Deep Convolutional Generative Adversarial Network for Procedural 3D Landscape Generation Based on DEM

Andreas Wulff-Jensen, Niclas Nerup Rant, Tobias Nordvig Møller(✉), and Jonas Aksel Billeskov

Department of Architecture, Design and Media Technology, Aalborg University Copenhagen, AC Meyers Vænge 15, 2450 Copenhagen, SV, Denmark
awj@create.aau.dk, {nrantl4, tnmal4, jbillel4}@student.aau.dk

Abstract. This paper proposes a novel framework for improving procedural generation of 3D landscapes using machine learning. We utilized a Deep Convolutional Generative Adversarial Network (DC-GAN) to generate height-maps. The network was trained on a dataset consisting of Digital Elevation Maps (DEM) of the alps. During map generation, the batch size and learning rate were optimized for the most efficient and satisfying map production. The diversity of the final output was tested against Perlin noise using Mean Square Error [1] and Structure Similarity Index [2]. Perlin noise is especially interesting as it has been used to generate game maps in previous productions [3, 4]. The diversity test showed the generated maps had a significantly greater diversity than the Perlin noise maps. Afterwards the heightmaps was converted to 3D maps in Unity3D. The 3D maps' perceived realism and videogame usability was pilot tested, showing a promising future for DC-GAN generated 3D landscapes.

Keywords: GAN · Deep Convolutional Generative Adversarial Network
PCG · Procedural generated landscapes · Digital Elevation Maps (DEM)
Heightmaps · Games · 3D landscapes

1 Introduction

A major part of videogames is the worlds the player can explore. These worlds establish the foundation of the games and set the mood for the gameplay. Over the years many different worlds have been created for games; from the small worlds of Age of Empires 3 [5] to the billions of planets created in No Man's Sky [3].

Each world has been created to suit the needs of the gameplay and constructed with techniques, which allow for such worlds to be made. Whether the game is set in space, the forest or in a medieval village, different techniques may prove useful for different situations. Some techniques may require a lot of man-hours for modeling and hand-crafting each hill and lake while others may require time to develop an algorithm to generate the maps for them (for examples see [6–9]).

With respect to the latter, this study aims to expand on the algorithms used by proposing the usage of a deep machine learning algorithm called Deep Convolutional Generative Adversarial Network (DC-GAN) [10] as the main framework for the map generation. An advantage of this framework compared to other procedural map generation algorithms is the wide and general applicability not commonly found in others [3, 7], as it can be trained to generate multiple different kinds of outputs with distinct features such as tundras, ravines and mountains not limited to one category. Potentially this could become a tool for game developers to generate landscapes faster and more customizable than unassisted procedural content generation. This method should be considered a combination between an unassisted and assisted method [11], where the developer can influence the output while still maintaining diversity and randomness, as it creates multiple landscape entities at ones, not obtainable by other experience-agnostic procedurally generated landscape solutions [12]. Furthermore, it goes beyond the popular idea of using Perlin noise [13, 14] or Digital Elevation Maps (DEMs) [15] as the basis for creating game maps (see for example Minecraft [4], No Man's Sky [3] and Cities: Skyline [16]), due to Perlin noise's popularity the diversity of the generated maps lacks variety, thus increasing the chance of a dull gaming experience. With respect to the DEM for some simulation games like Cities: Skyline it can be engrossing to work with a familiar environment, while in other games such as first person or third person games it can counteract the experience as the environments are not deviating as much.

2 DC-GAN and Its Previous Applications

In 2014 Ian Goodfellow et al. introduced the Generative Adversarial Networks (GANs). They were used to generate images that look photographically authentic compared to the training image dataset [17]. GANs are based on adversarial training, where two players learn from each other by pursuing conflicting goals. GAN consist of two neural networks, one called the generator and another called the discriminator. The discriminator's goal is to distinguish between the training data and the generated data, which is learned through traditional supervised learning. The goal for the generator's is to generate images, which are similar to real images such that the discriminator network cannot tell them apart from the real data. Thus, enabling the generator to learn through reinforcement learning. This method makes the two networks compete against each other thus constantly improving the other's abilities. The overall goal of the GAN is to generate images that look like the images from the training dataset to a point where the discriminator is not able to tell whether the image is real or fake. In theory, this process of continuously producing more realistic images will continue until the goal is reached. This competition is also called a min-max game or zero-sum game [18], where each adversary is trying to minimize its own loss in worst case scenario.

With respect to the functions and architectures of the networks. The discriminator is a four-layer deep convolutional network with ReLu activation functions. It is represented by a differentiable function D and training data x. x is convoluted from start input of 64X64X3 through the network to 4X4X512 feature maps, which is fully connected to a single output neuron [10]. This neuron determines if the input is a generated image D(x) = 0 or an image from the dataset D(x) = 1. Therefore, will the discriminator try to do the following for the input data: maximize D(x) for every image from the training data distribution, and minimize D(x) for every image generated by the generator.

The generator, on the other hand, is a deconvolution network embedded with ReLu activation functions. It starts with a random vector z of 100 values. This vector is sent through a four-layered deep deconvolutional network ending with a 64X64X3 image. The generator is represented by a differentiable function G. G(x) has the reverse goals of D(x), meaning it tries to minimize D(x) for every image from the training data distribution, and maximize D(x) for every image generated by the generator [18].

The presented GAN network has been trained on a multitude of generation problems with different degrees of success. Lotter et al. [19], used GAN to predict the next frame in a simple head rotation animation. However, multiple models were used to create sharp output images, as a single GAN tended to blur eyes and ears. Ledig et al. [20] later demonstrates that a modified GAN can recover a low-resolution image to its high-resolution stage, due to its ability to generate multiple right answers compared to other tested algorithms. Zhu et al. [21] created an interactive GAN application in which the user can draw. Afterwards the machine converts the drawing to a detailed realistic photo. A similar application was created by Brock et al. [22]. In this application GAN was used to interpret rough sketch modifications edited unto a realistic image, and convert it to the realistic counterpart. The interpretation and conversion between sketch and realism have been used by Isola et al. [23] as well. They have trained GAN to convert satellite images to sketches and sketches of, for example bags, to realistic looking bags. Lastly, Zhang et al. and Reed et al.

[22–24] have achieved to create multiple images based on text description. However, the usual GAN did not create as diverse output as wished, but a modified version called StackGAN corrected this problem.

The investigated studies all showcases perfect examples of how a well-trained and adjusted GAN can work. Some of the problems being harder than others, for instance the interactive GANs seems to be harder, as they may require a big database within their core to be able to guess rather accurately what the user tries to draw. While on the opposite side of the spectrum there is the GAN, which presumably only has been trained with a limited number of images to be able to create for instance a high-resolution image based on the low-resolution input.

Furthermore, a challenging problem faced by [18] is the task of creating animals based on classes from ImageNet [27]. Through this training the GAN created distorted abominations with wrong amounts of different limbs and even concatenations of different creatures. Thus, showcasing one of the limits of GAN.

3 Data Collection for Map Generation

Through the review of how the GAN works and with what it has been utilized, it is certain that to create diverse, but still realistic game maps we need a dataset with the same, yet diversely utilized set of features. Such dataset can for instance be found in heightmaps like DEMs, which is contained within a database called viewfinder-panoramas [15]. In this database the Alps and Norway was chosen, as they have different slopes, attitudes and mountain types. The Norway dataset set was later discarded, because it contains coastlines, which initiated oddly placed lakes in the generated maps, thus reducing the realism of the generated maps. The database from the Alps was sampled into 360,000 images using a program called Auto Hotkey, which was employed to automatically take screenshots of different areas of the Alps. Every 3 s an image of a random location of the Alps was taken, then sliced into 9 smaller pieces to fit the 64X64 entry space of the Discriminator. The process lasted 30 h (an example of the DEM screenshots can be found in Fig. 1).

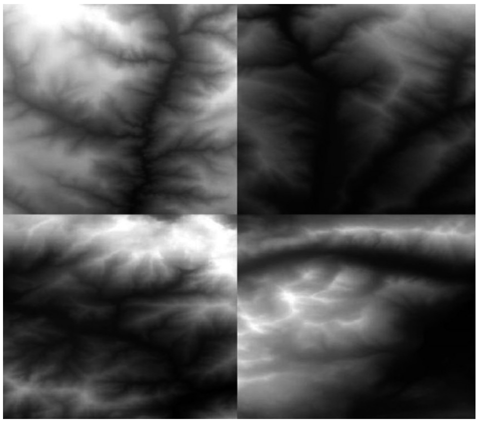

Fig. 1. Example of images scraped from DEMs

4 Evaluations

To evaluate and optimize the system three different methodologies have been employed. The focus of the first is to optimize the network, while the second and third concentrates on objective and subjective parameters of the generated maps.

4.1 Optimizing the Network

The network has different hyper parameters which can be adjusted to make the network perform better or worse with different advantages and disadvantages. The two parameters we will look at are batch size (BS) and learning rate (LR). BS being how many data samples the network is fed with before updating the weights inside the system. LR is how big of a step the gradient decent moves at each learning epoch. Too big BS can result in having too little data to train with, thus the network will never learn correctly. While too little BS could lead to overfitting. The LR can be adjusted to make the system learn faster with a big LR, but at the cost of missing the global minimum. With a small LR, the network will take long time to learn and to even reach a minimum, which unfortunately is likely to be the local minimum.

The evaluation will evaluate three different BSs (16, 64 and 256) and Three different LRs (0.002, 0.0002 and 0.00002). The most optimized network will be the one reaching a satisfying result with the fewest number of epochs, showcasing an output least influenced by repetitive noise patterns.

4.2 Diversity Evaluation

For the objective measuring method, diversity tests between the generated maps and Perlin Noise will be conducted. The diversity is accessible through two image processing algorithms, Mean Square Error (MSE) and Structured Similarity Index (SSIM). MSE has previously been used to estimate the similarity of the pixel intensity between an original image and a distorted image [1]. Therefore, it can be valid to use to measure the difference, thus diversity between two images within a dataset. SSIM goes beyond intensity and investigates the difference in luminance, contrast and structure, by running a kernel through the images we want to compare. Thus, comparing smaller subsets at a time instead of the whole image [2]. The SSIM will allow for a closer inspection of the diversity.

From both tests the diversity within the images datasets will be established. The greater the MSE value the more difference does exist between the images in the sample sets. The smaller SSIM value the greater are the differences between the images.

4.3 Preliminary Usability Test

While the other evaluations focus on how to improve the system and to access the diversity of generated images, a small usability test is needed to test for the aesthetics, traversability and realism of the game maps. Seeing the maps are still heightmaps, thereby not usable to explore in game settings. We will use Unity3D to convert the heightmaps to 3D landscapes with preliminary texturing using Unity's built-in terrain tool. For the test, we will recruit a small sample of participants. Each will spend five minutes walking around three of the generated maps. Afterwards, they will be asked eight questions:

1. Does the map look realistic to you?
2. How does this landscape compare to other video games you have played?
3. How was the landscapes' traversability?
4. How did you feel about the diversity of the landscapes?
5. Does the map look like something you would see in the real world?
6. What kind of game do you see these maps being used in?
7. What landscape from the real world does the maps remind you off?
8. Do you have any general comments or suggestions for further implementations?

With question 1, 4, 5, and 7 we wish to evaluate the realism of the generated landscapes, as they are derived from a dataset based on the real world.

Question 2, 6, and 8 examines the usability and in which situation the user could see these maps be used in, and at the same time understand if they have any comments or suggestions for improvements.

At last question 3 is about how well the user could traverse in the landscape without getting frustrated or felt like they exploited the physics.

5 Results

5.1 Optimizing the Network

For all the batch sizes, a progressive improvement of the output based on the steadily increasing number of epochs can be seen (see Fig. 2).

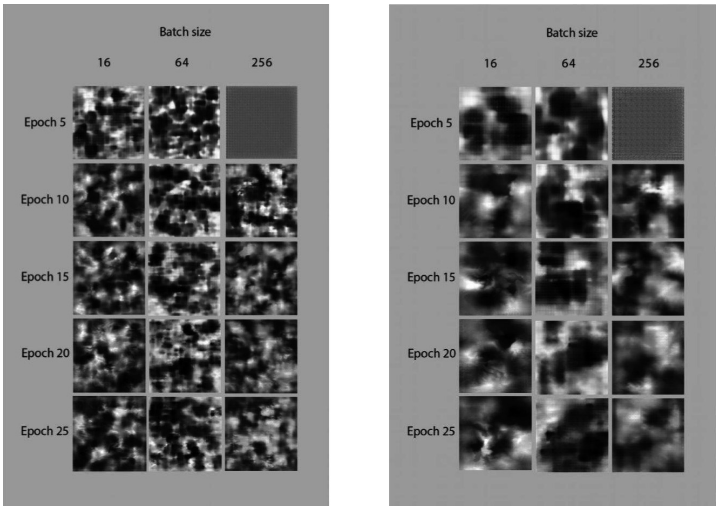

Fig. 2. Output from 3 different batch sizes, through 5 different epochs and 2 different sized images from the Alps dataset

The batch size 16 learned quicker, as the noise is non-visible already in the early epochs. The images look sharper compared to those with batch size of respectively 64 and 256 as they look blurred. Furthermore, the noise is still a bit visible even in the last epoch for batch size 64 and 256. This observation could indicate that the networks need more epochs or more training data to train properly, as some of the noise from the z vector is still persistence. From the 10th epoch the 16-batch size network seems to generate noise free inputs, which compared to the other network witnesses that increasing the batch size seems to only worsen the output. Additionally, at the 5th epoch the batch size of 256 showed no learning at all, as it only shows noise, while the other two have already learned the initial steps towards creating a heightmap. Given the current dataset this evaluation indicates, that a batch size of 16 gives the best possible output.

To further optimize the output the next step is to change the learning rate to see if the previous results can be improved. The results for changing the learning rate, from the default value of 0.0002 to respectively 0.002 and 0.00002, showed no improvements. The results got significantly worse when it was set to 0.002, where the network did not learn at all and continued to generate noise. When the learning rate was set to 0.00002 the network learned, but the final output was worse than the original. Thereby, it was chosen to work with the following hyper parameters: Learning rate: 0.0002. Batch size: 16.

5.2 Diversity Evaluation

The diversity tests' results found in Table 1 are derived from 10000 comparisons between either 500 generated images or 500 Perlin noise images, in which it was ensured that the same two images were not compared twice.

Table 1. Shows the descriptive results of both diversity tests. PN = Perlin Noise, GAN = Generated heightmaps from the GAN network.

	PN - SSIM	GAN - SSIM	PN - MSE	GAN - MSE
Mean	0,7247	0,3109	2829,2957	6021,6372
Median	0,7260	0,3106	2739,2960	6002,5174
SD	0,0241	0,00306	788,5848	706,6115
SE	0,0007617	0,00066779	7,8858	7,0661

Descriptively it is evident that there is a higher degree of diversity within the generated image dataset compared to the Perlin noise dataset. MSE mean GAN > PN and SSIM mean GAN < PN. Statistically this observation is well supported through the Wilcoxon Ranked Sum test which showed p = 0 thus $p < 0.05$, and a H-value = 1. A parametric test could not be utilized as an Anderson Darling test showed that neither of the result datasets were normally distributed.

5.3 Preliminary Usability Test

For this test three students, 1 female and 2 males, were recruited. The feedback from those were positive in terms of realism. Generally, the participants could relate the landscape to real world settings, and thought in general that the landscapes looked like something from the real world (an example of the landscape can be found in Fig. 3). The diversity in the heightmaps, was noticed, but not apparent. Another positive thing was the traversability of the landscape, as all the participants found it easy to traverse.

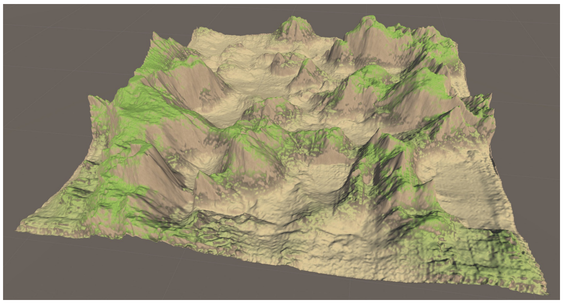

Fig. 3. One of the final landscapes implemented in Unity, which was shown to the participants in the third test.

The textures in the landscape were roughly implemented, and even though the participants were told to focus on the height of the landscape, the participants had difficulties with excluding the textures in their judgement - both in terms of realism and diversity. This indicates that both parameters cannot be fully satisfied without proper texturing as it goes hand in hand with the landscapes.

6 Discussion

The results from the three tests showed both the possibility to create diverse heightmaps and make them usable within the context of games. However, the diversity measures could be questionable, as they are based on pixel value comparisons. This could unintentionally have inflated the test results to show more diversity, than what is real. Another measure to go beyond the per pixel or group of pixels comparison could be through considering the frequency domain and compare the frequency distribution between the images. This will in turn showcase where on the spectrum, and thus which level of detail there are mostly diverse within the group of images. Nevertheless, it is subject for another study. For this paper and the chosen algorithms, it can be justified that on the pixel level the datasets are significantly different from each other.

With respect to the third test the diversity between the landscapes and the texturing were questionable. Even though, there was only 3 subjects in the test these factors were clear drawbacks. The observation nicely illustrates, that isolating the heightmaps of a terrain from textures and other aesthetic elements is hard for the user, thus showing that all of it is intertwined. By further developing the textures of the terrain and add a procedural content generator to create trees, grass and rocks would highly enhance the aesthetic appreciation of the landscapes, and thus heightening their perceived usability. The texture of a future development could be developed by the GAN as well. Seeing that the heightmaps are based on real world photos, the same locations can be found on Google Earth. By training the GAN with the Google Earth images, the output images can be used as basis for the textures. This improvement will make the texture creation more automatic and possibly with greater diversity compared to the textures used by the terrain editor in Unity. Furthermore, a test with just 3 participants are proven to be biased, as it can rarely be replicated, but such small preliminary test can give valuable guidelines without giving the full solid picture. A last remark we want to discuss is the network it-self and its hyperparameters. Through the test we saw that a batch size of 16 was most efficient in producing the best results. However, it is still questionable if a lower batch size with our dataset could have improved the network even further, or if the batch size of 16 is the most optimal. Another thing is the learning rate, which performed best at 0.0002. With respect to this and the conducted tests a more efficient network could possibly be found if the learning rate was only slightly adjusted either up or down compared to the big steps to 0.002 or 0.00002.

7 Conclusion

The focus of this study was to investigate the usability of a deep convolutional generative adversarial network to generate 3D landscapes for video games, which could challenge conventional methods of game map generation. We attempted solving this problem by proposing a novel pipeline, using DEMs gathered from the Alps as a dataset for a Torch implementation of a DC-GAN. Afterwards the generated heightmaps were used in Unity to create 3D landscapes. Through the study three different evaluation of the product was conducted. One testing the optimization of the networks, one testing the diversity of the output, and one qualitative test for general feedback and realism of the landscapes. The first test showed good results when lowering the batch size to 16 with a learning rate of 0.0002. The second test showed greater diversity within the generated heightmaps compared to Perlin noise maps. The third test gave mixed results as the test participants had a hard time excluding the textures from their perception of the landscapes, but overall thought the landscapes were easy to traverse and looked like landscapes from areas in the real world. Combined, all the tests showed an ability to create functional landscapes based on real world areas. We believe that our attempt to use machine learning to generate maps for video games was overall successful, and can in fact challenge the use of Perlin noise for map generation. Furthermore, there is basis for continued research and experimentation in this area of machine learning and procedural generation.

Exploring the use of DCGAN's applications with procedurally generated landscapes could lead to more interesting possibilities including but not limited to combining landscapes, generating buildings and fauna, adding lakes or oceans and generating textures, which in the future could be done interactively e.g. [21–23] and be implemented to create game maps on the fly or endless terrains wherever a user moves.

References

1. Wang, Z., Bovik, A.C.: Mean squared error: love it or leave it? IEEE Sig. Process. Mag. **26**, 98–117 (2009). https://doi.org/10.1109/MSP.2008.930649
2. Wang, Z., Bovik, A.C., Sheikh, H.R., Simoncelli, E.P.: Image quality assessment: from error visibility to structural similarity. IEEE Trans. Image Process. **13**, 600–612 (2004). https://doi.org/10.1109/TIP.2003.819861
3. Cook, M.: We've Run Out of Planets - Procedural Generation After No Man's Sky. Cut Garnet Games (2016)
4. Mojang, A.B.: Minecraft. Stockholm, Sweden (2013)
5. Microsoft: Age Of Empires III. Microsoft, Redmond (2013)
6. Yannakakis, G.N., Togelius, J.: Experience-driven procedural content generation. IEEE Trans. Affect. Comput. **2**, 147–161 (2011)
7. Togelius, J., Preuss, M.: Towards multiobjective procedural map generation
8. Shaker, N., Asteriadis, S., Yannakakis, G.N., Karpouzis, K.: Fusing visual and behavioral cues for modeling user experience in games. IEEE Trans. Cybern. **43**, 1519–1531 (2013). https://doi.org/10.1109/TCYB.2013.2271738

9. Shaker, N., Togelius, J., Nelson, M.J.: Procedural Content Generation in Games: A Textbook and an Overview of Current Research. Springer, Berlin (2014). https://doi.org/10.1007/978-3-319-42716-4
10. Brandon, A.: Image completion with deep learning in TensorFlow. GitHub (2016)
11. De Carli, D.M., Bevilacqua, F., Pozzer, C.T., Cordeiro D'Ornellas, M.: A survey of procedural content generation techniques suitable to game development. In: Brazilian Symposium on Games and Digital Entertainment, SBGAMES, pp. 26–35 (2011). https://doi.org/10.1109/sbgames.2011.15
12. Yannakakis, G.N., Togeliu, J.: Artificial Intelligence and Games (First Public Draft) (2017)
13. Perlin, K.: Improving noise. In: Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2002, p. 681 (2002). https://doi.org/10.1145/566570.566636
14. Rose, T.J., Bakaoukas, A.G.: Algorithms and approaches for procedural terrain generation. In: 2016 8th International Conference on Games and Virtual Worlds for Serious Applications, VS-Games 2016, pp. 4–5 (2016). https://doi.org/10.1109/vs-games.2016.7590336
15. De Ferranti, J.: Viewfinder Panoramas (2012)
16. Paradox Interactive: Cities: Skyline. Paradox Interactive. Stockholm, Sweden (2015)
17. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative Adversarial Networks, pp. 1–9. (2014). https://doi.org/10.1001/jamainternmed.2016.8245
18. Goodfellow, I.: NIPS 2016 Tutorial: Generative Adversarial Networks (2016). https://doi.org/10.1001/jamainternmed.2016.8245
19. Lotter, W., Kreiman, G., Cox, D.: Unsupervised learning of visual structure using predictive generative networks, pp. 1–12 (2016). https://doi.org/10.1109/iccv.2015.465
20. Ledig, C., Theis, L., Huszar, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., et al.: Photo-realistic single image super-resolution using a generative adversarial network (2016)
21. Zhu, J.-Y., Krähenbühl, P., Shechtman, E., Efros, A.A.: Generative visual manipulation on the natural image manifold. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9909, pp. 597–613. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46454-1_36
22. Brock, A., Lim, T., Ritchie, J.M., Weston, N.: Neural photo editing with introspective adversarial networks (2016). https://doi.org/10.1177/1470320311410924
23. Isola, P., Zhu, J.-Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks (2016)
24. Zhang, H., Xu, T., Li, H., Zhang, S., Huang, X., Wang, X., Metaxas, D.: StackGAN: text to photo-realistic image synthesis with stacked generative adversarial networks (2016)
25. Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B., Lee, H.: Generative adversarial text to image synthesis (2016)
26. Reed, S., van den Oord, A., Kalchbrenner, N., Bapst, V., Botvinick, M., deFreitas, N.: Generating interpretable images with controllable structure. In: ICLR 2017, pp. 1–12 (2016)
27. Krizhevsky, A.: Learning Multiple Layers of Features from Tiny Images. Technical report, Science Department, University of Toronto, pp. 1–60 (2009). 10.1.1.222.9220