal Perlin Noise

Fig. 1: The proposed method using Spatial GANs (a) vs. other common approaches (b,c,d). The comparison shows a large improvement on the learning structure of the SGAN over the previously attempted DCGAN results - which was trained on similar elevation data. While providing a contender to the predominantly used fractal Perlin noise, with structures learned directly from real terrain. Table I shows a more concise comparison.

Lastly a discussion of issues and a summary conclusion appear in Sections VI and VII.

II. Background

A. Procedural Content Generation (PCG)

PCG is a method of utilising techniques and algorithms to generate content through automated processes with a focus on randomness, with the definition explicitly excluding any content that has a manual creation process (using graphics engines, inbuilt editor tools) [7]. Though this is an arguable statement, as content generated with procedural techniques that are polished through manual involvement may still be classified as procedural, creating a slightly unclear divide on what PCG encompasses. Though there are a vast amount of contexts that PCG surrounds such as; Height maps, AI agents, rules of games, automated balancing etc.

PCG was originally created as a way of compressing data, with Akalabeth: World of Doom (1979) being one of the first instances, though since setting precedence for a vast amount of games to adopt this technique, using terrain generation methods to condense endless maps into a few lines of code [8]. These early games (e.g. Rogue (1980)) would use PCG as a method of holding a larger static game world that would usually require more data storage than was available at the time, using a seeded random number generator the exact worlds the developers devised could be replicated. Another classic example is the adventure game Elite (1984), a game where thousands of planets were generated procedurally, while using seeds to recreate prior interesting generations, reducing what would have been an intractable problem given the memory bottlenecks at that time into a remarkable, near-endless game.

B. Deep Learning Generative Methods

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.

Generative adversarial network (GANs) (2014) [9] 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.

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. Variational autoencoders, a type of unsupervised learning method, has been used to generate levels [10], visually changing graphics outside of game engines using convolutional filters to change the colours to something more understandable to the user [11]. Also improved texturing of 3D environments with the use of GANs [12]. Lastly a study on level generation was conducted using GANs [13]. 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.

C. Related Previous Work

A similar task and the original idea of generating terrains using GANs from regions of the planet was first mentioned in [14], where the authors attempt to generate similar outputs from a data set from the Alps. Utilizing multiple state of the art deep learning optimization's combined with a deep convolutional GAN (DCGAN) [15], a type of GAN built around the augmentation of convolutional layers in place

TABLE I: Comparison of algorithms covering features that would be deemed important within an automated generation tool.

Algorithm	Tileable (Seamless)	Detailed Generations	Non-Repeating Features	Feature Selection
SGAN	-	✓	✓	✓
Fractal Perlin Noise	✓	✓	✓	-
Perlin Noise	✓	-	-	-
Diamond-Square	✓	✓	✓	-
DCGAN	-	-	✓	-

of the multilayered perceptron [16] structure of the original GAN. With the filters of the convolutional layer producing a model focused more at spatial correlations, they show human interaction results with their output rendered height maps which show promising potential. Though the inherent lack of upscaling from a DCGAN shows a limitation from a usability perspective. Furthermore, the terrain outputs produced appear fairly noisy and lack an amount of similarity to their derived regions, perhaps due to the oversaturation of varied data points used.

Recently in 2017, a novel framework was proposed for the use of Spatial GANs (SGAN) [17]. Spatial GANs modifies the DCGAN [15] to remove fully connected layers, this allows for the scalability of outputs to any size and has properties that allow learned features to be mapped translation-invariant on to outputs. The SGAN architecture also follows DCGANs by removing all pooling layers, replacing them with strided convolutions. The strides within the generator and discriminator networks inherently learn the up and down scaling respectively, with 1/2 stride in the generator and 2 stride in the discriminator (Stride size affects how far the kernel will move when calculating outputs - stride of 1/2 will double the output size, 2 will halve the output size etc.). The input noise distribution of an SGAN begins with a tensor of latent noise compared to a single vector, providing a much larger range of possible spatially-variant structures for generations of the network. Overall SGAN boasts an accurate texture specific solution with an extremely fast output time compared to the previous methods discussed, due to the simplicity of network design - removing the aforementioned fully connected layers.

III. Data Collection and Preprocessing

The data was collected from an open source elevation map data set Shuttle Radar Topography Mission 30m (SRTM) courtesy of NASA [18]. This data provides a 30m to 1-pixel spatial resolution of the elevation topology of the planet, providing enough detail for the generative model used to learn the underlying features. Each region was approximately taken from a corresponding 100km^2 region.

Using the python library Elevation.py, a module for; accessing, downloading and processing SRTM elevation data, an automated tool was created to quickly process large areas of the planet given a rectangle area defined by longitude and latitude points, allowing the system to explore terrain patches for random locations of the planet, to selectively choose inputs

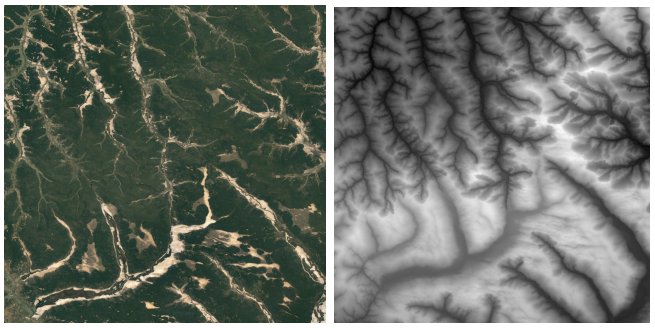


Fig. 2: Example training image region (right) (100km^2) within Southern Africa (-14.12, 17.34) (left). Capturing the elevation data of the region in to a height map format. With darker areas of the height map corresponding to flatter terrain and whiter with higher terrain.

to the network that are structurally interesting and different enough from other examples to train. This created easily manipulatable data which was banded from its raw continuous height data into a grayscale (1 colour channel) heightmap format, which was required as the input shape for the network.

The image was resized by a factor of 0.5, using anti-alias resizing, compressing some large areas of terrain which initially created a near intractable task for the network to learn. This value was chosen through a subjective initial run through of varying sized inputs to the network. Resizing by a half was a value such that the input features were still recognizable, but the relatively small filters of the network would still learn. The height map data was largely clean, though when initial runs were conducted there was a small amount of pixel noise within the downloaded height maps. To reduce the chance of this noise appearing in the outputs a small median blur of kernel size 3x3 was applied to remove the noise without affecting the structural integrity of the height map. An example of the final clean height map can be seen in Fig. 2 alongside the real region the elevation data was extracted from (Taken from Google maps).

IV. Methodology

In this section, there will be an explanation of how and why network parameters were chosen. The experimental setup that the network was trained on, and the times involved. Finally the network training process itself, with a further discussion on training issues and how they were alleviated.

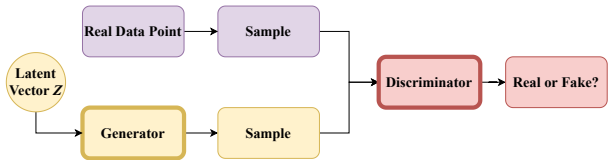


Fig. 3: Generative network training sequence, sample is taken from Generator or Real Data Point and fed through to the discriminator, which classifies the inputs as discussed in background II-B.

A. Network Parameters

As discussed in the introduction, the spatial GAN model [17] provides a method of learning translation-invariant features from an input image through a discriminator network, mapping these features back on to a tensor of noise using an upscaling generator network. A visualisation of the network structure is shown in Fig. 3.

The original paper [17] referencing SGANs reviewed the parameter tuning and recommended values for the discriminator and generator. For the learning properties exhibited within the paper filter sizes of 5 for both networks were used. While using a depth of 5 layers within both networks.

Using these parameters as a starting point, multiple runs of the model were conducted to explore the best parameters for filters and depths of both networks. This was an initial preliminary test run over a small number of epochs, the outputs were compared visually to their input region as a benchmark. It was found that changing the generator filter sizes from 5 to 7 allowed a more detailed representation of conjoining features within the height maps, essentially allowing the filter to contain more feature information of a large area, while keeping the discriminator filters lower, at a size of 3, inhibiting the speed at which the discriminator could learn, crucial to the equilibrium of the two networks. Previously the discriminator would rapidly outperform the generator network which would cause the discriminator to almost always correctly classify the inputs correctly.

While testing the depth of the network, it was observed that the deeper the network the more of the local structures could be combined. The network was tested with depths of 4, 5 and 6 with 5 being the recommended value in the original paper. The depth used in this network was chosen as 6 for this specific task, as the outputs needed to contain a large portion of connected features that resulted within the input region. Even though the deeper network could produce more detailed and connected region, the training time overhead was much larger. Using 1 high end graphics card (GTX 1080TI) as a benchmark the times per epoch for each depth were as follows; depth of 4 - 13.5s, depth of 5 - 32.9s, depth of 6 - 128s. Any depth above this required more GPU memory than was accessible.

A combination of the deeper network and more fine-tuned discriminator and generator filters allowed for a smooth transition of trained regions, with values that shifted away from the chosen parameters showing struggles to learn, and less visually detailed outputs. The exact hyper parameters used within the paper can be seen more clearly in table II.

TABLE II: List of the main SGAN hyperparameters for the results shown in this paper. N^i represents the current layer.

Parameter	Value
Optimizer	ADAM
Learning rate	0.0005
Momentum	0.5
L2 regularization	1e-5
Batch size	32
Depth (N)	5 & 6
Discriminator Filter's	(3,3) * $2^{(N^i + 6)}$
Generator Filter's	(7,7) * $2^{(N^{max} - N^i + 6)}$

B. Experimental setup

The training models were conducted on a cluster of 8 GTX 1080 TI's. Each input region could be fully trained within around 4 hours of training to a point of an output that resembled the original training point's features; While training to a point of completion took around 4 times that time (16 hours total), though the final 12 hours of training showed relatively little visual improvements and would only be required for extremely detailed results. Training was deemed complete when no noticeable visual changes were seen over several epochs. This could also be quantified by exploring the loss graph, when the plot of the loss for the generator started to plateau with no more changes this signalled the end of training. The whole training process could easily be scaled back to one or more lower end card, though training times would take significantly longer.

C. Network Training

Each discriminator iteration, a mini-batch of 32 images was constructed, taking random identically sized crops (256x256) of the input, this allowed the network to learn the structure of the overall images through smaller representations, generalizing far better on generated outputs. The network architecture is shown in Fig. 3. The inputs of real data to the discriminator were decayed over every early epoch [19], allowing the generator to remain relevant within the early stages of the training process due to the inherent noisy outputs the generator initially produces. These inputs had a noise mask of Gaussian blur applied with a sigma of 0.5 decaying down to a minimal value before being removed. This occurred from the first epoch through to epoch approximately 50, where the noise was so miniscule there was no more effect.

To further assist the generator in the learning of the highly complex structures a method used in [20] was implemented, a technique of optimally training a network that perhaps struggles with mode collapse within the generator (wherein the network collapses, producing invariant samples from a

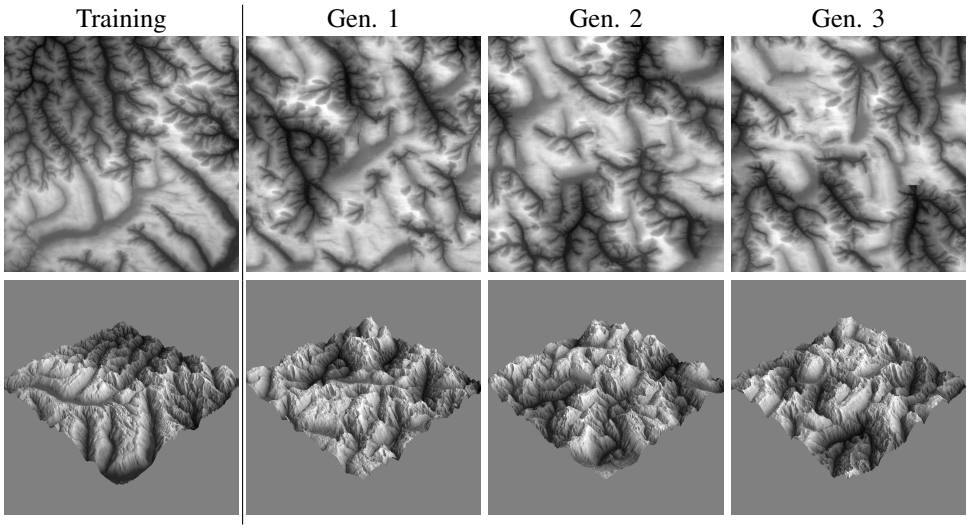


Fig. 4: Height map (top) and 3D render (bottom) representation for training and 3 generations regions for (17.3,-14.1,18.1,-13.3)

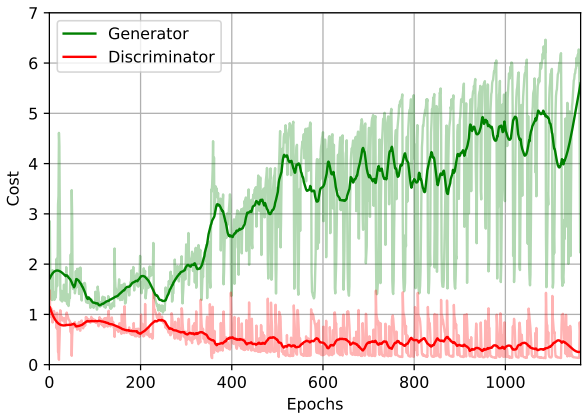


Fig. 5: The cost of the generator and discriminator network over time - until a point of completion in training. Transparent lines show the raw cost change, full lines show a more smoothed visualization of the cost using a Savitzky - Golay filter with a window size of 71.

few more heavily weighted learned samples). During each epoch the generator was trained 2 times for every discriminator update, creating a more balanced approach for the previously under performing generator network. Lastly when feeding values in to the discriminator the network utilized GAN batch normalization [21]. By only allowing the discriminator to see one type of alternating input (real or fake) per iteration, the discriminator avoids internal conflicts between data distributions when updating gradients. The network was trained in all instances for a varying amount of time based on the input feature's complexity, though as mentioned above generally around 4 hours for a structurally similar output was given - with extra time spent optimizing the for improved visual concatenation of features. Generally speaking the more complex the structure of the input region to the network the longer the network would need to be trained for. There was little indication to determine the network was converging on a visually similar output, other than the subjective visual appearance of the outputs.

To determine when a region was fully trained, the loss function of the generator could be examined. One measurement used was the change in loss over epochs, where the generator loss would start to level out, eventually plateauing an example shown in Fig. 5. At this point, the network would show little signs of visual change within the outputs. In any case, the model's parameters, alongside a visual output sample, were saved every 50 epochs ensuring that data was captured frequently enough that if an interesting formation of terrain within a height map occurred the model could be reloaded to produce more outputs from that data point.

The early iterations of training were fairly stochastic, usually within supervised deep learning the loss function gives a good representation of the learning of the network, in this situation the loss function of the generator would rapidly spike and trough early on, until some point of convergence later on into training.

For each full training session for the generative network one entire region of the desired output structure would be used, alongside mini-batch crops. The region used was of size 1400x1400 corresponding to a 100km^2 region's elevation data. As previously discussed the input image was downsampled prior to this stage to ensure the filters could learn the structures of the terrains.

Post-training a median blur of filter size 5x5 was applied to the outputs. This was due to small noise being retained in the output image from the generation process. Although visually the outputs were feature-wise almost identical, the small amount of noise would create a transition of features within the rendered version to appear unsmooth with sharp points.

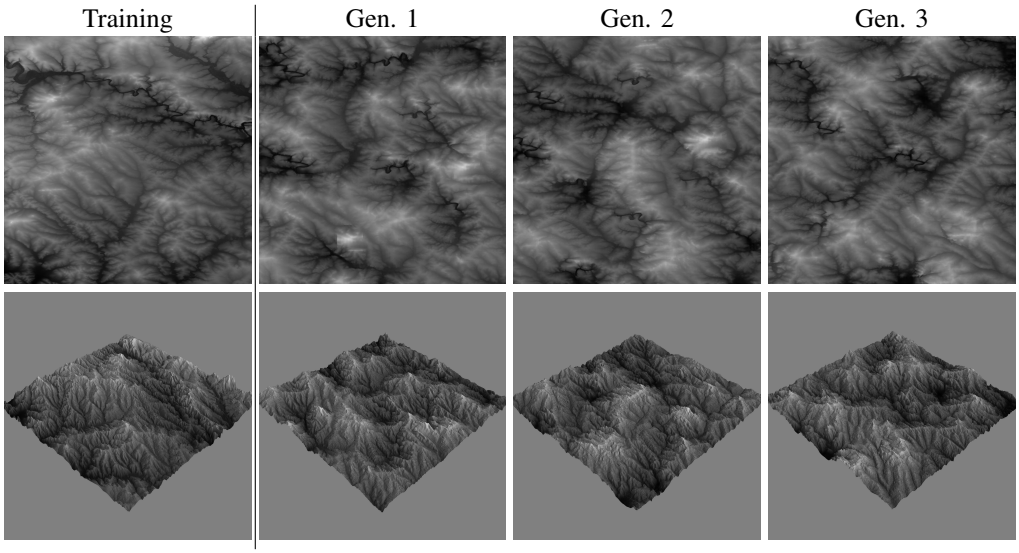


Fig. 6: Height map (top) and 3D render (bottom) representation for training and 3 generations regions for (125.4,53.1,126.2,53.9)

V. Results

A. Generated Output

Following the final training of a model, the network could then be reloaded and produce replica samples of the original region almost instantaneously. With the output size of the image only limited by GPU memory. Fig. 4 shows a grid comparison of one of the generated samples, each generated region taking roughly 50ms to generate on a GTX 1080Ti. These generations were output at a resolution of 2.4k x 2.4k, roughly 2x the size of the input texture which is a similar resolution prior to downscaling the input.

Several regions of the planet were chosen based on their ability to convey the different features and structures that the method is capable of understanding, though the selection of coordinates to acquire regions was relatively random. Figs. 4, 6, 7 and 8 showing four chosen trained regions alongside their general location and more specifically their longitude-latitude position. These figures show the direct output of a fully trained network based on the corresponding input, and for comparison an example of how they would appear in a very basic 3D render, mimicking how a generated sample might be visualised within a specific use case scenario, minus the texturing and entities.

When analysing the figure results in individual detail it is apparent that there are some limitations with feature connections. In Figs. 4 and 6 the input contains a large number of valleys and ridges, the generations fail to completely connect these features together. Whereas in Figs. 7 and 8 that have less connected regions in the input, the generations appear to have less issues with the transfer of features.

When comparing the outputs to a more robust method such as Perlin noise, the proposed method provides a far more structured and controlled output, with direct control through given inputs of the network. Whereas Perlin noise can generate complex fractal terrain shapes, the features may appear to continuously repeat over large distances, alongside the element of randomness within a Perlin noise algorithm it cannot ensure that a specific feature will exist within an area.

B. Participant Findings

Creating an argument for visual results is extremely subjective as to what appears to look more visually appealing/similar, excluding algorithmic comparison techniques such as Keypoint Matching [22] or inspecting distribution of pixels, which appear bias for the data produced in this paper as the output images contain inherently similar distributions to their training regions. Therefore to cement the justification of useability within virtual environments, an experiment was devised to discover participant's ability to distinguish between the multiple fake regions that were generated. The results of the study show a strong correlation of the generated samples with the original training points with 90.15% classification accuracy when asking participants to classify a sample against multiple regions, one of which the sample was derived from.

262 individuals were tested through various online forums, notably Reddit forum r/proceduralgeneration. From the 4 trained regions shown in Fig. 4, 6, 7 and 8 the model generated 20 random samples for each region, leaving a pool of 80 samples. During the survey the participant would be shown a total of 20 samples, with each sample individually shown alongside the 4 training regions, these were drawn from a uniform distribution of the 80 generated samples. The survey explicitly asked the participants to record their answer based on which of the 4 original training regions they thought more closely resembled the shown sample, since each sample was derived from one of the training regions the accuracy could be recorded for each choice. The survey took on average just over 2 minutes (standard deviation(SD) of 90 seconds) for each participant to complete, giving around 6 seconds (SD 4.5 seconds) per answer.

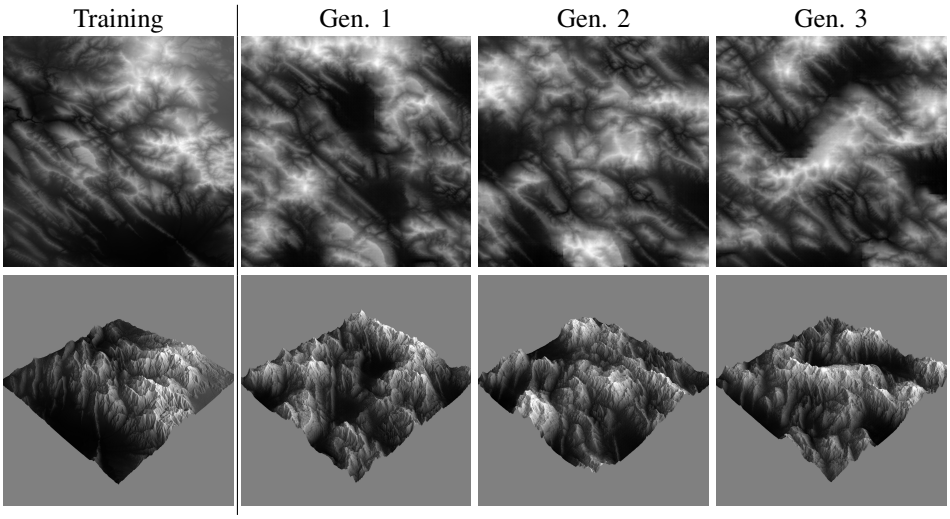


Fig. 7: Height map (top) and 3D render (bottom) representation for training and 3 generations regions for (44.3,36.1,45.1,36.9)

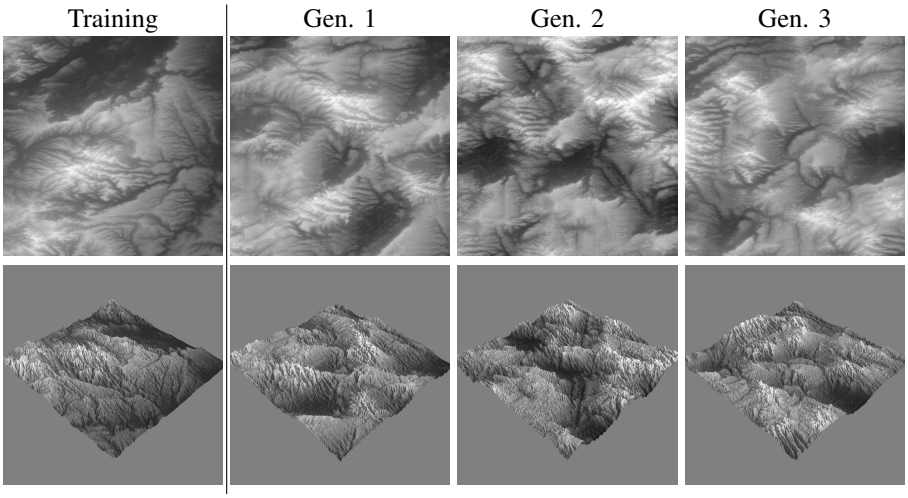


Fig. 8: Height map (top) and 3D render (bottom) representation for training and 3 generations regions for (-1.1,52.3,-0.3,53.1)

A non-parametric Wilcoxon test revealed a significance of **p <0.001**; showing a highly significant result for users managing to classify generated height maps with the regions they were derived from.

VI. Discussion

There were several issues that arose when generating the results shown in this paper from preprocessing to training the GAN on complex inputs. this section will focus on the discussion of encountered obstacles and how they were overcome.

The network captures most of the local structures - but fails to connect these small local structures into the larger and more impactful areas; large valleys, long mountain ridges etc. This can be seen within [17] where large connected structures of satellite images fail to be learned. The structure of terrain can be more accurately connected by increasing the depth of the generator and discriminator though by doing so requires a larger amount of memory and training time. Furthermore, the method struggled with regions that didn't exhibit visual repeating features, as the filters would be constantly updated in a different direction to the general feature trend. This can be observed through the results of the paper always containing a general feature trend across the training images.

During early experiments, results and parameter tuning were very much visually centred, running a model and analyzing the results after several hundred epochs. Though in some cases the networks learning potential could fail, in this instance the network had to be stopped manually and restarted, an artefact caused by rapid learning of the discriminator within GANs [23]. Although once decayed inputs to the discriminator were implemented this issue disappeared. It was also noted that the first 100 epochs did not influence or correlate to how the final outputs appeared, with a large variance over a small range of epochs on the data structures of outputs.

While considering both factors, it made for a very difficult and sometimes stochastic experience when training the GAN.

Though through the various additional optimization's: batch norm, decayed noise etc. this made the process far less unpredictable allowing for a more controlled training process.

VII. Conclusions & Future Work

Overall a robust method for the learning and generation of elevation data in the form of a height map has been proposed. With a pipeline describing how one can gather large open source elevation data and apply a type of generative adversarial network that has the ability to learn position-invariant features. Outputting these on to a tensor of noise that up-scales to an arbitrary size limited only by GPU memory, with features exhibited in the output of the network appearing spatially different but structurally similar to the learned input.

The novel contributions of this work show a new technique to generate height maps for 3D environments. Where previous work has been shown using GANs for height maps, the outputs were unstructured and lacked the ability to upscale arbitrarily. SGANs improves on some issues of the previous approach that used DCGANs.

This method provides benefits that commonly used techniques such as Perlin noise 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.

The work and application has the ability to change how designers of 3D environments create terrain 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 use-able output regions being shown to the development team

Though this paper has introduced a new robust tool to generate height maps based on a target region, the limitations discussed previously to create a barrier to entry within certain situations. The lack of ability to seamlessly integrate with other generated regions is a large problem in continuous open world environments that generate data on the fly.

Further research will investigate the creation of a similar type of model that has the ability to endlessly patch together results shown in this paper. Alongside this, a method of learning multiple region's features simultaneously to create a complete mixture of the input regions would provide more variety for generated outputs. The two proposed improvements coupled together would be the start of a tool that would allow designers to specify where they wish certain features to appear. Through the manipulation of the input noise to the generation network, the learned features within the filters of the network can be more highly activated in the desired location.

This tool would essentially be an unsupervised learning model that would allow the user to draw or paste sections of a collection of learned regions on to a canvas, in real time the model would output the height map with seamless transitions between the chosen features for any resolution size.

References

[1] K. Perlin, "An image synthesizer," *ACM Siggraph Computer Graphics*, vol. 19, no. 3, pp. 287–296, 1985.

[2] J. Becker and D. Sandwell, "Srtm30_plus: Srtm30, coastal & ridge multibeam, estimated topography," *Electronic journal. URL: http://topex. ucsd. edu/WWW_html/srtm30_plus. html*, 2006.

[3] A. Fournier, D. Fussell, and L. Carpenter, "Computer rendering of stochastic models," *Communications of the ACM*, vol. 25, no. 6, pp. 371–384, 1982.

[4] B. B. Mandelbrot and J. W. Van Ness, "Fractional brownian motions, fractional noises and applications," *SIAM review*, vol. 10, no. 4, pp. 422–437, 1968.

[5] T. Antoniades, "Iggi 2018 conference key note," Sep 2018.

[6] AboundDragons, "Perlin Noise, Procedural Content Generation, and Interesting Space." https://heredragonsabound.blogspot.com/2019/02/perlin-noise-procedural-content.html, 2019. [Online; accessed 6-February-2019].

[7] J. Togelius, E. Kastbjerg, D. Schedl, and G. N. Yannakakis, "What is procedural content generation?: Mario on the borderline," in *Proceedings of the 2nd international workshop on procedural content generation in games*, p. 3, ACM, 2011.

[8] A. Amato, "Procedural content generation in the game industry," in *Game Dynamics*, pp. 15–25, Springer, 2017. PCG Online and offline definitions. A few examples of games that used pcg History of PCG.

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[10] R. Jain, A. Isaksen, C. Holmgård, and J. Togelius, "Autoencoders for level generation, repair, and recognition," in *Proceedings of the ICCC Workshop on Computational Creativity and Games*, 2016.

[11] 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.

[12] J. Klein, S. Hartmann, M. Weinmann, and D. L. Michels, "Multi-scale terrain texturing using generative adversarial networks," in *2017 International Conference on Image and Vision Computing New Zealand (IVCNZ)*, pp. 1–6, IEEE, 2017.

[13] 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.

[14] 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.

[15] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015.

[16] R. J. Schalkoff, *Artificial neural networks*, vol. 1. McGraw-Hill New York, 1997.

[17] N. Jetchev, U. Bergmann, and R. Vollgraf, "Texture synthesis with spatial generative adversarial networks," *arXiv preprint arXiv:1611.08207*, 2016.

[18] R. Bamler *et al.*, "The srtm mission: A world-wide 30 m resolution dem from sar interferometry in 11 days," in *Photogrammetric week*, vol. 99, pp. 145–154, Berlin, Germany: Wichmann Verlag, 1999.

[19] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," *arXiv preprint arXiv:1701.04862*, 2017.

[20] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," *arXiv preprint arXiv:1701.07875*, 2017.

[21] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[22] L. Daoud, M. K. Latif, and N. Rafla, "Sift keypoint descriptor matching algorithm: A fully pipelined accelerator on fpga(abstract only)," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '18, (New York, NY, USA), pp. 294–294, ACM, 2018.

[23] S. A. Barnett, "Convergence problems with generative adversarial networks (gans)," *arXiv preprint arXiv:1806.11382*, 2018.