Realistic and Textured Terrain Generation using GANs

Ryan Spick
Department of Computer Science
University of York, UK
ryan.spick@york.ac.uk

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

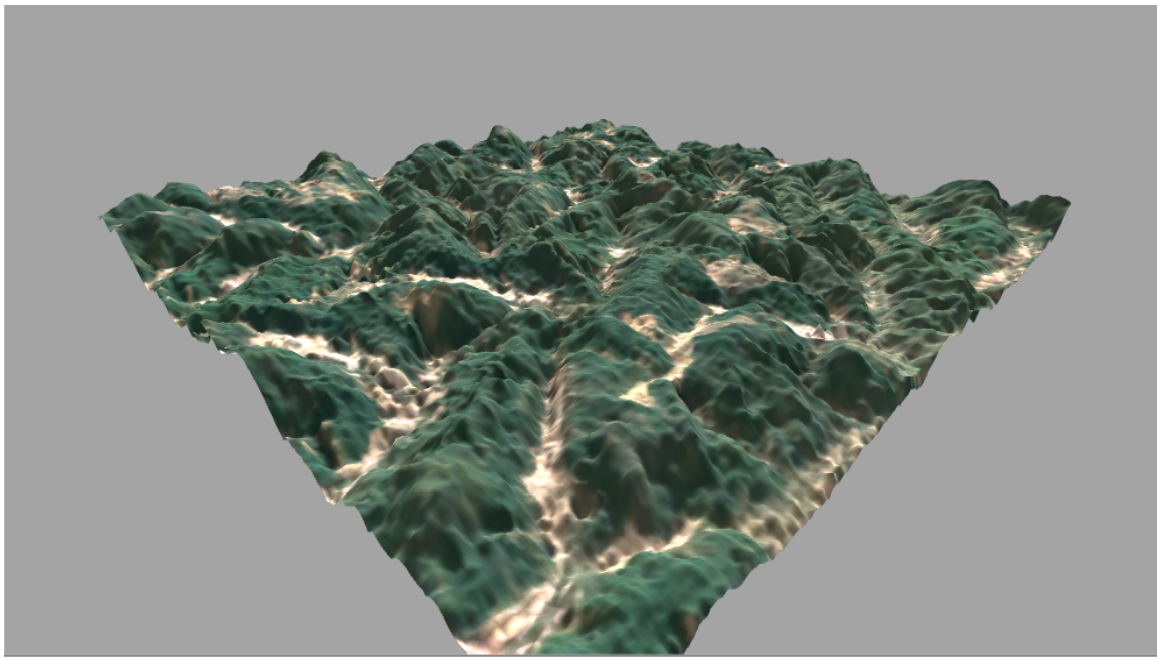


Figure 1: Example of a generated fully-textured region using the proposed technique and data processing in this paper.

ABSTRACT

In computer graphics and virtual environment development, a large portion of time is spent creating assets - one of these being the terrain environment, which usually forms the basis of many large graphical worlds. The texturing of height maps is usually performed as a post-processing step - with software requiring access to the height and gradient of the terrain in order to generate a set of conditions for colouring slopes, flats, mountains etc. With further additions such as biomes specifying which predominant texturing the region should exhibit such as grass, snow, dirt etc. much like the real-world. These methods combined with a height map generation algorithm can create impressive terrain renders which look visually stunning - however can appear somewhat repetitive. Previous work has explored the use of variants of Generative Adversarial Networks for the learning of elevation data through real-world data sets of world height data. In this paper, a method is proposed for learning not only the height map values but also the corresponding satellite image of a specific region. This data is trained through a non-spatially dependant generative adversarial network, which can produce an endless amount of variants of a specific region. The textured outputs are measured using existing similarity metrics and compared to the original region, which yields strong results. Additionally, a visual and statistical comparison of other deep learning image synthesis techniques is performed. The network outputs are also rendered in a 3D graphics engine and visualised in the paper. This method produces powerful outputs when compared directly with the training region, creating a tool that can produce many different variants of the target terrain. This is ideally suited for the use of a developer wanting a large number of specific structures of terrain.

CVMP '19, December 17–18, 2019, London, United Kingdom

ACM ISBN 978-1-4503-7003-5/19/12.
https://doi.org/10.1145/3359998.3369407

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; **Computer graphics**.

KEYWORDS

Deep Learning, Generative Adversarial Networks, Procedural Content Generation

ACM Reference Format:
Ryan Spick and James Alfred Walker. 2019. Realistic and Textured Terrain Generation using GANs. In *Proceedings of European Conference on Visual Media Production (CVMP '19).* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3359998.3369407

1 INTRODUCTION

The creation of 3D terrain for virtual environments can take a large amount of time for designers [Shaker et al. 2016]. Authored terrain in these environments can sometimes struggle to convey exactly what is required by the developer's vision. In certain environments, the terrain realism is paramount to achieve a level of immersion within the game. Current manual tools that are used are subject to skills and experience of the designer who uses them, such as Unity or World Machine.

3D terrain within virtual environments is usually created through displacing a flat grid of vertices with an image of data points that correspond to the depth at which points on the grid should be displaced. Usually, these are called height maps or displacement maps. There are other techniques, however, such as directly mapping Perlin noise [Perlin 1985] onto a grid at specific points.

Inherent problems are created with tools that use common noise-based algorithms, such as Perlin noise. Perlin noise is capable of producing realistic elevation values using multiple octaves of data. However, it can be seen as repetitive by some developers [AboundDragons 2019].

An important consideration when creating terrain is the use of texturing to add to the aesthetic of the terrain. A computationally cheap method of texturing a terrain is using an algorithm that uses a combination of gradient slope and height values. Textures can be added to produce realistic yet structurally repetitive textures across the entire terrain in a straightforward manner. Terrain authors can also be used to manually "paint" terrains within 3D engines using various brush tools.

The terrain in the real world has followed millions of years of erosion and other geological effects to reach the appearance and structure that it exhibits currently. So one way of assisting the process of design would be to use data from the real world, in which the terrain has already gone through the realistic geomorphic process. With data being extremely abundant from sources such as NASA's Shuttle Radar Topography Mission (SRTM) elevation map [Becker and Sandwell 2006], as nearly the entire planet's elevation data collected at a high resolution of one arcsecond (30m along the equator). Furthermore, with advances in satellite imagery, high detailed terrain maps can also be acquired from sources such as Copernicus Sentinel satellite or Google Earth. With both elevation and texture data, a powerful data set of real-world image data can be obtained.

This paper will focus on the use of deep learning techniques to create a novel approach for both height and texture generation within the same model, allowing an entirely automated method for producing coherent data for both texturing and displacing terrain.

Generative Adversarial Networks (GANs) [Goodfellow et al. 2014] provide a versatile method of learning underlying distributions in data sets, mapping these learned features to an output that can mimic the input data. Deep convolutional GANs (DCGANs) [Radford et al. 2015] use convolutions to find areas of correlation within an image between data, a technique more applicable to spatial data such as images or video.

A non-spatially dependant GAN called a Spatial GAN [Jetchev et al. 2016] can be used in this context to learn specific features and textures of a terrain region. Providing a perfect environment to output regions of terrain for designers by sampling the direct data of the real world, where feature space isn't important.

The contributions for this paper extend previous work in this area, where the author generated high detail height maps from a direct feed of real-world regions [Spick et al. 2019]. This work aims to further optimize the use of spatial GANs to improve the outputs on terrain, provide a benchmark for outputting height maps with synthesised texturing learned inherently within a neural network. There will also be comparisons between real and generated regions in a rendered version shown in Unity; these data points will be both visually and structurally analyzed. Lastly the analysis of data will be compared with previous state of the art texture synthesis methods, such as DCGAN and the work of Gatys *et al.*, showing an improvement for the use of this type of architecture for terrain generation.

The paper structure is as follows: Background work and related literature are discussed in Section 2. The methodology and approach, which discusses data collection and network training, with an in-depth analysis of training enhancements are in Section 3. Results are then shown for a number of trained regions, with a method comparison of previously used generative tools in Section 4. Finally, a discussion of results and conclusion appear in Sections 5 and 6.

2 BACKGROUND WORK

The generation of terrain in computer vision has been a long researched topic with procedural content generation (PCG) originally being used as a way of simply compressing data in video games. One of the earliest examples Akalabeth: World of Doom (1979), compressing the game's world using a user-specified seed. This has since set precedence for a vast amount of games to adopt this technique, allowing the generation of near-endless game environments into just a few lines of code which could be executed either on the fly or pre-computed [Amato 2017].

Another advantage of these compression techniques, that existed when data storage mediums were heavily constrained, for example in Rogue (1980), would allow the developers to use more content in the game than would have been viably stored on a conventional data storage device. Another example is adventure game Elite (1984) generating thousands of planets procedurally using computationally efficient techniques, with prior interesting generations discovered by the developers being reproduced by simply storing the seed of the random number generation. This is eerily similar to popular games today, such as No Man's Sky (2016) which uses modern rendering techniques to create beautiful traversable planets, but still employs the methods discussed in the previous examples to create a near-endless amount of planets. Giving users the ability to explore a near-infinite amount of game space, which if were stored on a storage medium would be impossibly large.

The generation of terrain for 3D environments has been a long researched area; the previous section touches on why these methods

are used. Although the approach has remained fixed throughout the years, the way procedural terrain has been created has changed.

2.1 Deep Learning & GANs

Deep learning is a method of processing data through a number of layers of varying computational functions, where each layer abstracts the input's representation. This creates a representation of the input data through the use of labels or clustering, the representations can be used in classification or generation of new data from the sampled distribution.

Deep learning has been used to generate content in a vast range of areas used in computer vision. In Procedural Content Generation via Machine Learning [Summerville et al. 2018] there are many examples of the application domain such as images, music and speech [Goodfellow et al. 2014].

GANs are a type of deep learning technique that uses two deep neural networks; a generator (G) network which attempts to create new samples from a learned understanding of the data distribution and a discriminator (D) which attempts to determine whether inputs are real or fake data, with the fake data being a direct input from the generator's output. These initially were designed with the use of multi-layered perceptrons for both networks.

In 2015, DCGANs [Radford et al. 2015] were created using Convolutional Neural Networks (CNNs) in place of the multi-layered perceptrons as a way of more efficiently and accurately mapping image inputs. The use of slight variations of GANs in computer vision and games can be seen over a large number of application areas, level generation in Mario [Volz et al. 2018], style transfer of one image domain into a different domain [Zhu et al. 2017] and generating 3D voxel-based models [Achlioptas et al. 2017].

Overall, GANs pose to be a useful upcoming tool for designers of graphical systems, especially in a domain where data is readily available. Based on the techniques shown in PCG via ML [Summerville et al. 2018] there is an obvious push for the use of unsupervised techniques that can generate similar data to a designer's examples. With the increase in available data in large quantities, GANs are proving to be an invaluable technique for unsupervised learning, which could otherwise be a near intractable problem for a supervised approach.

2.2 Height Maps

Height maps are a 2-dimensional raster image of values points which are used to displace a mesh to create 3-dimensional terrain in computer graphics. There are several well used procedural algorithms for populating height maps with elevation values.

Diamond-square [Fournier et al. 1982] is an algorithm variant of mid-point displacement, which iteratively takes subdivision of a grid averaging the middle point of a set of corner values which are randomly seeded. A slight offset is usually added to the average, which is decayed over time to determine the coarseness of the terrain. The resulting height map appears smooth and flowing like real terrain, with the ability to control the average height and coarseness easily.

Gradient noise is one of the most common low-level tools for the use in terrain generation algorithms. Ken Perlin [Perlin 1985] originally implemented this gradient noise into an algorithm known as Perlin noise, producing continuous smooth interpolations of points on a grid. Later Simplex noise development optimized Perlin noise to produce smooth values in an extended number of dimensions in a more computationally efficient way. On its own, a layer of this type of noise does not necessarily produce realistic height maps. Fractal Brownian Motion (FBM) [Mandelbrot and Van Ness 1968] is an extension of Perlin noise that combines multiple layers of Perlin noise, each with different heights (amplitudes) and frequencies. This results in height map generations that appear more realistic.

An approach that builds on the non deterministic gradient based noise is an interactive approach [Gain et al. 2009], where the author generates terrain landscapes from a series of drawn curves, spine and silhouettes. These regions are then populated with height values through noise propagation. The approach shows more active control over user centred design compared to the previously discussed techqniues.

Similar to the approach in [Gain et al. 2009], deep learning generative methods have also been used to show promising results when generating heightmaps from real-world data. Though instead of utilising user authoring, the techniques have centered around using pre-existing data containing many desirable features for a height map. Using DCGANs [Wulff-Jensen et al. 2017] features were learned from elevation data, though the lack of non-spatial dependencies meant the height map outputs were often incoherent. Further work using Spatial GANs [Spick et al. 2019] created a more stable process of generating larger scale height maps from specific regions, with outputs being highly recognizable with their derived regions.

2.3 Texturing

Texturing of graphical environments is a key part of increasing realism and gives the author a huge amount of control of the final aesthetic of the environment. Texturing is usually performed as either automated or supervised.

Using an algorithm that uses a combination of gradient slope and height values textures can be added to produce realistic yet repetitive textures across the entire terrain in a straight forward manner. Perlin noise can be used to generate additional information in an automated way, such as generating biomes for different types of texturing to be applied, Such as sand, grass, mountainous etc. Terrain artists can also add textures manually to the environment, directly controlling what textures appear in each area of the environment.

An issue that applies to these methods is the actual creation of multiple textures to be used. There is a large amount of research on the topic of specific texture generation. However, this paper will focus on deep learning techniques that have been recently shown to outperform any previous techniques.

Gatys *et al.* [Gatys et al. 2015] shows an exploration of using deep learning, specifically convolutional neural networks, to generate textures using the networks filters. Target textures are trained on a prior trained VGG-19 network, with outputs being closely related to the original image in terms of structure, though there are some disparities with colour and shapes of image.

Further work that built on top of the texture synthesis using CNNs was the adoption of GANs [Goodfellow et al. 2014] with the

use of Spatial GANs [Jetchev et al. 2016], an approach similar to DCGANs [Radford et al. 2015] but would have no concept of the spatial positioning of features.

Textures are defined as a uniform (brick wall) or irregular (wood panel) image which for most terrain texturing applications do not require a spatial correlation of features, where other image generation task such as generating faces, where eyes and mouths would need to remain spatially correlated.

3 METHODOLOGY

This section will discuss the processing of data and the techniques used to generate a coherent output, that can be mapped to a 3D render with a realistic mapping of texture data from the regions real-world satellite data.

The creation of a terrain authoring tool would allow designers to simply pass two coordinates pointing to the corners of a rectangular area covering a region of the world. This would then be passed to a GAN model which could learn and output variant regions of textured terrain for direct use within a computer vision system.

The method in this paper consists of training an SGAN with a carefully prepared input image containing a textured image which is layered directly on top of a height map using longitude and latitude coordinates.

3.1 Data

Data collected for this paper was done using NASA's SRTM [Becker and Sandwell 2006] data set, which maps the majority of the planet's elevation data. A geo-rectangle of longitude-latitude values were used to define regions that would then gather and populate a height grid with the values for the corresponding points from the SRTM data set. This creates a single channel grey scale texture which would be used as a heightmap. Areas of high values shown in white, with colours creating a gradient to black for the lowest height values. An example of the heightmap data can be seen in Fig. 2a.

The actual longitude-latitude values were generated entirely randomly, which created a large data set of randomly selected regions of the planet - this was to avoid a bias in the data selection process, since the SRTM mapping also includes oceanic regions, which were almost entirely flat height map regions. After inspecting an initial pass of data collection, the average height of oceanic height values was below an average threshold of 0m across all elevation points. Therefore if any region in the data set had a minimal range of height values (average height of the elevation under 0m Mean sea level (MSL)) they were discarded.

For the height maps corresponding texture data, Google maps API was used. The API was used to create calls based on the longitudinal and latitudinal positions of the chosen rectangle area that would visually align with the gathered height map data. An example of the textured data can be seen in Fig. 2b.

There were two problems in the texture data set that required further cleaning; clouds appeared in some satellite images, and where possible the entire region was simply discarded. In regions of highly reflective terrain, such as snow/ice, the captured texture was extremely saturated with the reflection of the sun, this was amplified more in higher elevated regions like snowy mountains. Any region containing any saturation of texture was also discarded. While there are techniques that aim to remove cloud and other anomalies from satellite images [Lin et al. 2012] it seemed unnecessary considering the large amount of usable "interesting" regions that were free of errors.

Table 1: List of the main SGAN hyperparameters for the results shown in this paper. N^i represents the current layer. Most paramaters were taken directly from the paper training height map data using SGANs [Spick et al. 2019]

Parameter	Value
Optimizer	ADAM
Learning rate	0.001 Decayed $(0.001 * 0.9986^{(epochs)})$
Momentum	0.5
L2 regularization	1e-5
Batch size	32
Depth (N)	6
Discriminator Filter's	(3,3) * $2^{(N^i + 6)}$
Generator Filter's	(7,7) * $2^{(N^{max} - N^i + 6)}$

A final processing step was to overlay the colour channels into one 4 colour channelled image, with the first 3 channels (RGB) corresponding to the texture. The final colour channel (A) would then contain the height data of the region. With the data in this format, the filters of the convolutions in the discriminator of the GAN will learn from both texture and height data values. An example of the stacked data can be seen in Fig. 2c. This also alleviates issues surrounding whether the data would align correctly from two separate data sources. Though it was found to align correctly for the areas tested and shown in this paper, there could potentially be other areas that do not align as perfectly as the areas shown here.

3.2 Training and Initial Outputs

The data was trained using a spatial GAN [Jetchev et al. 2016] model with non-spatially dependant feature learning. This section will discuss the training process, hyperparameters and important optimizations when training a GAN.

Each iteration of training consisted of a pass of 32 random crops of the input image. These region crops built-up feature data understanding in the generator over time. The network hyperparameters can be seen in Table 1 and more visually in Fig. 3. These parameters were explored in a previous paper and found to produce the most optimal visual results for this type of data [Spick et al. 2019].

Assisting the performance of the network is a large struggle in generative deep learning, depending on the type, structure and complexity of the data - more often than not, the generator and discriminator will tend to outperform each other. There were three important steps that were taken to ensure a smoother learning process that would result in more coherent image results; decayed learning rate, early stopping and discriminator/generator training balance.

The decayed learning rate is a type of optimization that is used in many deep learning models and is not specific to GANs [Fritzke 1997]. This technique decays the learning rate hyperparameter

(a) Height map (b) Texture (c) Height and texture data layered

Figure 2: A randomly sampled height map image (a) and it's aligned corresponding texture map (b) acquired using google maps - alongside the final training data texture (c) showing alignment of height values with the texture pixel values.

overtime/epochs. As the learning rate decreases, the factor of the change from the network's updates is far smaller. By performing this decay process slowly throughout training, allows the use of a larger learning rate at the beginning of training, with a smaller learning rate being achieved close to the end of training to avoid large changes in the weights. This produced outputs of far higher visual fidelity, with a much less stochastic training process.

An example of the learning rate over a number of epochs can be seen in Fig. 4 with the learning rate being calculated as $0.001 * 0.9986^{(epochs)}$ providing a close to exponential decay. A value of slightly less than 1 was used as the coefficient of decay to provide a very slight reduction in learning rate over epochs.

An important part of deep learning is knowing when to stop training a model. Images are inherently difficult to statistically understand, unlike processing other data types. Early stopping is a way of determining when the model is trained to a point that any further training could only cause detrimental effects. The network training took into account the constant loss of the generator, where it was normal for the loss to steadily decrease as the discriminator improved, until a point at which loss plateaued. Therefore, the gradient of the loss was calculated constantly throughout training for a window of 25 previous epochs, when the gradient was 0 ± 0.01 for 10 consecutive epochs, the network training was halted. These values were found to be accurate when analysing a network that was trained for an exaggerated amount of time - where once the network's generator loss started to plateau there were no witnessed or statistical improvements in the image quality - and in some cases, degeneration occurred.

Usually, the end of training values were signalled at epoch 600-700 of training. This can be confirmed with the use of statistical measures of similarity. For each output of the network, the statistical comparison values using structural similarity (SSIM) and mean squared error (MSE) were used. Their results averaged across 10 trained regions can be seen in Fig. 5, where a stagnation in change can be seen at around the 600 epoch mark. SSIM is a method of detecting similarity in structures between two images, this detects additional information that might not be attributed with MSE.

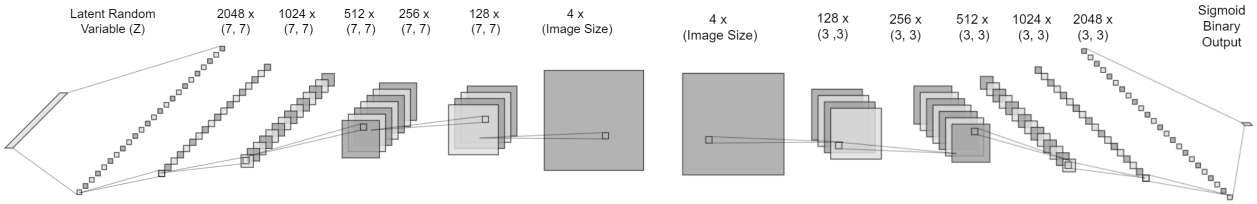
Table 2: Median blur kernal size against change in structural similarity index for an average of 10 output regions. Used to discover optimal median blur to apply to post process height map channel of output image. Value chosen where the SSIM difference (change in SSIM) was minimal against visual data loss.

Kernal Size	SSIM	Change in SSIM
1	0.417	0.110
3	0.527	0.031
5	0.558	0.021
7	0.579	0.015
9	**0.594**	**0.011**
11	0.597	0.003

Where MSE attempts to perceive the actual pixel error. SSIM is an important measure with the data in this paper, as the structural features of the data are not spatially dependent, meaning they can be located anywhere in an output image.

Lastly, a technique used to optimally train a generative network is the balancing of one over-performing network compared to the other. In our case, the loss of the discriminator was always far lower than that of the generator, indicating that the generator part of the network was struggling, or simply the discriminator was getting too good at predicting real against fake data. In this case, the solution was to train the generator twice for every update of the discriminator, this meant the discriminator loss reduced far less dramatically.

Post-processing of the images was performed to ensure that any inherent noise from the training process would be removed. This was important when attempting to keep the structural soundness of the terrain. If there was slight noise especially in the height map then the rendered displacements at the point of interference would look out of place. This was not as important for the texture map as this did not affect the dimensional space of the render.



(a) Generator **(b) Discriminator**

Figure 3: Architecture of generator (a) and discriminator (b), showing the number of filters and kernel size. Output image size of generator (a) is dependant on the latent random variable size. Real images and generated images are iteratively passed through the discriminator.

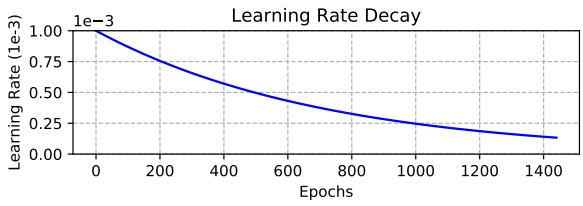


Figure 4: Learning rate value decayed over epochs

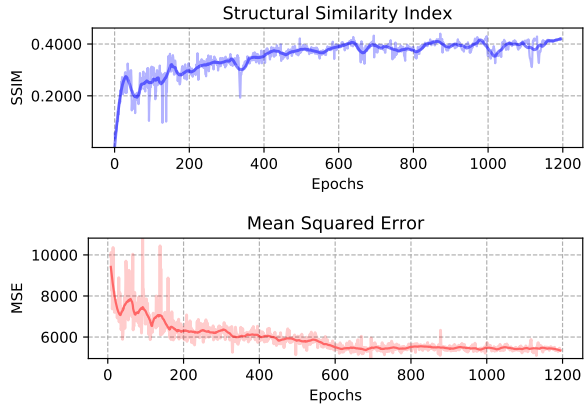


Figure 5: Analyse over epochs of the average structural and MSE similarity of 10 training regions. Determines a point at which results no longer visually increase in quality.

SSIM was used to statistically choose the most optimal median blur kernel size, SSIM plateaued at a kernel size of 9 with a large jump in similarity from 3-9, without removing too much of the output images quality. A selection of kernel sizes and corresponding SSIM values can be seen in Table 2. The change in SSIM for larger kernel sizes affected the coarser details of the heightmap too aggressively to warrant using.

4 RESULTS

In this section, the results of the network will be shown, followed by a comparison of outputs from the Spatial GAN with previous approaches for texture synthesis using a CNN [Gatys et al. 2015] and the commonly used DCGAN [Radford et al. 2015].

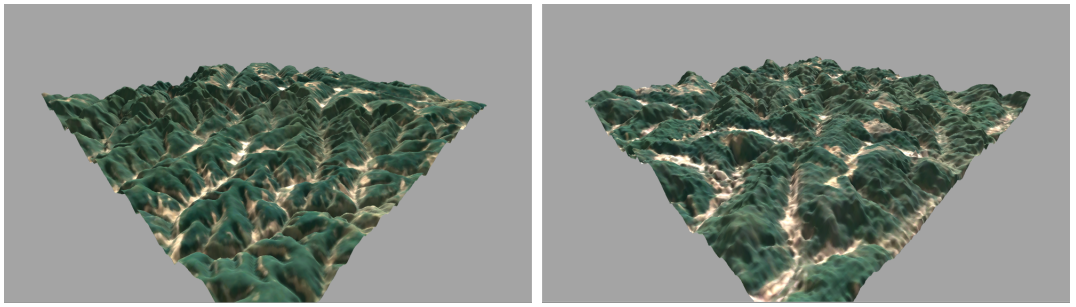
Any 3D rendered images shown here were rendered in Unity with no added post-processing effects - except the applied median blur to remove noise as discussed in the Section 3, where a kernel of (9,9) was applied to just the height data (alpha channel) to remove resulting noise between displacement points, the texture data was left unaltered from the output.

Table 3: Structural similarity index (SSIM) and mean squared error (MSE) analysis of height and texture data used to render the comparisons in Fig. 6 against the base line region.

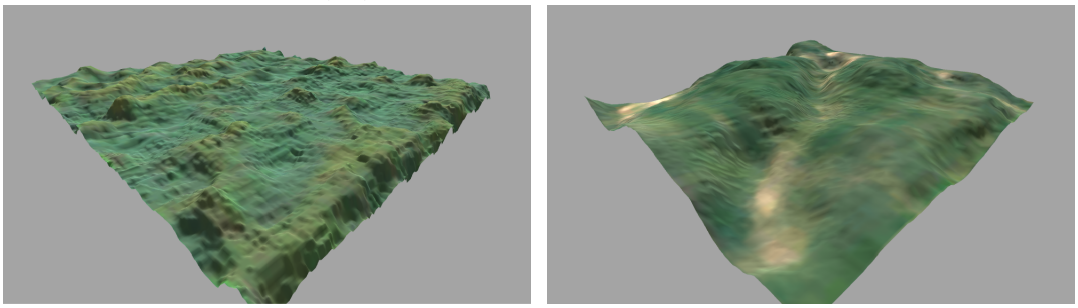
Method	SSIM	MSE	Average Fidelity Loss Compared to SGAN
SGAN	0.587	5876	-
DCGAN	0.410	10515	43%
Gaty *et al.*	0.310	11948	70%

4.1 Method Comparison

The results in this paper compare the proposed method of using Spatial GANs to other deep learning techniques that have been previously used on texture generation. The examples in Fig. 6 show the baseline rendered region (a) and three other regions comparing common deep generative techniques, SGANs (proposed) (b), DCGAN (c) and the CNN method (d) used by Gatys *et al.* There is an obvious advantage to the proposed SGAN generation. This is largely due to the inability to upsample to a larger resolution using DCGANs due to the use of an architecture without fully connected layers in the SGAN. Though DCGAN does capture underlying features at a very broad level, there is a distinct lack of low-level detail that the original region contains that DCGAN fails to learn. Furthermore, Gatys method fails to learn any of the complex features of the terrain, though when the input data was heavily down-sampled there was a better understanding. Though to keep the results unbiased the results are shown for an input training region of size 1300 x 1300 pixels, the size of the original data points. In Table 3, each region from the generative methods are compared to the baseline using MSE and SSIM. This further extends the above comparison on which generative technique produces the better results.

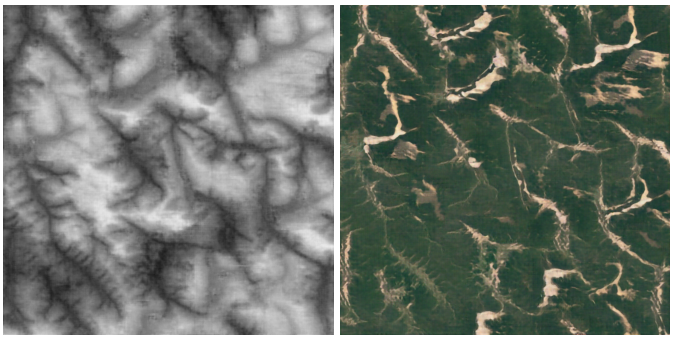


(a) Baseline (Training region) (b) Proposed SGAN



(c) Gatys et al. (d) DCGAN

Figure 6: Visual comparison of 3 methods with the baseline training region rendered as a 3D terrain with generated texture.



(a) Generated height map (b) Generated texture

Figure 7: Height map and corresponding generated texture for a trained region - with the input data for these generated textures shown in Fig. 2c.

4.2 Outputs

Each terrain rendered in this section contains no additional detail enhancements. They were rendered purely in Unity with the texture map laid directly on top of the rendered height map. Figs. 8, 9 and 10 show a random sample of generated data (b) from fully trained networks given the baseline region as the training data (a).

Each network trained to generate these renders output a 4 channel image which was separated into two images of equal size. One with the elevation values used to displace the terrain, and the other used to map colours from the texture to the displacement. Each result required one network to be independently fully trained on one specific target region - the resulting renders can be seen as a collection of distributed learned features from the generative network.

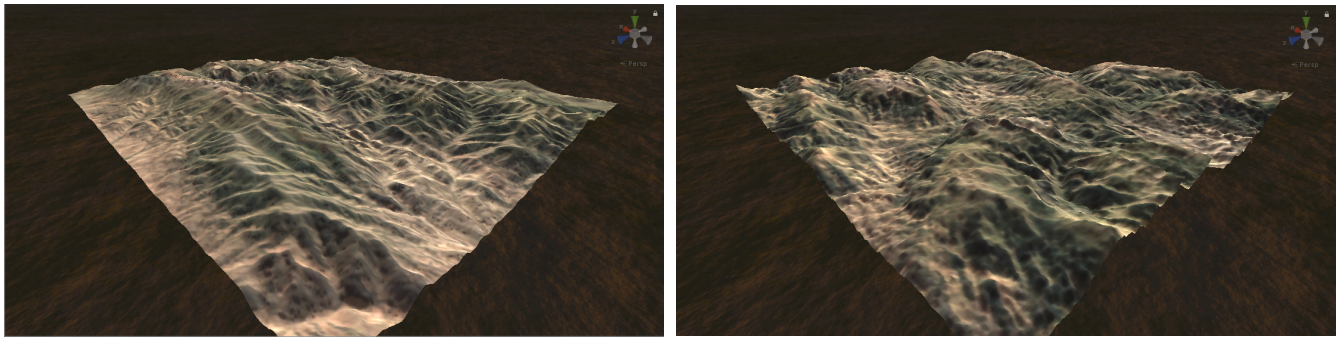
5 DISCUSSION

Each result can be treated independently with its derived region, this section will overview each region and explain the strengths and weaknesses of the resulting renders. The regions chosen were to select a wide range of shapes of terrain and accompanying texture structures.

The 2D texture height map and accompanied texture were used to calculate SSIM and MSE with their derived regions, this can be used to discover how similar the outputs will appear.

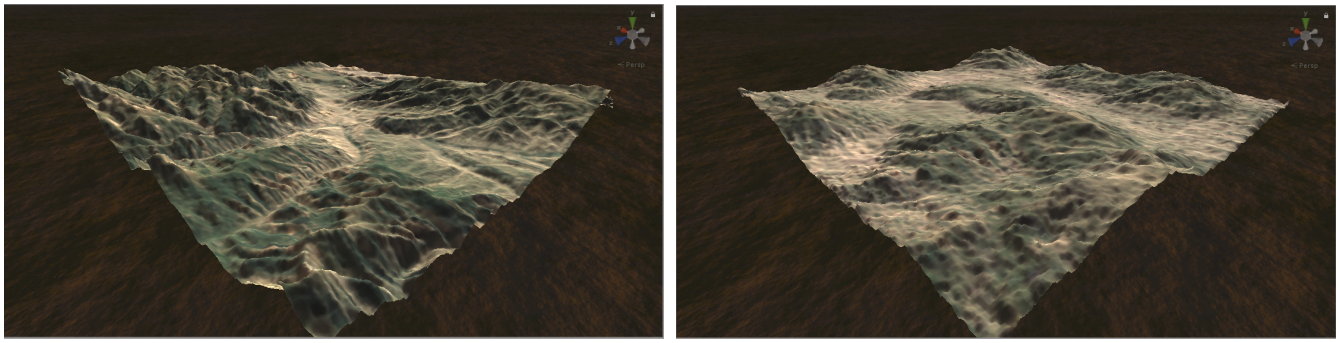
Figs. 8, 9 and 10 all visually appear similar to their derived region, justified further with an average 0.48 similarity index for all results. There is a disparity in the overall feature learning of the method, this is due to the size of the filters and depth of the network. Where the deeper the network the more of the larger structures can be abstractly learned. Though it is clear that smaller features which can be imagined as the features that make up the large structures of terrain are represented well. These strengths and weaknesses are well discussed in previous work and a prominent issue [Jetchev et al. 2016; Spick et al. 2019]. With the increasing use of higher-powered hardware, the depth of the network could be increased, in turn increasing the detail of larger structures of the outputs.

Further justification was explored using the similarity of colour pixel values, calculated using a difference in a colour histogram comparison of the exhibited colour values. For all resulting images, the colour distribution similarity was above 85%. This justifies the



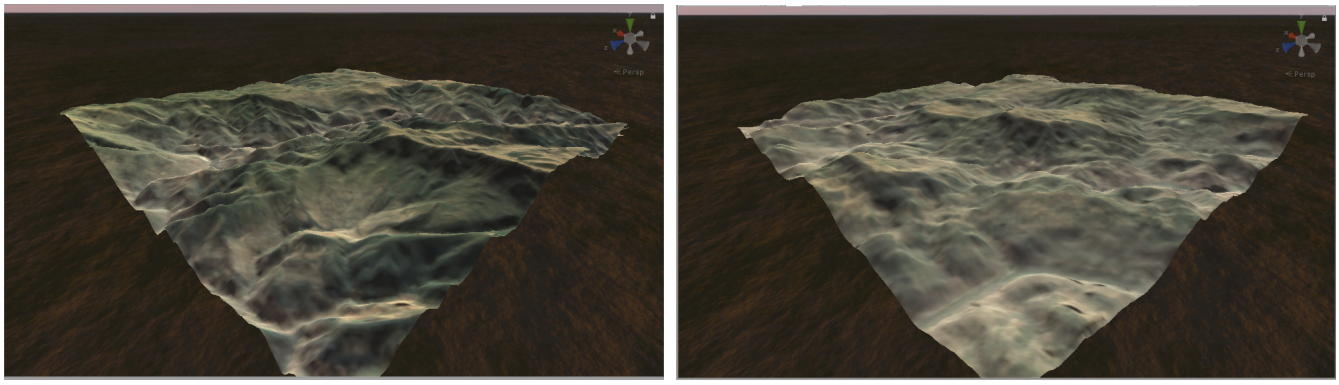
(a) (b)

Figure 8: Training region (a) and generated region (b) - showing a predominantly desert region with several long connected peaks. Comparing (a) to (b) with an MSE of 7346 and SSIM of 0.40.



(a) (b) Generated region with full texturing

Figure 9: Training region (a) and generated region (b) - showing a region with multiple small hills and a large flat area. Comparing (a) to (b) with an MSE of 7883 and SSIM of 0.44.



(a) Original region (b) Generated region with full texturing

Figure 10: Training region (a) and generated region (b) - showing a region with wide shallow gradient hills with several ravines running through the terrain. Comparing (a) to (b) with an MSE of 8007 and SSIM of 0.59.

similarity of texture colours learned which is slightly more difficult to visually compare.

Certain data points proved difficult to train, creating unfavourable results. Input regions where the texture contained cloud or highly reflective terrain interference corrupted the learning of the texture part of the input. Though the elevation data was unaffected.

Based on the methodology, there were variations in the type of data that worked well with this approach. Regions of repeating

terrain features would be far easier to train compared to those with a variety of terrain features distributed across the region. This can be partially seen in the observable results. Fig 8 struggles to mimic the entire distribution of data and instead focuses on certain features, where the original region contains a large amount of sprawling collective terrain structures.

6 CONCLUSION & FUTURE WORK

Overall, results in this paper show an exploration of a more specific type of generative adversarial network for generating multiple data in one instance. The method proposed offers the use of generative deep learning to automate the process of environment generation - by creating the height data and texture data from a singular input of the desired region.

The paper builds on previous work of height map generation, using DCGANs and later SGANs for results that are more accurate for the use in rendering terrains. The proposed work in this paper extends earlier efforts with the inclusion of texture data, increasing the complexity of the data by 3 channels per pixel. This warranted a more in-depth analysis of training methods, to create a more stable training process. There has been a description of techniques such as early stopping, decayed loss and balanced network training etc. with these methods. Compared to the previous work on SGANs the visual fidelity of the outputs has been more coherent. Further analysis was conducted on previously state of the art techniques and has been shown to outperform these methods at the domain of texture generation.

The novelty of the work has the ability to change how designers create environments for games and simulations. Previously authoring tools such as Unity, or the use of Perlin noise, have been harnessed to create terrains towards the author's intent. Although the method evaluated here may not fully replace the designers engagement in creating environments, the concept provides an automated way of generating a large number of variant environments, where these generations could utilise the aforementioned methods (Perlin noise) to add more fine-tuned detail, or entire features could be adapted to add areas pivotal to the design of the environment.

Alternatively, if designers require a quick way of generating 3D environments that follow a specific structure of a target region the proposed method would be reliable. The advantage of this technique over using the baseline region is the ability to generate a large number of the terrain regions, ideal for graphical systems that wish to iteratively vary the terrain environment while keeping to a design theme.

The analysis of optimization techniques has been discussed in the methodology section. With SSIM and MSE being used to statistically determine the quality of the outputs. These were reliable quantifiable measures of image "goodness" that correlated to how the results were visualised.

Further research will explore the use of GauGANs [Park et al. 2019], a recent exploration by Nvidia for image inpainting of segmented regions of training data, providing a method of low input drawing to create coherent outputs - with the expressed difference having full control over the features and structures that will appear in the drawn segments. Opposed to the method shown here, which only allows feature selection at a high level of which collection of features should exist.

Using this technique applied to the type of data that has been collected in this paper will result in a tool that will provide even more control over the distribution of learned features. Allowing authors to create large flowing landscapes of described regions. Another branch of future work will also investigate one limitation of this method, this being the seamless connection of arbitrarily large terrain regions, where some design choices require the generation of tillable regions that can create an endless environment. The use of conditional GANs [Mirza and Osindero 2014] [Isola et al. 2017] could be used to learn a seamless band between two regions, essentially creating a pseudo endless terrain that would require the training of a second generative network.

ACKNOWLEDGMENTS

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.

REFERENCES

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

Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. 2017. Learning representations and generative models for 3d point clouds. *arXiv preprint arXiv:1707.02392* (2017).

Alba Amato. 2017. Procedural Content Generation in the Game Industry. In *Game Dynamics*. Springer, 15–25. PCG Online and offline definitions. A few examples of games that used pcg History of PCG.

JJ Becker and DT Sandwell. 2006. SRTM30_PLUS: SRTM30, coastal & ridge multibeam, estimated topography. *Electronic journal. URL: http://topex. ucsd. edu/WWW_html/srtm30_plus. html* (2006).

Alain Fournier, Don Fussell, and Loren Carpenter. 1982. Computer rendering of stochastic models. *Commun. ACM* 25, 6 (1982), 371–384.

Bernd Fritzke. 1997. Some competitive learning methods. *Artificial Intelligence Institute, Dresden University of Technology* (1997).

James Gain, Patrick Marais, and Wolfgang Strasser. 2009. Terrain Sketching. In *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games (I3D '09)*. ACM, New York, NY, USA, 31–38. https://doi.org/10.1145/1507149.1507155

Leon Gatys, Alexander S Ecker, and Matthias Bethge. 2015. Texture synthesis using convolutional neural networks. In *Advances in Neural Information Processing Systems*. 262–270.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. 2017. Image-to-image translation with conditional adversarial networks. *arXiv preprint* (2017).

Nikolay Jetchev, Urs Bergmann, and Roland Vollgraf. 2016. Texture synthesis with spatial generative adversarial networks. *arXiv preprint arXiv:1611.08207* (2016).

Chao-Hung Lin, Po-Hung Tsai, Kang-Hua Lai, and Jyun-Yuan Chen. 2012. Cloud removal from multitemporal satellite images using information cloning. *IEEE transactions on geoscience and remote sensing* 51, 1 (2012), 232–241.

Benoit B Mandelbrot and John W Van Ness. 1968. Fractional Brownian motions, fractional noises and applications. *SIAM review* 10, 4 (1968), 422–437.

Mehdi Mirza and Simon Osindero. 2014. Conditional Generative Adversarial Nets. *CoRR* abs/1411.1784 (2014). arXiv:1411.1784 http://arxiv.org/abs/1411.1784

Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. 2019. GauGAN: Semantic Image Synthesis with Spatially Adaptive Normalization. In *ACM SIGGRAPH 2019 Real-Time Live! (SIGGRAPH '19)*. ACM, New York, NY, USA, Article 2, 1 pages. https://doi.org/10.1145/3306305.3332370

Ken Perlin. 1985. An image synthesizer. *ACM Siggraph Computer Graphics* 19, 3 (1985), 287–296.

Alec Radford, Luke Metz, and Soumith Chintala. 2015. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015).

Noor Shaker, Julian Togelius, and Mark J Nelson. 2016. *Procedural content generation in games*. Springer.

Ryan J Spick, Peter Cowling, and James Walker. 2019. Procedural Generation using Spatial GANs for Region-Specific Learning of Elevation Data. (2019).

Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. 2018. Procedural content generation via machine learning (PCGML). *IEEE Transactions on Games* 10, 3 (2018), 257–270.

Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M Lucas, Adam Smith, and Sebastian Risi. 2018. Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, 221–228.

Andreas Wulff-Jensen, Niclas Nerup Rant, Tobias Nordvig Møller, and Jonas Aksel Billeskov. 2017. Deep Convolutional Generative Adversarial Network for Procedural 3D Landscape Generation Based on DEM. In *Interactivity, Game Creation, Design, Learning, and Innovation*. Springer, 85–94.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. 2017. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*. 2223–2232.