Dual Critic Conditional Wasserstein GAN for Height-Map Generation

Nuno Ramos
Instituto Superior Técnico
University of Lisbon
Lisbon, Portugal
nuno.m.ramos@tecnico.ulisboa.pt

ABSTRACT

Traditionally, video-game maps are either made by hand, which is a very inefficient process requiring many man-hours, or made using Procedural Content Generation (PCG) techniques, which rely on a predetermined algorithm to generate every feature of the map. This approach is flawed in a multitude of ways: creating the algorithm is an arduous process, the results lack realism and it's hard to create more complex geographical structures, such as bays, peninsulas, or diverse archipelagos. More recent studies have tried an approach using Deep Learning algorithms, which have their own limitations. Most importantly, these algorithms take away the creative freedom of the designers. To circumvent this problem we propose a system that transforms low fidelity sketches into realistic height-maps through a Deep Learning model we call the Dual Critic Conditional Wasserstein GAN (DCCWGAN), thus providing high visual quality without removing control from the user.

CCS CONCEPTS

• **Computing methodologies** → *Neural networks.*

KEYWORDS

Height-map, Deep Learning, Image-to-Image Translation, GAN, Conditional GAN

1 MOTIVATION

Maps are a crucial part of most video-games: in exploration games different areas can provide interesting challenges for the player, in strategy games more defensible geography can be considered a critical asset. If a video-game contains a map it is a reasonable assumption that the quality of the player experience is somewhat tied to the quality of this map.

In the video-game industry maps are made in one of two ways: either by hand, which is a very time-consuming task, whose results depend both on the prowess of the designer, who designs the challenges present in the map, and the artist, whose job is to implement those ideas; the other method is to procedurally generate those maps, which involves designing an algorithm to create a map from a set of parameters and random values; this approach has it's own problems, namely that it's results have sub-par quality and take control away from the designer, who, for the most part can no longer specify exactly which geographic formation appears where. The disadvantages incurred by these procedural content generation algorithms mean that they are used mostly for tile-based maps, or maps that are, in practice, infinite, meaning that it's impossible for them to be hand-crafted. Recent studies [10] have tried to overcome the limitations of traditional PCG techniques by implementing machine learning algorithms, using real-world geographic information to train networks created for this specific purpose. While this approach produces promising results it has a fundamental flaw: similarly to PCG techniques, it removes control from the developers. In order to create a level with a specific layout a developer would have to sieve through a large number of images generated by the network until the right one was found, and even then it could differ from the initial vision.

There have been many recent breakthroughs in using machine learning algorithms for image generation [2, 6, 9] and image-to-image translation [5, 11–13], particularly in transforming low-level images into realistic depictions of the same content. Coupled with the fact that machine learning techniques themselves have also been in a steady state of evolution we show that it is possible to leverage this technology to develop a tool that allows developers to draw a rough sketch of a map that will be transformed by deep learning model into a realistic version of the terrain depicted in the sketch, thus maintaining the visual quality while reducing effort without having to sacrifice control.

2 BACKGROUND

2.1 Video-game maps

Video-game maps are made up of multiple different components, or layers [7]. The more notable inclusions in this category are:

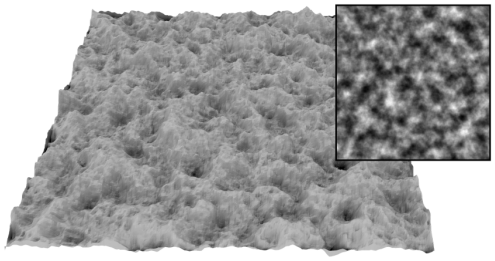

Figure 1: Height map created from PCG techniques. Image from [7].

- Height-map: contains the height of the terrain.
- Hydraulic map: contains the water in the terrain, rivers, lakes, etc...
- Vegetation map: determines which areas have trees and other vegetation.
- Biome map: determines to which biome each region belongs to.

The above is list is by no means comprehensive, different games have different requirements and may use other types of maps to satisfy those requirements. Of all the components listed above the most complex is by far the height-map, this makes it the hardest to recreate in a believable way, additionally, elevation tends to play a large role in how the player interacts with the game - high ground tends to be more valuable as it is easier to defend or rocky mountains may provide a direction unlikely to be attacked through. This creates a difficulty for the developers and designers of the game: on one hand the map must appear realistic, on the other it must provide interesting gameplay; a realistic map with no interesting features will oversimplify the game, while a map with varied features but that appears unrealistic will ruin the players' immersions.

2.2 Generative Adversarial Networks

A Generative Adversarial Network (GAN) is an architecture for machine learning that can create data similar to the input it is given. A GAN contains two distinct networks, a Discriminator (D) and a Generator (G), that function with opposite goals, hence the term adversarial [4]. The Discriminator's job is, for any given input, to distinguish whether it came from the original data-set (ground-truth), or if it has been fabricated. The Generator, on the other hand creates random items to be evaluated by the Discriminator. In essence, because these two networks learn off of each other's outputs each one's improvements will force the other to improve. It is worth noting that while the theoretical architecture uses two separate networks this is not the case in practice, in which a stacked model of the two networks is created for ease of training.

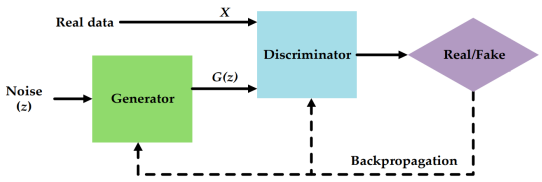

Figure 2: Generic structure of a GAN. Image from [3].

GAN's are trained by creating a batch comprised equally of real images from the data-set and of images created by the Generator. The Discriminator then assigns it's predictions to the given batch and is trained through back-propagation based on how correct each prediction was. After this step a second batch of images is created, though this time comprised entirely on predictions from the Generator. These predictions are then evaluated by the Discriminator and back-propagation is used to train the Generator, based on how close the images were to being classified as real images by the Discriminator.

A key detail about the GAN architecture is that there is the Generator allows for an input, a random vector, usually labeled z. This vector is necessary because there needs to be point of randomness within the system, otherwise the Generator, which is a deterministic system, would be able to create a single image.

2.3 Conditional GAN

While GAN's are one of the best deep learning architectures for image generation they lack a very important feature: it is impossible to guide their output within the space of possible results. Conditional GAN's provide a way to remedy this limitation [8]. Traditional GAN's require a single noise vector input, but it is impossible to predict the output of the network given only the noise vector, other than performing the same calculations that will be performed by the Generator of the GAN.

In order to guide the image generation both the Generator and the Discriminator need to accept a new input, which is what will be used during inference to guide the image generation process. This additional information may be a class labeling, allowing, for example, the generation of specific digits when using the MNIST data-set. More relevant to our research is to use a smaller or simpler image as this additional information, which is one of the most commonly used methods of image-to-image translation.

To train a cGAN real samples are complemented with their respective additional information, when inputted into the Discriminator, with the goal that it will learn this relationship. When training with fake samples the Generator is given a noise vector, but also this additional information. The output of the Generator is then used to train the Discriminator, along with the information the former used. The goal of the Discriminator is then, not only to discriminate between real and fake samples, but also to ensure that the image and the additional information display the same relation that is observed in ground-truth samples.

2.4 Wasserstein GAN

The Wasserstein GAN, or simply WGAN is a model suggested by Arjovsky et. al [1] to combat a problem very common when training GAN's: because it is easier to differentiate between real and fake images than it is to create realistic looking images often the Discriminator becomes proficient too fast, ceasing to provide useful information to the Generator; one can imagine that a well-trained Discriminator can not only label fake images as such, but also every similar looking image, meaning that the Generator is unable to improve. WGAN's exist as a way to combat this issue: instead of training a Discriminator it uses what is known as a Critic (C). The Critic's job is to always provide information to the Generator, with the usefulness of this information increasing as the Critic is further trained, something that is not true for regular Discriminators. This is done by, instead of labeling images with either 1 or 0, the Critic rates the realness of the image, known as the Wasserstein estimate. In this model the loss functions are defined as such:

- Critic loss: (average critic score on real images) – (average critic score on fake images)
- Generator loss: -(average critic score on fake images)

This definition for the Critic's loss creates a function that doesn't saturate and converges to a linear function, providing cleaner gradients than a generic Discriminator. While this is the largest change there are other subtleties to this model:

- The Critic is trained n times more than the Generator. The authors suggest a value of 5 batches of training for the Critic for every batch the Generator is trained on. This is done as a consequence of the fact that results always improve for a

better trained Critic, which would not be the case with a Discriminator.
- The weights on the Critic are clamped to a limited box after every batch. The range suggested by the authors is $[-0.01, 0.01]$. If the magnitude of this range is too high the Critic will take too long to converge, while a value too low will render the Critic unable to learn some features.

2.5 Map generation from GAN-based models

The work of Nunes et. al [10] focused on experimenting different models of deep learning models to investigate which would perform the task of creating maps for strategy games better. In total, four different models were explored: Deep Convolutional GAN (DCGAN), Wasserstein GAN (WGAN), Progressively Growing GAN (ProgGAN), and a VAE model with adversarial loss. Of the four models tested, the DCGAN and VAE + GAN provided results inferior in quality to the ProgGAN and WGAN. These last two models provided results with great visual quality, with the ProgGAN being more efficient to train than the WGAN.

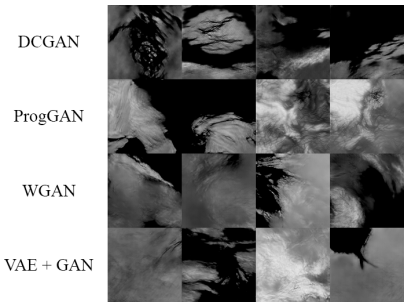

Figure 3: Sample of results obtained by the researchers of [10].

3 SYSTEM OVERVIEW

Previous researchers [10] focused heavily on testing multiple Deep Learning architectures and comparing their results. This research used that knowledge and opted to build only upon one of the most successful models showed, the Wasserstein GAN [1], iterating the model based on results achieved. The WGAN was chosen over the ProgGAN as, despite having a longer training period, it is more simple and therefore, in our opinion, more likely to maintain good results in spite of changes added.

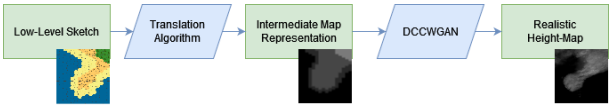

Figure 4: Data flow for the system, user creates low-level sketches (left) and the system outputs Realistic Height-maps (right).

The data flow for our system (illustrated on Figure 4) begins with the low-level sketch, which is created by the user and exists only during inference. While we illustrate a specific type of sketch it is important to stress that the system is very easily adapted to use a different tool to create sketches. The low-level sketch is then used as input to a translation algorithm, which transforms it into what we call the Intermediate Map Representation (IMR) format.

For our system to function there needs to be a format that will be used as input by the user, but is also able to be created from the ground-truth data-set, so as to compare if the content matches during training, this format is the Intermediate Map Representation (IMR). While it would be possible for a user to create maps directly in the IMR format, we opted to create it from an existing map-creating tool, using a simple, deterministic algorithm. Users wanting to use a different map-creating tool need only to create this translation algorithm. The IMR format exposes the average height of each cell in a hexagonal grid. The IMR outputted by the translation algorithm is then used as input to our deep learning model, the Dual Critic Conditional Wasserstein GAN (DCCWGAN), both during inference and training, resulting in the height-map.

3.1 DCCWGAN

In order to choose an architecture for our deep learning model we started by dividing the problem into two distinct learning processes:

- Create realistic maps.
- Create maps whose content corresponds to the given input.

Given this division we opted for an architecture with a Generator, but two distinct critics, one for each of the necessary learning processes. The first of these Critics evaluates only the visual qualities of the images generated, and forces the Generator to create more and more realistic images; for this reason we call this the Realism Critic. The second critic is responsible for evaluating how well the contents of the generated maps correspond to the input used in their generation; this critic is called the Conversion Critic. Training is done using one Critic at a time, depending on the current loss. The decision of which Critic to use is explained in further detail in Section 3.6.

3.2 Generator

The Generator is the network responsible for generating images. It accepts two separate inputs: the IMR and a noise vector. The IMR controls the general layout of the final result, while the noise vector, similarly to other GAN's, provides a source of entropy, adding a layer of randomness to the output and allowing the same IMR input to generate multiple different results.

Figure 5: Structure of the Generator.

Figure 5 represents the complete structure of the Generator. All deconvolutional layers use kernel size of 3 and stride of 1, except for the layer following the IMR input, which uses a kernel size of 3 and dilation rate of 2, and the final convolutional layer, which uses a filter size of 5.

There are a few details worth pointing out about the architecture of our Generator: first, we start by increasing the number of channels on the IMR branch *before* it is concatenated with the noise branch. The justification for this is that when concatenating these two branches the weight given to the noise branch is greatly reduced, as it is now responsible for only 1 of the 129 channels. While this is not critical, and the network is able to learn to give less importance to the noise, we found that this simple change positively affected the final results. Another decision that may appear counter-intuitive is the number of channels across the network, starting at 129 on the main branch, then decreasing to 64 only to expand gradually to 256 before again decreasing gradually to 1. While we tested different configurations, including removing the intermediate layers such that the model goes directly from 129 channels to 256, we found this configuration to yield optimal results.

3.3 Conversion Critic

The Conversion Critic, unlike a generic WGAN critic, receives a paired input: a real or fake image and an IMR. The IMR given is either generated from the real data-set or, in the case of fake images, the IMR used as input for the Generator. The purpose of this network is to discriminate whether the general content of the image matches that of the IMR given. In the case of real images the pair should match since the IMR is created from the image itself; in the case of fake images the network attempts to correct the generator to force this content matching.

The data flow for training the Conversion Critic can be observed in Figure 6. The left side of the image refers to training with fake images, while the right side of the figure refers to training with real images. In essence, the goal of the Conversion Critic is to learn the opposite of connection A, in other words, how to generate a map given only it's IMR representation.

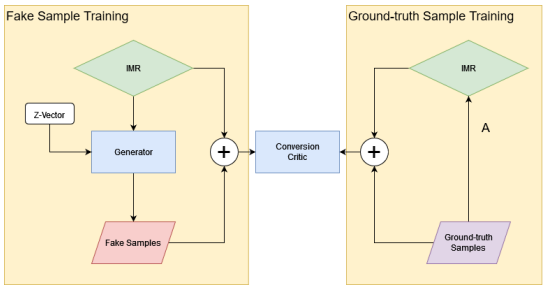

Figure 6: Training of the Conversion Critic.

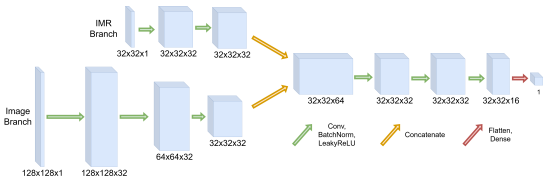

Figure 7: Structure of the Conversion Critic.

Figure 7 represents the complete structure of the Conversion Critic. All convolutional layers use a filter size of 4, except for the layer following the IMR input, which uses a kernel size of 3 and dilation rate of 2, and the final convolutional layer, which uses a filter size of 5. When image width and height decreases this is done using a convolutional layer of stride 2.

The Conversion Critic has a much simpler task than that of the Realism Critic, therefore requiring less parameters and less total memory to train. This network has two separate inputs that then concatenate channel-wise. It is worth noting that the Image Branch of this network downsamples very rapidly, this is because, due to the way in which we defined the IMR format, it is much easier to confirm that the two branches have similar geographical content at a lower resolution.

3.4 Realism Critic

This critic functions exactly as a critic in any generic WGAN. It takes as input either a real or fake image and outputs a score of the realness of the image received. Unlike the content critic, this critic does not receive the IMR, and therefore judges only the visual quality of the image, and not its accuracy.

Figure 8: Structure of the Realism Critic.

Figure 8 represents the complete structure of the Realism Critic. All convolutional layers in this critic use a stride value of 1 and filter size of 4. While on a surface level this network appears smaller, as it contains less layers, it contains over 7 times more parameters. This is because while the Conversion Critic only guides the broader strokes of the image, the Realism Critic is responsible for the finer detail, which is a more difficult task that will be held to a higher standard by the end user.

A different approach to the architecture of this network would be to have more channels and more groups of Convolutional, Batch Normalization and LeakyReLU activation layers, but intertwined with some downsampling layers, so as to reduce the required memory. We tested variations of this architecture and found the results to be inferior.

3.5 The data-sets

As a starting point we were graciously given the data-set used by the researchers of GAN-Based Content Generation of Maps for Strategy Games [10], which is itself a filtered and augmented version of the Shuttle Radar Topography Missions (SRTM) [1] data-set, created by NASA. This data-set is used as the ground-truth for this research, but it requires a pair to be used as input of the model.

The pair to the ground-truth data-set is the Intermediate Map Representation (IMR), and is used both in training the Deep Learning model and is what the user will create as input to the final system. This format is a simple grid of values corresponding to the average elevation value in that area. It is important to note that

[1] https://www2.jpl.nasa.gov/srtm/

these values belong to a limited number of options, the amount of which was carefully chosen: a number of classes too low will make the ground-truth data-set not match the IMR format as closely, giving an unpredictable output while simultaneously limiting user freedom; a number of classes too high will, in turn, difficult training as some values may become too rare in the ground-truth data-set. The number of values existent in the IMR format is five. While these class values do not have definitive real-world equivalents we can still infer what they represent most commonly, note that the values present in the IMR format are in the range $[0, 1]$, the same range used by the height-maps during training. The classes present in the IMR format are the following:

- **0.0:** Ocean, this is the only class whose real-world counterpart is well defined.
- **0.1:** Coastlines, river deltas, swamps and other landmasses that tend to be partially submerged. While this class has the lowest absolute frequency it is very important as it is the lowest non-zero value and, as such, serves to distinguish landmasses regardless of how low they are.
- **0.2:** Plains, forests, deserts and other types of land of mid-low altitude.
- **0.3:** Plateaus, hills and other terrain of mid-high elevation.
- **0.5:** Mountainous terrain. It is worth noting that any sufficiently tall mountain on the ground-truth data-set will map to this value, regardless of its elevation, as such, it is impossible for the user to specify the height of a mountain. We consider this limitation to not be too detrimental to the end result, and the existence of classes with even higher values is difficulted by the fact that it would not have a sufficiently significant sample size for the model to train with.

To create the IMR format, we analyze each image in the ground-truth data-set and obtain the average elevation of all pixels that correspond to each hex cell. Each of these averages is then compared to the list of possible values for the IMR format and is attributed one of the two closest values randomly based on how close it is to either, according to Algorithm 1.

Algorithm 1: Mapping of area elevation to IMR class.

```
for hex do
    diff = highest_lower_IMR_class -
      lowest_higher_IMR_class
    r = random[0, 1] * diff
    if r + highest_lower_IMR_class < hex then
        class = lowest_higher_IMR_class
    else
        class = highest_lower_IMR_class
```

It is important to note that we do not simply map the area's elevation to the closest IMR class, but instead introduce some randomness to this conversion. This randomness makes the paired data-set less homogeneous, removing examples where the whole image would be a single class, a common occurrence with river deltas or plains. River deltas are particularly problematic as they could become the ocean class (zero elevation) despite containing land, which in turn would make the model generate similar land masses where only ocean was intended. In Figure 9 we can observe an example of this phenomenon: despite there being two separate landmasses, were the conversion to happen using only the closest class the top-most landmass would not be translated into the IMR, in turn the Generator may learn that ocean could occur with this type of landmass, which is not desirable.

Figure 9: Left: Original image from ground-truth data-set (brightness has been increased for ease of readability); middle: IMR with no randomness; right: IMR using randomness.

An important detail to note is that the topology of the IMR format is different to that of an image by the simple fact that it is based on hexagons, therefore having six neighbouring cells instead of the four present in a square grid. This presents a problem when implementing the system as convolutional operations are, usually if not always, only available for square grids. To circumvent this issue, whenever the IMR format is used in a network, we build a square grid where each cell is a 2x2 square, offset vertically by a single pixel every other row, thus maintaining the topology of the original hexagonal grid. Note that all depictions of the IMR format are using this offset square grid. As a consequence of having the this format as input the first convolutional layer must have a dilation rate of 2, as otherwise it would only be able to evaluate the current cell and two of it's neighbours; having a dilation rate of 2 allows the cell and four of it's neighbours to be evaluated. Note that this means that each original hex cell is evaluated 4 times, with the top and bottom cells being present in all evaluations and each pair of left/right cells only once.

3.6 Training

The training of our system, for each GAN, is identical to the algorithm used to train any generic WGAN, as presented in Algorithm 2, the only difference from this algorithm is that our system uses not one, but two separate GAN's, and as such, it is necessary to decide which GAN to train at any given point, which is done through Algorithm 3.

The tasks required of each of the Critics are of different scales of difficulty, with the Conversion Critic having a simpler task. Theoretically it is possible to lower this latter Critic's learning rate and train both GAN's every epoch, however, this is neither easy to accomplish, nor is it optimal; a better strategy is to evaluate the current loss and choose a GAN to train accordingly. This approach has several advantages: it doesn't require a lower learning rate on the Conversion Critic since it won't be trained every epoch; it is flexible and adapts quickly to sudden changes in the loss function. It is important to note two aspects of how Algorithm 3 works: it depends entirely on the loss of the fake images, and not in the critic's evaluation of the real images, this is because, in theory, the loss of

Algorithm 2: WGAN training algorithm with $n\_critic = 2$ and $batch\_size = 64$

```
Normalize p_data between −1 and 1;
for epochs do
    shuffle p_data;
    half_batch = batch_size / 2;
    iterations = number of images of p_data / half_batch;
    for iterations do
        Choose GAN to train according to algorithm 3;
        for n_critic do
            z_vectors = half_batch samples from N(μ, σ²);
            real_images = half_batch images from p_data;
            fake_images = half_batch samples from
              G(z_vectors);
            y_real = half_batch size vector of value −1;
            y_fake = half_batch size vector of value 1;
            Train C with real_images labelled as y_real
              using gradient descent with Wasserstein
              estimate;
            Train C with fake_images labelled as y_fake
              using gradient descent with Wasserstein
              estimate;
        z_vectors = batch_size samples from N(μ, σ²);
        y_gen = half_batch size vector of value −1;
        Train G with z_vectors and y_gen labels using
          gradient descent with Wasserstein estimate;
```

Algorithm 3: Choice of GAN to be trained

```
for iterations do
    if last_conversion_fake_loss×2 >last_realism_fake_loss
      then
        gan = conversion_gan
    else
        gan = realism_gan;
```

the real images will continuously increase, therefore becoming a function of how much the network has been trained; the loss for the fake images of either network, on the other hand, is affected by the training of the other, thus, if the training of a network negatively impacts the loss function of the other, this will be corrected by changing which network is being trained. The second important aspect of how this decision is made is that it allows for a weight to be assigned, in the example shown, and the final value used is to give twice as much importance to the Realism GAN over the Conversion GAN, the reasoning being that the task of the Conversion GAN is less critical as the user will not notice if the content doesn't fully align, and the looser this restriction is, the more freedom allowed for the Realism GAN to improve the aesthetic aspect.

3.7 Architecture Variations

In this section we will go over several of the most important decisions made regarding the architecture chosen, describing in detail the possible options, outcomes tested and possible reasoning for the final result. The three decisions to be discussed are the following:

- Bilinear interpolation: use bilinear interpolation in the Generator in an attempt to overcome possible limitations of the IMR format.
- Forced Conversion Critic Scheduling: schedule which GAN to train in such a way that no GAN is left untrained for too many consecutive epochs.
- Per-epoch scheduling: schedule which GAN to train on a per-epoch basis.

3.7.1 Bilinear Upscaling of IMR. One possible improvement to the system would be to use bilinear upscaling in the Generator. Recall from the structure of the Generator (Figure 5) that it uses upsampling layers to go from the resolution of the IMR format to the resolution of the final height-maps. The upsampling layers in the final model use nearest neighbour interpolation, however, there is a theoretical justification for using bilinear upsampling: such an upsampling method would make for smoother features, something already more aligned with the features present in nature than the rougher changes in altitude created by the nearest neighbour interpolation method.

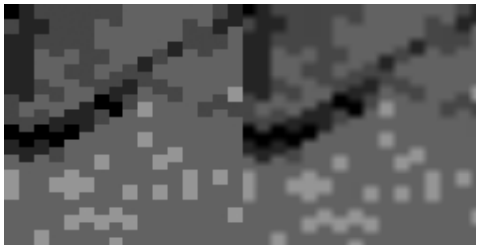

Figure 10: Comparison between original IMR and upscaled with bilinear interpolation.

3.7.2 Forced Conversion Critic Scheduling. The method for choosing which GAN to train, established in Algorithm 2 has a seemingly large drawback: it is not unreasonable to think that not training a network for various epochs would bring adverse results, given that the results of the Generator will be vastly different from the last time it was trained. While this is not necessarily implicit in the algorithm it is also not prevented, and through empirical experimentation we discovered that, for the optimal learning rates, it was common for the Conversion GAN to go hundreds of epochs without being trained once. To combat this issue we tested a maximum limit of consecutive epochs trained per GAN, therefore forcing the less trained GAN not to go too many epochs without training.

3.7.3 Per-epoch Scheduling. The algorithm for scheduling which GAN to train doesn't need to be run once per epoch: it may be run once for any arbitrary number of epochs, or, conversely, to any fraction of an epoch, down to a singular batch. We theorise that making this decision on a per-epoch basis or per-batch basis may have significant effects in the final results, as, the more granular this decision is made, the less each GAN may go without being trained, which may impact results positively. On the other hand, scheduling the training on a finer scale than a whole epoch may

cause each GAN to train on a fraction of the data-set, theoretically making each GAN train on non-overlapping halves of the data-set, though this extreme case is statistically impossible.

4 RESULTS

In this section we will discuss the most relevant experiments done and the results achieved. All tests were done on a server using a Intel(R) Xeon(R) W-2223 CPU with 98 GB's of RAM and two GeForce RTX 3090 GPU's, however, only one was ever used at a time.

All the experiments described are compiled with the RMSProp optimizer, with a learning rate of 0.00025 for the Conversion GAN and 0.0005 for the Realism GAN. Following the results of the researchers of [10] we attempted to train for 5000 epochs, however, some experiments were cut short to save time, in case their intermediate results were proving to be unsatisfactory.

4.1 Conversion Evaluation

In this section we will evaluate our model based on how well the content of the final height-map matches the content of the input from which it was generated. After acquiring results we determined that a simple empirical observation was enough to ascertain how closely these contents match. The reasoning behind this decision is that we believe that tools such as ours, that cater to human perception, are best evaluate by it.

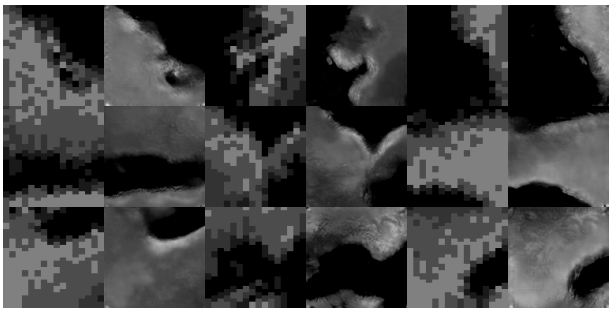

Figure 11: Examples of images generated from our final model.

In Figure 11 we can observe some maps generated from our final model. We believe these results to be of high quality, reproducing the given content successfully, with little artifacts or added noise. We can observe that the model maintains the general profile of the terrain while adding some texture. It should be noted that the contour of the coastline is kept identical to the IMR used, while the elevation within land differs slightly, fluctuating depending on the noise vector given, Section 4.1.1 shows in further detail the effect of this noise.

4.1.1 Effect of the Noise Vector. As described in Section 3.2 the Generator allows a random input vector, this vector requires a careful balancing of the model as, if the vector is not given enough importance, then there will be little randomness in the images, in the extreme case each IMR input can only generate a single output map. On the other hand, if the noise vector is given too much importance then the IMR input will not be respected and the output map will not represent the given input.

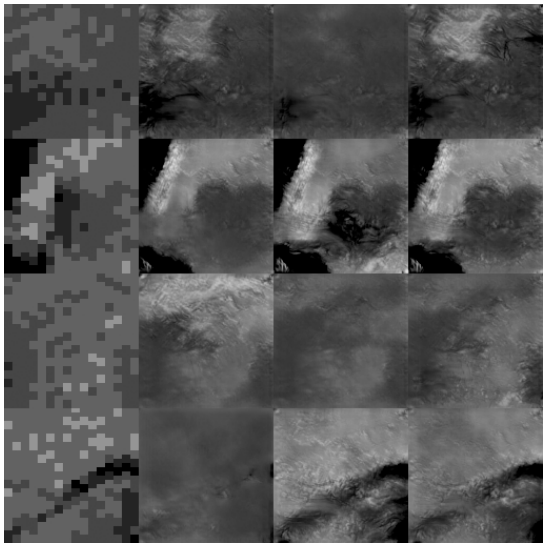

Figure 12: Influence of the noise vector on map generation. Left-most image is the IMR input from which the other images in the same row are generated.

From Figure 12 it can be seen that the noise vector has a significant impact in the topology of the maps created, while at the same time maintaining a the general content present in the supplied IMR file. This is a crucial feature of the system as some of the maps may have undesired features, in spite of their semblance to the given input. The existence, and correct functioning of the noise vector allows the user to maintain most features but receive a different result, with the noise vector acting as a style guide.

4.1.2 Validation data-set. In order to test how well the model creates any type of terrain, and to ensure it wasn't overfitting, we created a separate data-set, comprised of 32 hexagonal maps, that were subsequently translated into the IMR format.

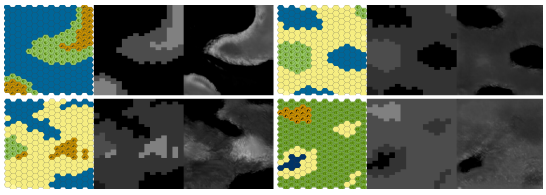

Figure 13: Examples of maps generated from the custom IMR data-set. For each set: Left: Original hex map. Center: IMR. Right: Resulting height-map.

Figure 13 represents some of the results obtained from the model when given IMR representations not present in the training data. This data is problematic to the model, not only because it was never observed, but also because it can express terrain that would be completely unrealistic. The quality of these results is lower than that of the observed data, as would be expected, however, when generating multiple samples of the same input some have quality on par with the previously observed results, further demonstrating the importance of the noise vector.

4.1.3 Artifacts and Shortcomings of the Generated Maps. Most maps generated do not contain significant artifacts or other structures that may giveaway the fact that these images are generated, however, some images do contain such structures, with the most common being a small islet, as present in Figure 14. While this type of terrain exists on Earth (and in the SRTM data-set, where the model learned this structure) it is not as common as it is in images generated from our model. The other main limitation of our model's results comes from the types of structures it is not able to reproduce, with the most flagrant omission being rivers. We believe that rivers are notoriously difficult for networks to reproduce faithfully, as they exist not only as local information (i.e. their shape), but they also present a more global logic - they flow from higher elevation positions to the ocean - and this logic may be harder for the model to learn. While some rivers are present in the results generated, they tend to remain in the same location, rather than from the source to the ocean, however, the more common result is that the network simply neglects the creation of rivers.

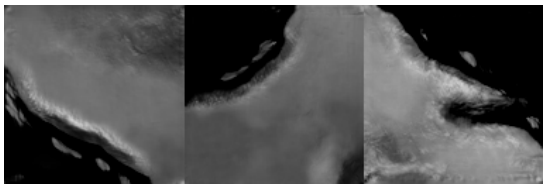

Figure 14: Formations generated along the coastline.

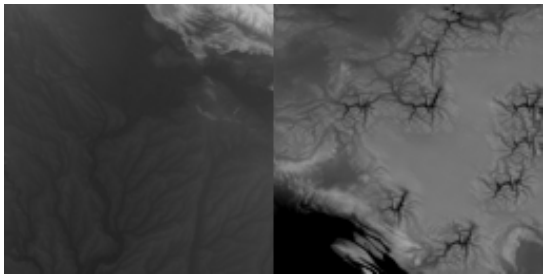

Figure 15: Left: Rivers as present in the SRTM data-set. Right: Rivers generated by our model.

4.2 Realism Evaluation

In this section we will explain our methodology for evaluating the results obtained on a visual basis, which we chose to evaluate through a short questionnaire, with the goal of confirming whether or not users can distinguish maps generated by our system from those present on Earth.

4.2.1 Test Overview. In order to evaluate our results on the goal of generating realistic images we conducted a user test where participants were shown 20 height-maps and corresponding 3D-renders, 10 of which from the NASA SRTM data-set, and 10 generated by our system. Participants were then asked to evaluate the origin of each map, using the sentence "This map represents geographic information from the Earth" and asking participants how much they agree with the sentence, in a Likert scale of 1 to 7; an example question can be seen in Figure 16. We considered the definition of "realistic" to be "representing things in a way that is accurate and true to life", therefore, our experiment is more successful the closer the user evaluations are, across both sets of data (ground-truth and system generated).

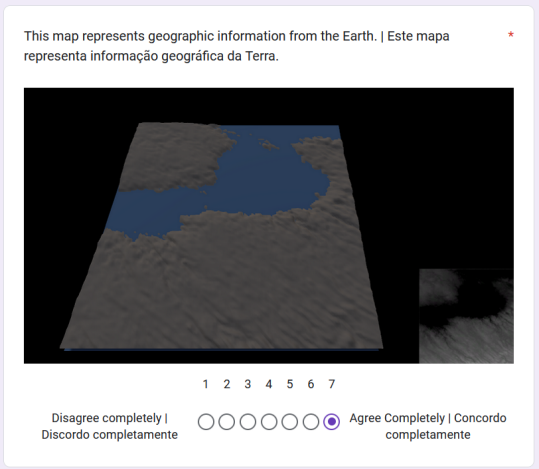

Figure 16: Example of a question from the user test. The remaining questions follow this format, changing only the image.

4.2.2 Result Analysis. We obtained a total of 79 participants in the study, which we consider an acceptable value. We started by analysing the Chronbach alpha of the evaluations of all ground-truth maps and all maps generated by our system, obtaining a value of 0.889 and 0.898, respectively. These values of Chronbach alpha fall within the category of "good" internal consistency, nearing the threshold of "excellent", for which a value of 0.9 is required. The consequence of this result is that we can aggregate both categories of maps, Earth's and ours, using their mean value, which is graphed in Figure 17.

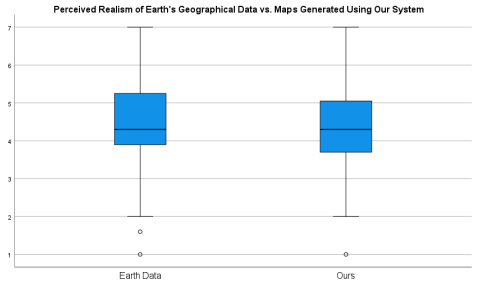

Figure 17: User perception of realism of maps generated with Earth's geographical data, vs. generated by our system.

We then performed the Shapiro-Wilk normality test on the means obtained, resulting in p-values of 0.032 and 0.013, for ground-truth and system generated means respectively, indicating that our data does not follow a normal distribution.

Because our data is non-parametric, we chose to evaluate the populations using the Wilcoxon's paired rank test, from which we determined that there was no statistical difference between the two populations, in other words, participants were unable to distinguish between ground-truth maps and maps generated using our system ($Z = -0.399, p = 0.69$).

4.3 Results of Architecture Variations

In this section we will discuss the results obtained from the three architecture variations explained in Section 3.7, both in terms of their visual quality, and how well they translate the content of their input. Please note that each of these variations was tested separately, resulting in a total of three different experiments, not counting the final results, which serves as a baseline.

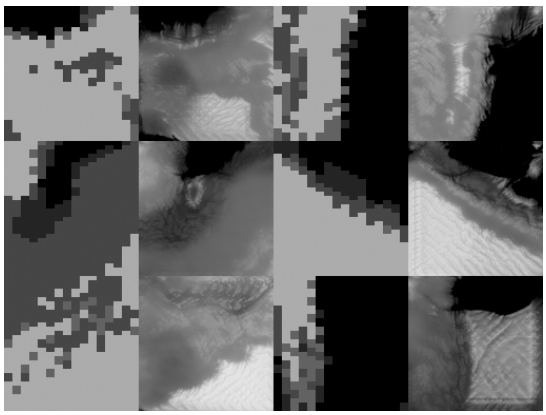

Figure 18: Results from bilinear interpolation experiment.

4.3.1 Bilinear Upscaling of IMR. After analysing the results of this experiment, from which Figure 18 is a subset, we concluded that the use of bilinear interpolation in the upsampling layers contributes negatively to the visual quality of the images generated. We believe that this decrease in quality is caused by an increase in the amount of values expressed by the IMR as a result of the smoother interpolation, which in turn may difficult learning.

Another observation of note is that this alteration creates patterns, which are especially evident in large, high altitude areas. One possible explanation for this fact is that this form of interpolation reduces the amount of times this type of areas are present, as sometimes they are interpolated to a lower altitude as a result of the neighbouring terrain, in turn, this may cause the model to learn the representation of this terrain from areas exhibiting a similar pattern.

4.3.2 Forced Conversion Critic training. In this experiment we tested a forced schedule that didn't allow either GAN to train for more than 10 epochs consecutively, which consequently means that each GAN is trained at least once every 10 epochs. We verified that the results declined in visual quality, which we believe to come from the fact that the Conversion GAN is not apt to maintain visual quality, and while the results may be more aligned to the content of the IMR, this is not particularly noticeable after a certain point,

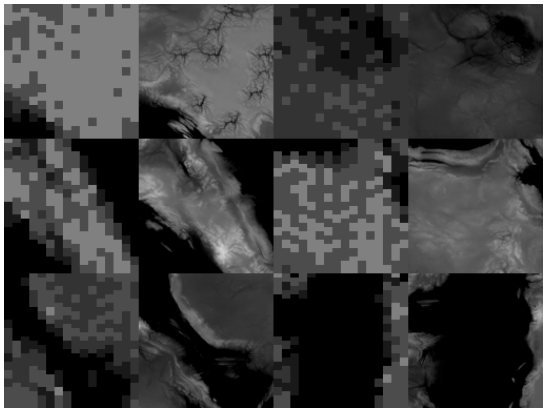

Figure 19: Results from forced conversion training experiment.

and serves only to limit the freedom of the Generator to create consistent and visually appealing images.

Figure 20: Results from the Epoch-based Scheduling experiment.

4.3.3 Epoch-based Scheduling. In this experiment we schedule which GAN to train at the start of each epoch, and train that GAN for the whole epoch. While this seems like the most obvious approach it has both upsides and downsides when compared to scheduling each batch. After conducting the experiment and observing the results acquired (Figure 20) we observe that often throughout training the content of the IMR and the resulting output stop matching, more specifically, the content matches with the opposite type of terrain: oceans become mountains, and vice-versa. We believe this is caused by an excessively high learning-rate of the Conversion GAN, which, over the course of an epoch flips the sign of some neurons, thereby causing the observed mismatch. For the final results we opted to schedule training on a per-batch basis, as opposed to lowering the learning rate, as we observed this former approach did not exhibit any of the problems theorised in Section 3.7.3.

5 CONCLUSION

Our goal for this thesis was to provide an alternative to current ways of generating height-maps for video-games. We needed a system that would create realistic and visually appealing results, while not requiring too much work or knowledge from the part of user, but at the same time allowing the user to specify exactly which geographical features should be present in the end result, and where. In order to accomplish this goal we designed and implemented a system that would take a very generic format, able to be converted to from a wide variety of existing tools, and generates a realistic height-map that contains the same features present in the supplied sketch. This generation is done using our proposed model, the Dual Critic Conditional Wasserstein GAN (DCCWGAN), a new type of conditional WGAN using two critics: The first Critic to evaluate the content of the input matches that of the generated map, while the second Critic guides the results to be more and more realistic. We then evaluated our system, determining through empirical observation that the content of the outputted maps closely match those of the supplied input. We also conducted user tests, from which we were then able to prove that users are unable to distinguish between maps we generated and maps created from geographical information of the Earth, which was the standard we set for realism. Overall, we consider that, while there is room for improvement, we achieved the goals we set out for, and contributed to existing knowledge by implementing a system that performs a form of image-to-image translation using multiple critics.

5.1 System Limitations and Future Work

While we consider the results achieved suitable, these are not without their limitations. The first great limitation comes from the resolution of the maps generated, which is merely a 128×128 image. While the system should theoretically work on larger images, this would require more memory, which in turn would increase the time required to train.

Another large limitation is the amount of control given to the user, who currently can only change the height the map. While some features such as vegetation would be simple to implement through a post-processing phase using PCG techniques, other features such as rivers should be included in the IMR, to be used by the networks themselves, this would, however, require a more complex algorithm to translate the ground-truth data-set into the IMR format as our approach is unable to detect rivers due to their relatively small effect on the elevation of that area.

REFERENCES

[1] Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein gan, 2017.

[2] Brock, A., Donahue, J., and Simonyan, K. Large scale GAN training for high fidelity natural image synthesis. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019* (2019), OpenReview.net.

[3] Feng, J., Feng, X., Chen, J., Cao, X., Zhang, X., Jiao, L., and Yu, T. Generative adversarial networks based on collaborative learning and attention mechanism for hyperspectral image classification. *Remote Sensing 12*, 7 (2020).

[4] Goodfellow, I., Bengio, Y., and Courville, A. *Deep learning*. MIT Press, 2017.

[5] Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2017), pp. 1125–1134.

[6] Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., and Aila, T. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2020).

[7] Millington, I. *Artificial intelligence for games*. CRC Press, 2019.

[8] Mirza, M., and Osindero, S. Conditional generative adversarial nets. *CoRR abs/1411.1784* (2014).

[9] Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral normalization for generative adversarial networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings* (2018), OpenReview.net.

[10] Nunes, V., Dias, J., and Santos, P. Gan-based content generation of maps for strategy games. In *GAME-ON 2022* (2022).

[11] Park, T., Liu, M.-Y., Wang, T.-C., and Zhu, J.-Y. Gaugan: Semantic image synthesis with spatially adaptive normalization. In *ACM SIGGRAPH 2019 Real-Time Live!* (New York, NY, USA, 2019), SIGGRAPH '19, Association for Computing Machinery.

[12] Richardson, E., Alaluf, Y., Patashnik, O., Nitzan, Y., Azar, Y., Shapiro, S., and Cohen-Or, D. Encoding in style: A stylegan encoder for image-to-image translation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2021), pp. 2287–2296.

[13] Wang, T.-C., Liu, M.-Y., Zhu, J.-Y., Tao, A., Kautz, J., and Catanzaro, B. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2018).