Programming Project
Pattern Recognition Lab
Summer Semester 2021

TERRAIN GENERATION WITH GENERATIVE ADVERSARIAL NETWORKS

Submitted By:

Name: Mahadev Prasad Panda
Matriculation Number: 22753794

Contents

List of Figures

1 Introduction

To develop realistic and diverse terrain that usually acts as the basis of various computer worlds, i.e., computer graphics, virtual environment development, a large amount of time and effort is spent. A high level of realism is required for better immersion of users in many games. The current tools of development for terrains like Blender, Cinema 4D, Unity, or World Machine are highly dependent on the designers' skills and experience for realistic creations [1].

Terrains in virtual environments are created by displacing points of a flat grid with help of a heightmap which contains data about the depth of the grid points. Open world games like Skyrim, Fallout use heightmap images to compress the map data of the game and render detailed landscapes in application with some interpolation algorithms that smooth out the differences between high points. Currently, the heightmaps are being created either manually by developers which is very time-consuming, or using algorithm-based methods such as mapping perlin noise [2] or fractal noise to a grid at specific points [3].

Though using perlin noise versatile height maps can be generated as it can create a smooth transition between height values, it sometimes can lead to unrealistic height maps. In Fractal Brownian Motion very detailed images can be created by controlling the degree of intensity of different noise layers, sampling, and averaging points from perlin noise over multiple levels of varying height but the height maps generated using this method have less noticeable features [4]. The dependency on noise can be avoided by using the diamond square algorithm. In this method, highly detailed terrain height maps can be created by continuously averaging between preselected points on a grid. But it is very difficult to control the resulting terrain shapes due to the randomness in the preselected points [5]. Though the algorithm-based methods can create realistic terrains they can be repetitive.

This project focuses on a fully automated process for creating height maps with the help of deep learning methods. For this purpose, a Generative Adversarial Network (GAN) is designed and implemented. The schematic diagram of a GAN can be seen in the Figure 1. GANs provide a unique method as they can learn the underlying data distribution of a data set and can map it to random noise vectors. A GAN contains two models: a generator G that maps the data distribution and a discriminator D that tries to predict the probability of a sample data belonging to the training data rather than coming from the generator G. During training discriminator D maximizes the probability of assigning the correct label to both training examples and samples from G and the generator G minimizes log(1 - D(G(z)) [6]. The min-max loss function of a GAN can be described by the following equations:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}\left[log(D(x))\right] + \mathbb{E}_{z\sim p_z(z)}\left[log(1-D(G(z)))\right] \qquad (1)$$

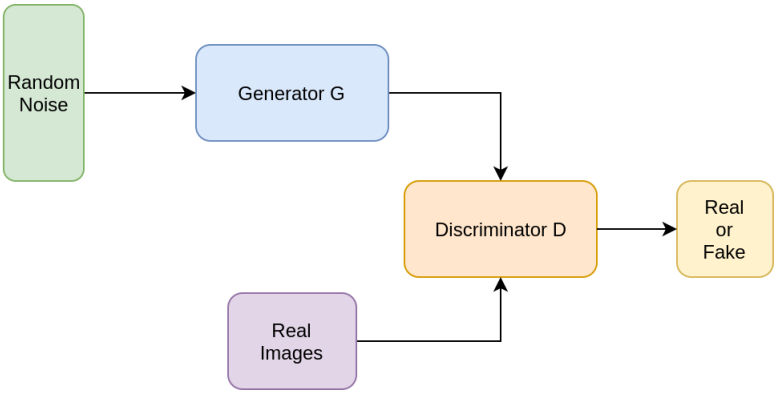

Figure 1: Schematic diagram of a Generative Adversarial Network (GAN)

2 Related Work

With an increase in the computational ability of computers, deep learning based approaches are gaining a lot of popularity in the different application fields. Beckham and Pal (2017) built a two-stage pipeline to randomly generate heightmaps as well as a texture map that is inferred from the heightmap. For the generation of heightmaps, a Deep Convolutional Generative Adversarial Network (DCGAN) and for texture synthesis, a pix2pix GAN is trained on openly available high-resolution terrain and heightmap data provided by NASA separately [7]. The generated height maps from the DCGAN sometimes exhibited small-scaled artifacts. The authors proposed to mitigate this problem using a deeper network and/or skip connections.

[8] employed Conditional GAN-based terrain synthesizers for authorization of virtual terrains. Each terrain synthesizer is trained for a specific task like generation of terrain evolution from erosion (erosion) synthesizer. Non-homogeneous texture synthesis using a Tile GAN was done in [9] where the authors created large-scale output from many input smaller resolution images by combining the outputs of GANs trained on smaller resolution images.

The term spatial GAN (SGAN) was originally introduced in [10] where the authors trained a new GAN model for Texture Synthesis. Instead of using a one-dimensional noise vector, the authors used a three-dimensional noise tensor as an input for the generator of the network which resulted in a well-suited architecture for texture synthesis. Spick et al extended the spatial GAN application to heightmap generation [1]. A variation of spatial GAN was trained on real-world digital elevation data taken from specific regions of the earth provided by NASA‘s SRTM. [11] further extended the work by training SGANS on height maps and texture maps stacked into single images and thus generating both textures and height maps from the same network.

In this project, a variant of the SGAN is used for generalized automatic generation of height maps. Unlike in DCGAN, the spatial dimension in the SGAN generator is not hardcoded. So, the same pretrained generator can be used to generate images of various spatial dimensions, dropping the need to train the generator again and again. This gives an architectural advantage to SGAN over DCGAN in the current application case.

3 Methodology

3.1 Data

The data required for training and testing was collected from free to access map data provided by Mapbox. For height map generation elevation data was collected. The elevation data from Mapbox is obtained in the form of raster images which encode the height values in the Red (R), Green (G), and Blue (B) channels in PNG format. Elevation data was taken at a zoom level of 11. 250,000 images were collected of dimensions 64x64x3. We divided the dataset into training dataset (150,000 samples)and testing dataset (100,000 samples). Figure 2 shows examples of training samples used during training.

The encoding of the actual height values as a base-256 number caused problems while using RGB data directly as input for GAN during training. Furthermore, we found that for all images the R channel in our dataset is 1. So we decide to consider only G and B channels while training which made the height values to be equally distributed in the value range. This reduced the number of different heightmap values from 16777216 to 65536.

The training samples are normalised using following equation,

$$h_n = \frac{G \cdot 256 + B - 32767.5}{32767.5} \tag{2}$$

where h_n represents normalized training height.

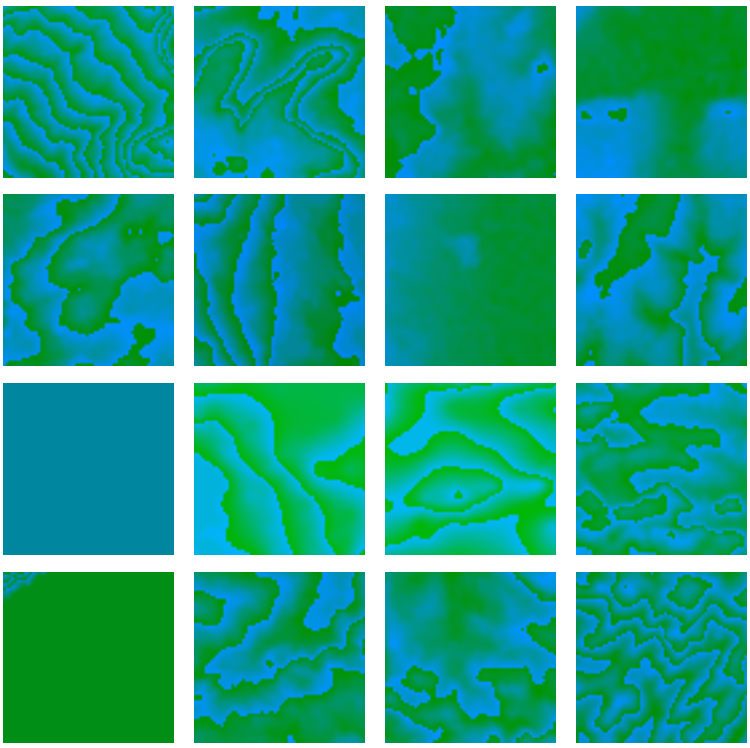

Figure 2: Sixteen training samples containing the height maps.

3.2 Model Architecture

For our task, we implemented a variant of spatial GAN. A spatial GAN learns translationally invariant features as all the fully connected layers are replaced with convolutional layers. We end up with a fully convolutional generator and discriminator. The generator learns to map a three-dimensional tensor onto an image space rather than using a one-dimensional vector. Figure 3 shows the schematic diagram of a spatial GAN architecture.

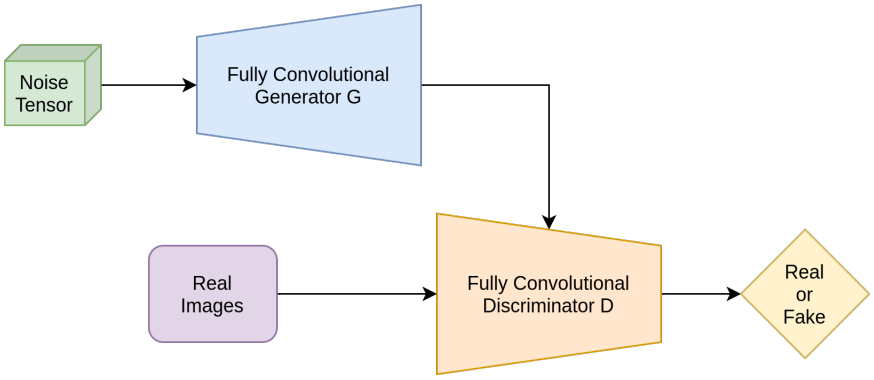

Figure 3: Schematic diagram of a spatial GAN.

For our task, we used a generator and discriminator architecture as shown in Figure 4 and Figure 5. The input tensor (noise seed) of the generator is sampled from a normal distribution with zero mean and standard deviation one. We started with a filter size of (3x3) for all the convolutional layers. In the generator, batch normalization is used after each transpose convolution layer. In generator and discriminator, LeakyReLu with a negative slope coefficient of 0.25 is used as an activation function for the intermediate convolutional layers. In addition to this, Dropout layer with a rate factor of 0.3 is used in the discriminator. The tanh non-linearity is used in the output layer of the generator whereas sigmoid non-linearity is used in the output layer of the discriminator. We started training with Adam with default parameter configuration as our optimizer with a learning rate of 0.00001 and a batch size of 128. We also applied one-sided label smoothing assigning the real images a value of 0.9 rather than 1.

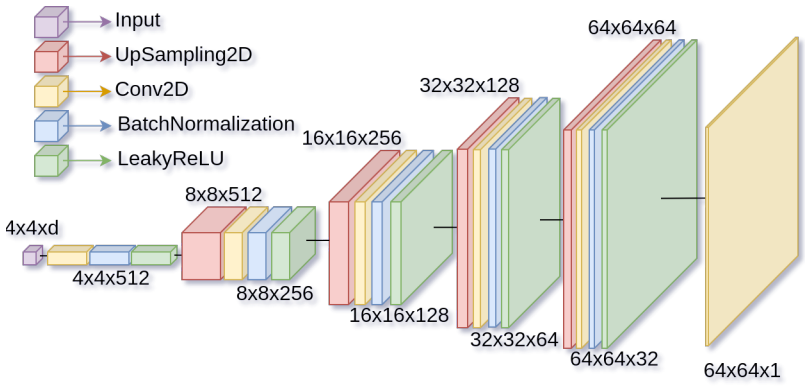

Figure 4: Schematic diagram of the generator architecture.

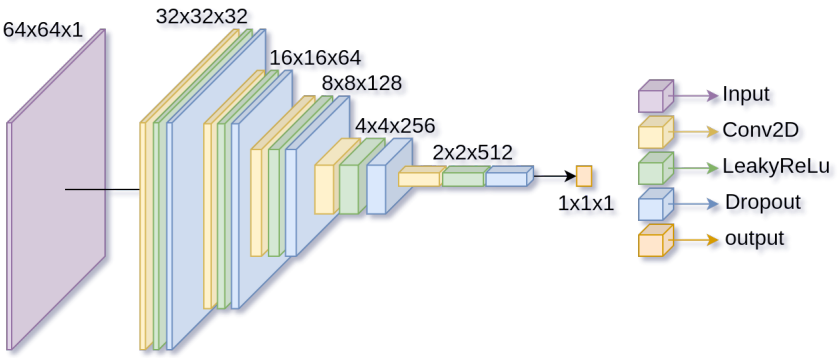

Figure 5: Schematic diagram of the discriminator architecture.

4 Experiment and Result

We ran a few training runs with the base parameter configuration as described in the previous section. The first problem we ran into was checkered patterns in the generator output. To solve this problem we replaced transpose convolution layers in the generator with a two-dimensional upsampling layer followed by a convolution layer. We trained the model with manual inspection and we noticed the network was learning trivial outputs like single colour height maps which can be obserbed from Figure 6.

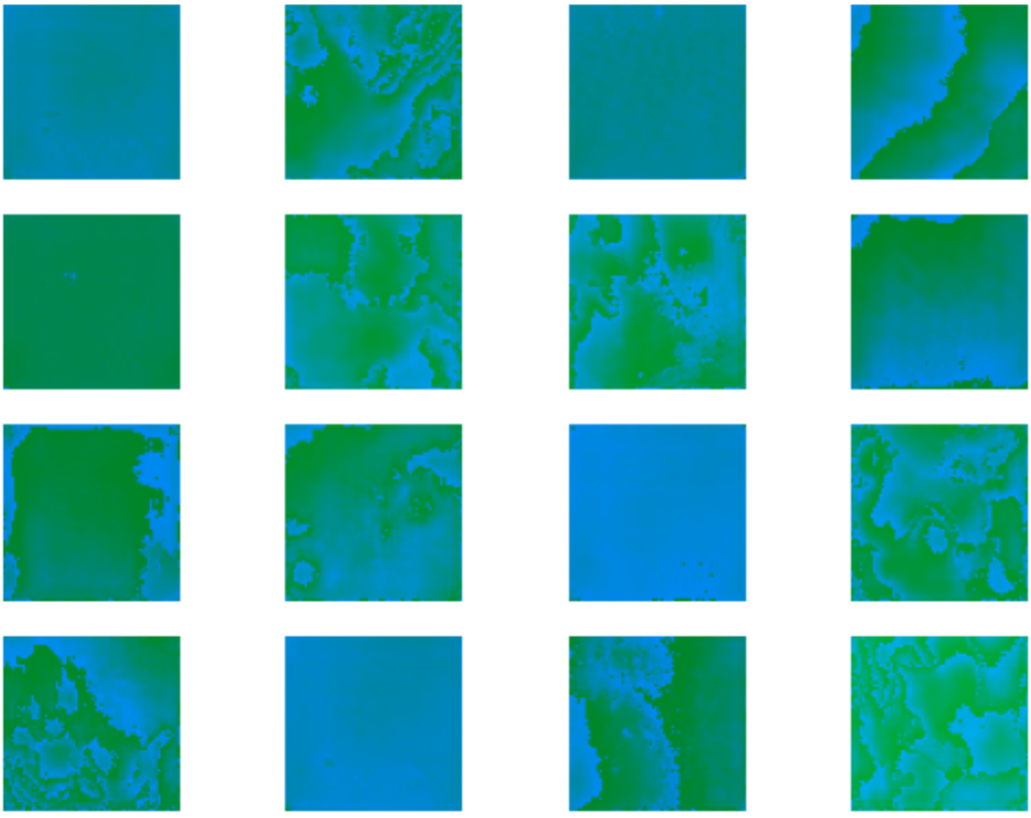

Figure 6: Genrated image from the SGAN generator before deletion of trivial images from the training samples.

After reviewing the training data set, we found a lot of occurrences of these trivial samples. We decided to remove these trivial input data from the training sample. Figure 7 shows the categories of data samples that were removed from the dataset.

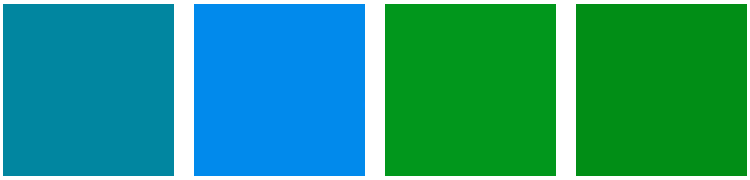

Figure 7: Four single colour delete category samples.

With several training runs after the trivial elements were deleted, we experimented with model depth, learning rate, kernel dimension, batch size, and choice of optimizers. For our task model depth, 5 (512-256-128-64-32 filters in convolutional layers) seems to work better than the model depth of 4 (512-256-128-64 filters in convolutional layers). We tried several learning rates and also applied the Two Time Scale Update Rule [12]. We used a learning rate of 0.0001 for the generator optimizer and a learning rate of 0.0004 for the discriminator optimizer. We started with a convolutional filter size of (3x3) but we switched to a convolutional filter size of (7x7) for the generator and a convolutional filter size of (3x3) which helped in creating less noisy outputs in the generator. Reducing the batch size of 128 to 64 gave us better results. Finally, we applied all the above changes and ran training for three optimizers, i.e, Adam, AdamW, and RMSprop. For our task Adam optimizer converged faster and showed better results. The learning curve in Figure 8 shows the behaviour of the generator loss and discriminator loss during training.

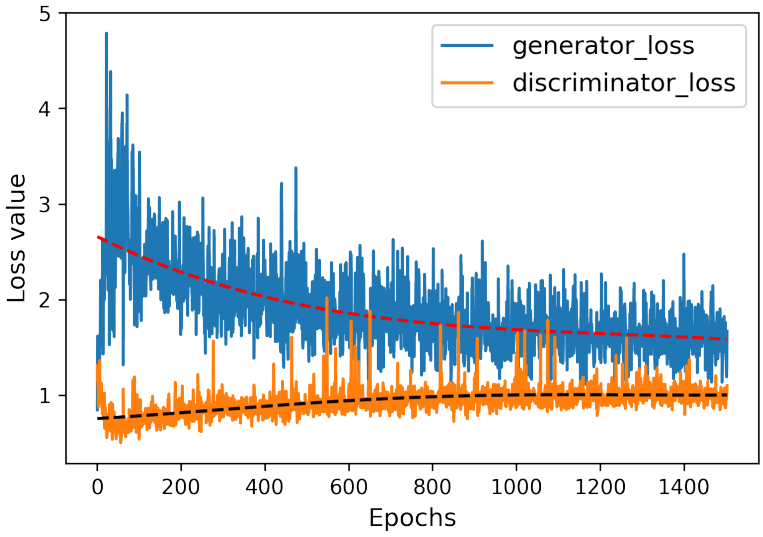

Figure 8: Training curve showing the variation of generator loss and discriminator loss with respect to epoch during training for Adam optimizer with default parameters with generator learning rate 0.0001 and the discriminator learning rate 0.0004.

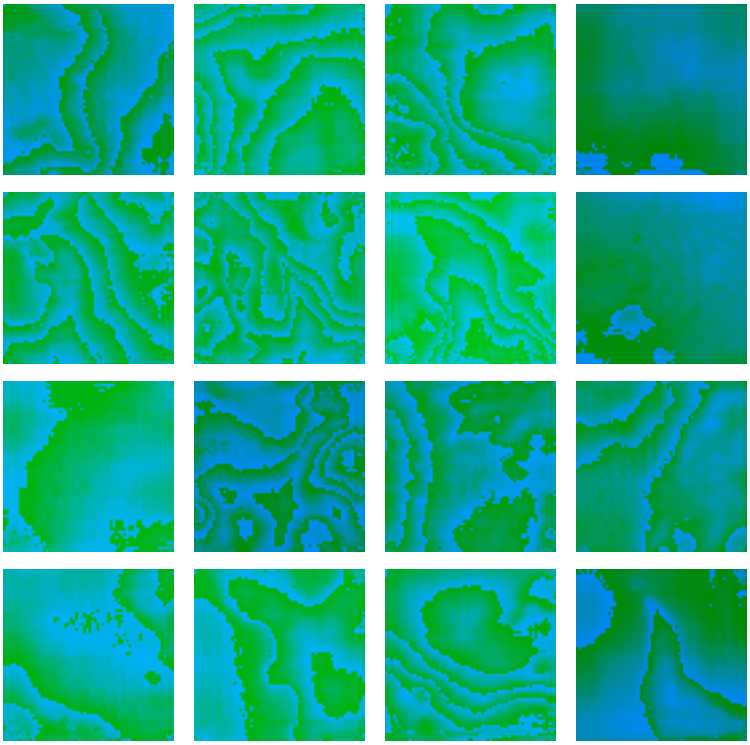

Figure 9: Generated output of size (64x64) from SGAN generator after 1490 epochs with Adam optimizer (with default parameters, generator learning rate = 0.0001, discriminator learning rate of = 0.0004, batch size of = 64), training sample size of (64x64).

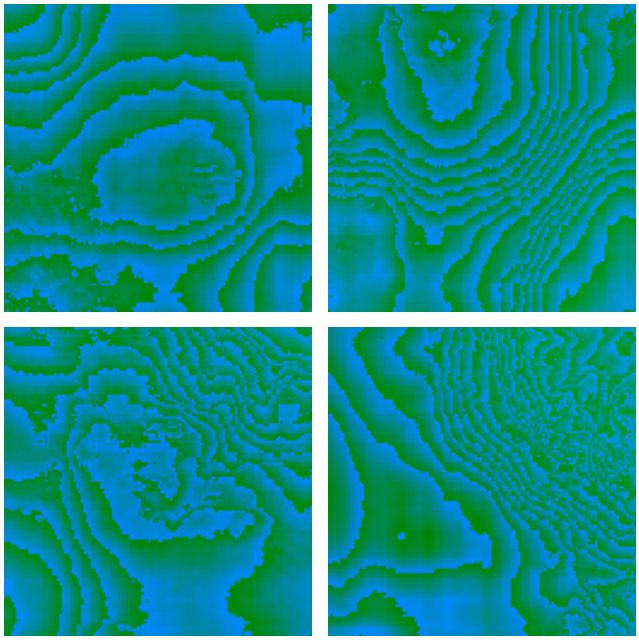

Figure 10: Generated output of size (128x128) from SGAN generator after 1490 epochs with Adam optimizer (with default parameters, generator learning rate = 0.0001, discriminator learning rate of = 0.0004, batch size of = 64), training sample size of (64x64).

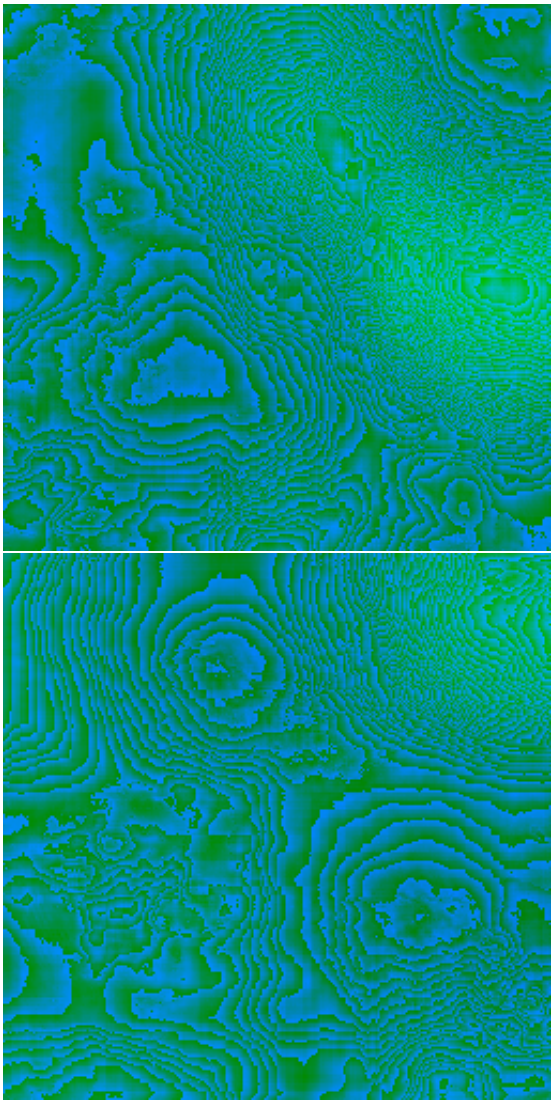

Figure 11: Generated output of size (256x256) from SGAN generator after 1490 epochs with Adam optimizer (with default parameters, generator learning rate = 0.0001, discriminator learning rate of = 0.0004, batch size of = 64), training sample size of (64x64).

The images in Figure 9, Figure 10, and Figure 11 shows the generated height maps from a sGAN generator trained on samples with spatial dimension of 64x64. We evaluated the model by calculating FID score [12]. The FID score was calculated between 90000 generated samples and 90000 testing samples using the official implementation provided in github. After running the evaluation we got an FID score of 92.83.

5 Conclusion

Though the results from sGAN model are promising, several factors can be improved further for it to be used in generalized terrain generation applications. The implementation of TTUR and a larger kernel size for the generator improved the training process and removed a lot of noise from the generated output. Especially the use of (7x7) convolutional kernels in generator and (3x3) convolutional kernels discriminator helps in generating

higher quality images as it allows the generator to retain more information to fool the discriminator. We observed the generator does not learn much in the first few hundred steps. We generated images with dimensions 64x64, 128x128, and 256x256 and to generate one sample it takes 0.75s, 1.14s, and 2.47s respectively on Intel Core™ i5-6200U CPU with four cores. But when the generator is used to generate larger images while it is trained on 64x64 images, it fails to generate diverse images. Furthermore, some images show noisy outputs. We believe this can be tackled by training the model on larger images and then using the generator to generate smaller images. Also, an implementation with a larger model size might help in giving better quality and more diverse images.

References

[1] R. J. Spick, P. Cowling, and J. A. Walker, "Procedural generation using spatial gans for region-specific learning of elevation data," in *2019 IEEE Conference on Games (CoG)*, pp. 1–8, 2019.

[2] K. Perlin, "An image synthesizer," *SIGGRAPH Comput. Graph.*, vol. 19, p. 287–296, July 1985.

[3] S. Stachniak and W. Stuerzlinger, "An algorithm for automated fractal terrain deformation," in *In Proceedings of Computer Graphics and Artificial Intelligence*, pp. 64–76, 2005.

[4] Ness, "Fractional brownian motions , fractional noises and applications (m & van ness 1968) the term " fractional brownian motions "," 2004.

[5] A. Fournier, D. Fussell, and L. Carpenter, "Computer rendering of stochastic models," *Commun. ACM*, vol. 25, p. 371–384, June 1982.

[6] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[7] C. Beckham and C. Pal, "A step towards procedural terrain generation with gans," 2017.

[8] E. Guérin, J. Digne, E. Galin, A. Peytavie, C. Wolf, B. Benes, and B. Martinez, "Interactive Example-Based Terrain Authoring with Conditional Generative Adversarial Networks," *ACM Transactions on Graphics*, vol. 36, no. 6, 2017.

[9] A. Frühstück, I. Alhashim, and P. Wonka, "Tilegan: Synthesis of large-scale non-homogeneous textures," *ACM Trans. Graph.*, vol. 38, July 2019.

[10] N. Jetchev, U. Bergmann, and R. Vollgraf, "Texture synthesis with spatial generative adversarial networks," 2017.

[11] r. r. spick and j. walker, "Realistic and textured terrain generation using gans," in *European Conference on Visual Media Production*, CVMP '19, (New York, NY, USA), Association for Computing Machinery, 2019.

[12] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," 2018.