Arellano, Silvia; Otero, Beatriz; Kucner, Tomasz Piotr; Canal, Ramon

# A 3D Terrain Generator: Enhancing Robotics Simulations with GANs

*Published in:*
Machine Learning, Optimization, and Data Science - 9th International Conference, LOD 2023

*DOI:*
10.1007/978-3-031-53969-5_17

Published: 16/02/2024

*Document Version*
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

*Please cite the original version:*
Arellano, S., Otero, B., Kucner, T. P., & Canal, R. (2024). A 3D Terrain Generator: Enhancing Robotics Simulations with GANs. In G. Nicosia, V. Ojha, E. La Malfa, G. La Malfa, P. M. Pardalos, & R. Umeton (Eds.), *Machine Learning, Optimization, and Data Science - 9th International Conference, LOD 2023* (pp. 212-226). (Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics); Vol. 14505 LNCS). Springer. https://doi.org/10.1007/978-3-031-53969-5_17

# A 3D Terrain Generator: Enhancing Robotics Simulations with GANs

Silvia Arellano1 Beatriz Otero 1
Tomasz Piotr Kucner2 Ramon Canal1

March 26, 2024

**Abstract**

Simulation is essential in robotics to evaluate models and techniques in a controlled setting before conducting experiments on tangible agents. However, developing simulation environments can be a challenging and time-consuming task. To address this issue, a proposed solution involves building a functional pipeline that generates 3D realistic terrains using Generative Adversarial Networks (GANs). By using GANs to create terrain, the pipeline can quickly and efficiently generate detailed surfaces, saving researchers time and effort in developing simulation environments for their experiments. The proposed model utilizes a Deep Convolutional Generative Adversarial Network (DCGAN) to generate heightmaps, which are trained on a custom database consisting of real heightmaps. Furthermore, an Enhanced Super-Resolution Generative Adversarial Network (ESRGAN) is used to improve the resolution of the resulting heightmaps, enhancing their visual quality and realism. To generate a texture according to the topography of the heightmap, chroma keying is used with previously selected textures. The heightmap and texture are then rendered and integrated, resulting in a realistic 3D terrain. Together, these techniques enable the model to generate high-quality, realistic 3D terrains for use in robotic simulators, allowing for more accurate and effective evaluations of robotics models and techniques.

GAN Terrain rendering 3D image generation Robotics simulators Sim2Real

## 1 Introduction

Simulation plays a crucial role in robotics by providing a controlled setting for testing algorithms and prototypes before conducting experiments on physical agents. These tools allow researchers to test robots in unusual, inaccessible, or

even impossible scenarios, improving safety and performance. However, developing 3D virtual environments is a demanding and time-consuming task. These environments have become increasingly complex and detailed over the years, making the process even more challenging. To enhance the user's experience and generate more realistic environments, researchers have been exploring procedural modeling as an ongoing line of research in recent years. [19]

Procedural modeling is a well-established technique in the video game industry, used in popular games like Minecraft [12] to generate diverse, never-ending worlds. Some commercial applications related to animation and video games provide users with the ability to procedurally generate landscapes and scenes for games and driving simulations. However, there is a lack of applications for automatic scenario generation prepared to be applied in robotics or robotics simulators. The requirements for 3D virtual environments in robotics are often different from those in video games or other industries. In robotics, the environments must be physically accurate and realistic to ensure that the results obtained in simulation can be extrapolated to real-world scenarios. This level of fine-tuning requires a high degree of control over the environment and the models used, which is often not possible with traditional terrain modeling techniques.

This paper proposes a complete pipeline to generate a realistic terrain, including both mesh and texture. Besides, the output of this pipeline aims to make it easy for the user to apply various surface parameters in each zone of the mesh based on the terrain type. These parameters, such as friction, contact forces, and other critical factors in robotics, are essential for achieving accurate simulations.

The pipeline is divided into four primary components, detailed in Figure 1. The first component involves heightmap generation, which is critical for obtaining a realistic mesh. This will be further explained in section 3. Enhancing the quality of the heightmap is essential for rendering a terrain precisely. For that reason, a super resolution algorithm is applied, increasing four times the original resolution of the image. This procedure is explained in section 4. Once the heightmap is generated and improved, the third component of the pipeline focuses on generating a texture that meets the user's requirements and is consistent with the generated map's topography. The user is expected to specify how many textures should be combined and how the combination should be done. After that, the system generates the texture. This procedure will be further amplified in section 5. Finally, the output from the previous steps is integrated by rendering the heightmap to obtain a mesh, which is then combined with the texture. The resulting mesh is then exported and can be used in any preferred robotic simulator. The integration step is further explained in Section 6.

The main contributions of this work are the following:

- Creation of an open-source heightmap dataset by extracting data directly from various online sources, as there was no comprehensive publicly available dataset at the time.
- Selection of an appropriate combination of neural networks to achieve

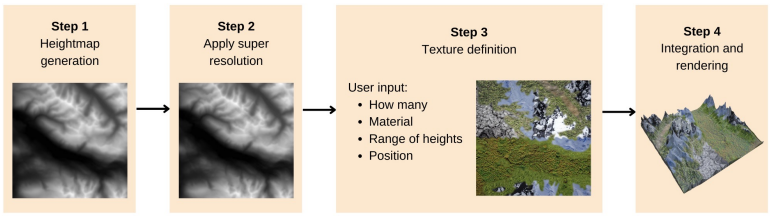



Figure 1: workflow pipeline of the 3D terrain generator, showing the various stages involved in generating a terrain from scratch.

diverse and high quality resources.

- A comparative analysis between the proposed pipeline and other existing techniques for generating 3D terrains for robotics simulations.
- Development of a software capable of generating a wide range of combinations of shapes and terrains to provide robots with diverse training scenarios.

The remainder of this paper is organized as follows. Section 2 provides a review of related work on procedural terrain generation. Section 3 and 4 illustrate the generation procedure of both meshes and textures, explaining the techniques and the Neural Network used in the process. Then, Sect. 5 presents the integration process, and Sect. 6 describes the potential uses and applications of this software in robotics. We draw the main conclusions in Sect. 7.

# 2 Background

## 2.1 Current terrain generation strategies

The aim of procedural modeling is, instead of designing content by hand, to design a procedure that creates content automatically [19]. Some of its advantages include the reduction of cost and time needed for scene generation, a high level of detail in the output without human intervention and low memory requirements. However, the main drawbacks are the difficulty in manipulating and controlling the generation process and the significant amount of computational resources required [7, 19].

Our proposal focuses on procedural modeling of outdoor terrain, which can be approached using three strategies: noise functions, erosion simulation, and texture synthesis. All of these strategies make use of height maps, which are a representation of a terrain's surface as a grid of elevation values.

The first way to generate new height maps is using bandwidth-limited noise functions like Perlin [16], which can capture the nature of mountainous structures. However, due to the randomness of these functions, manipulating the outcome and customizing it according to the user's needs becomes a challenging

task. Gasch et al. [6] use noise functions to generate arbitrary heights, and combine them with previously given artificial features, like figures or letters, in a way that the final result can be perceived as real.

Another strategy for generating terrains is erosion simulation, which involves modifying the terrain using physics-based algorithms that aim to resemble natural phenomena. A renowned study in this field was carried out by Musgrave et al. [13], where they computed the surface erosion caused by water, by considering the quantity of water that would accumulate at each vertex of a fractal height field and the impact of thermal weathering.

The last procedure considered is texture synthesis, which aims to generate new surfaces by studying and finding patterns in existing terrains. In this way, it can capture their realism, so that they obtain compelling results. Panagiotou et al. [15] used a combination of GAN and Conditional GAN (cGAN) to create images that resemble satellite images and Digital Elevation Models that can match the output of the first GAN.

Our approach can be included in the last type of procedure, as we use a Deep Convolutional Generative Adversarial Network (DCGAN), a type of generative model that has shown great success in generating high-quality images, to create heightmaps of terrain that will be rendered as 3D meshes. We combine this method with an Enhanced Super-Resolution Generative Adversarial Network (ESRGAN) to improve the quality of the generated samples, and a technique that is based on chroma keying to assign different textures to the terrain depending on its height.

### 2.2 Related existing applications

In recent years, we can observe a rapid acceleration in research on data-driven methods across multiple fields. However, they are heavily dependent on the amount of available data. As mentioned before, there is a lack of applications for automatic scenario generation in the context of robotics or robotics simulators. Nevertheless, there are similar approaches in the field of video games and animation. Some remarkable commercial applications in this fields are Procedural Worlds [2], which provides Unity [24] users with the ability to procedurally generate landscapes and scenes for games and driving simulation, and iClone 8 [17], a 3D animation software with a special focus on nature terrain generation.

There are also several design programs available, such as Terragen [21], and E-on Vue [20], that allow users to create computer-generated environments. Nevertheless, these programs require expertise in 3D design tools and lack the automatic generation feature that our tool aims to provide.

## 3 Heightmap generation

The first step in the pipeline aims to create a terrain mesh. To create varied surfaces, this approach takes advantage of heightmaps, which are grayscale images that represent the elevation of a terrain and can be transformed into 3D

surfaces once they are rendered. Although these images are in grayscale, we chose to represent them in the RGB color space for our dataset. Our objective for this part of the pipeline is to generate diverse and realistic heightmaps while avoiding those that resemble flat planes with little height variation, as they may not be of great interest. For this purpose, we have studied two different methods: unconditional image generation with diffusion models and Generative Adversarial Networks (GANs).

On the one hand, Denoising Diffusion Probabilistic Models (DDPM), also known as diffusion models, were introduced by Ho et al. [10]. A DDPM is a generative model used to produce samples that match a given dataset within a finite time. The diffusion model consists of two processes: a forward diffusion process that gradually adds Gaussian noise during several timesteps to the input data until the signal has become completely noisy, and a reverse denoising process that learns to generate data by removing the noise added in the forward process.

The diffusion process uses small amounts of Gaussian noise, which allows for a simple neural network parameterization. Diffusion models are known for their efficiency in training and their ability to capture the full diversity of the data distribution. However, generating new data samples with a diffusion model requires the model to be forward-passed through the diffusion process several times, which can be computationally expensive and result in a slower process compared to GANs. In contrast, GANs only require one pass through the generator network to generate new data samples, making them faster and less computationally expensive for this particular task. However, it is important to remark that the computational efficiency of both models can vary depending on factors such as dataset size, model complexity, and implementation details.

On the other hand, GANs, first proposed by Goodfellow et al. in [8], are a type of neural network architecture that include two models: a generator and a discriminator. The generator creates fake data that is meant to be similar to the real data, while the discriminator tries to differentiate between the real and fake data. The objective of the generator is to create synthetic images so that the discriminator perceives them as real. In this study, we utilized Deep Convolutional GANs (DCGANs) to generate heightmaps. The model used in this study was adapted from the code provided by Tang's DCGAN256 repository on GitHub [22]. As one of our goals is to make this tool practical and accessible to a wide range of users, we have chosen to prioritize the use of GANs over DDPM for our task. By doing so, we aim to reduce the computational resources required and increase the speed of the generation process, which will make the software more efficient and agile.

## 3.1 Architecture of the GAN

The GAN architecture utilizes a generator that receives a random noise vector of size 4096 as an input and transforms it into a fake heightmap with a size of 256x256x3. It is composed of eight transposed convolutional layers, as detailed in Figure 2. The discriminator in this study takes in images of size 256x256x3

and processes them through six convolutional layers, as shown in Figure 3. The output of the discriminator is a sigmoid activation function that indicates the probability of the input image being real or fake.

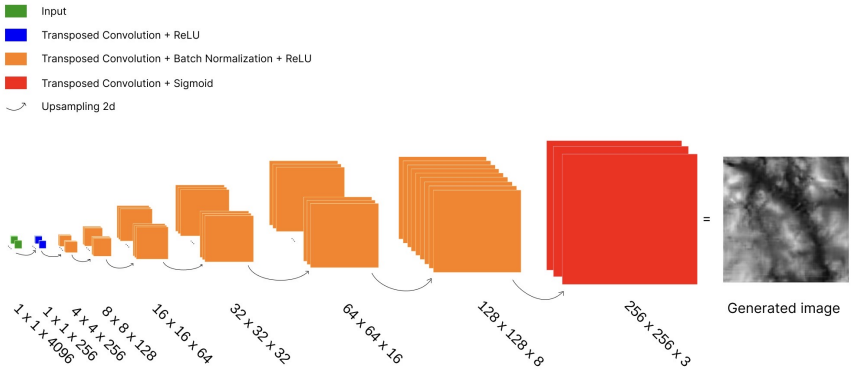



Figure 2: Structure of the DCGAN generator

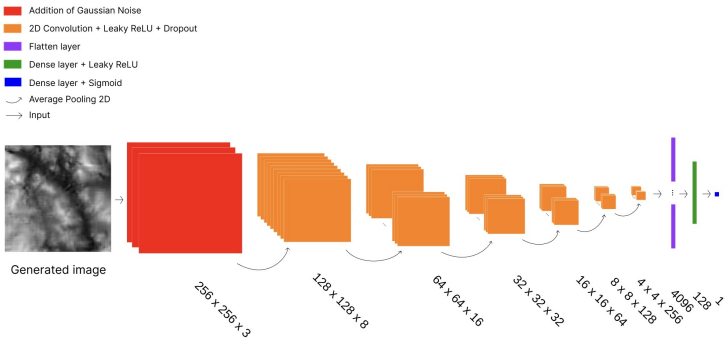



Figure 3: Structure of the DCGAN discriminator

## 3.2 Obtention of the training data

To ensure accurate results during neural network training, it is crucial to use a sufficient number of training samples that are diverse enough to avoid overfitting. However, as there were no open-source heightmap datasets available, we had to create our own. Our dataset consists of 1000 real heightmap images from various terrains worldwide. We obtained the images entirely from the web *Cities: Skylines online heightmap generator* [1] using web scraping techniques. Each heightmap represents an area of 17.28 $km^2$ and has a resolution of 1081 x 1081 px.

To enhance efficiency and facilitate experimentation, we preprocess the samples before training. The images are initially represented with 16 bits per pixel,

but we represent them with 8 bits in RGB format to achieve a reduced file size and ensure compatibility with a broader range of tools and software. Furthermore, to improve computer speed and efficiency, we reduce the resolution of the samples to 256 x 256 px. To avoid a biased dataset with too many plain terrains, which may be of less interest to users, we filter out images based on their standard deviation. To determine the threshold for plain terrains, we compute the standard deviation of a noise image and keep those images that have a standard deviation of 35% or more of that value. This way, we ensure the presence of plain terrains while maintaining diversity.

### 3.3 Training

After preprocessing the data, the GAN is trained using a learning rate of $10^{-4}$ and a batch size of 16. The GAN is trained for 75,000 steps, resulting in the losses and results displayed in Figure 4. The figure shows the losses from both the generator and the discriminator, differentiating between the cases when the discriminator is presented with real or fake images. In addition, the figure contains an example of each part of the training, which are detailed below.

The graph on Figure 4 illustrates that the discriminator and the generator play a min-max game, where if one of them decreases its loss, the other one will increase it. To explain this further, we can divide the training into three parts, separated by dashed lines on the plot. It's worth noting that monitoring the loss values is primarily useful for tracking the GAN's progress over the steps and ensuring that the training is not diverging, rather than measuring the improvement in the quality of the generated images.

In the first part of the training (Part A), corresponding to the early steps of the training, the discriminator starts guessing randomly (since it hasn't received enough training), causing an increase in its loss and a small value in the generator's loss. In the second part (Part B), the rate of change in the losses becomes smoother, and the generated images start to become more realistic and clear, although there are still some areas with blurry patterns. Finally, in the latter part of the training (Part C), the losses become stable, and the generated images are realistic and visually diverse, which satisfies the requirements for our application.

### 3.4 Evaluation

We aimed to evaluate the quality of the generated images using metrics commonly used in the image generation field, such as Inception Score (IS) and Fréchet Inception Distance (FID). However, due to the nature of our images, neither of these metrics may be the most appropriate for our evaluation.

The Inception Score (IS), defined by Salimans et al. [18] is an algorithm used to assess the quality of images created by a generative image model such as a generative adversarial network (GAN). It uses an Inception v3 Network pretrained on ImageNet and calculates a statistic of the network's outputs when applied to generated images. As stated by Barratt et al. [3], in order to obtain a

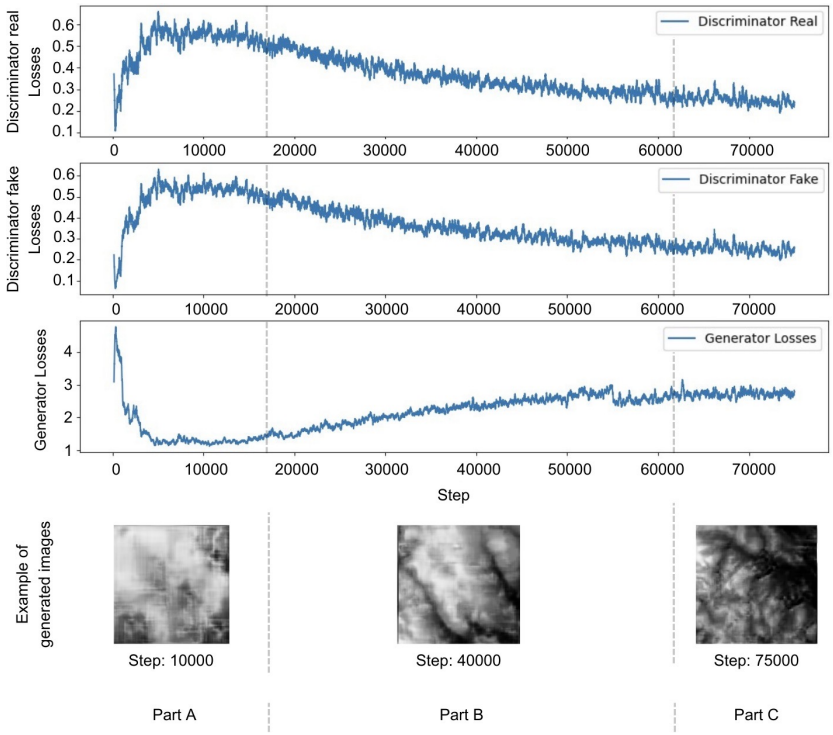



Figure 4: Evolution of DCGAN losses over training epochs, along with representative images from each stage. From top to bottom, the figure displays the losses of the discriminator when given real data, the losses of the discriminator when given fake data, and the losses of the generator.

good IS, the images should be sharp. Besides, it is expected that the evaluated images belong to a great variety of classes in ImageNet. However, our images don't belong to any of the ImageNet classes, and our dataset isn't organized in classes, like the would IS expect. Therefore, the IS may not be the most appropriate metric to use for evaluating image quality.

The FID, defined by Heusel et al. [9] is also used to assess the quality of images generated by a GAN. The FID quantifies the similarities between two image datasets by comparing their distributions, specifically by calculating the distance between feature vectors calculated for real and generated images. A lower FID indicates that the generated images are more similar to the real ones and hence indicates better image quality and a better model.

We compared a dataset of 1000 real heightmaps with a dataset of 1000 generated heightmaps and obtained an FID score of 274.54, which suggests that the two datasets are not very similar. However, it's important to note that the images in our datasets have a wide range of standard deviations, which can have an impact on the FID score. This is because FID measures the similarity

$$\mathbf{PSNR} = 20 \log_{10} \left( \frac{MAX_I}{\sqrt{MSE}} \right) \quad (1) \qquad \mathbf{MSE} = \frac{1}{n} \sum_{i=1}^{n} (x_i - \hat{x}_i)^2 \quad (2)$$

between two sets of images based on their feature statistics, and the feature statistics used in FID computation are sensitive to variations in pixel intensities. Therefore, this would mean that in our case the FID score may not accurately reflect the visual similarity between the real and generated images.

# 4 Super resolution

To render the result with the appropriate resolution, we studied two different methods of increasing the resolution of the heightmap, including ESRGAN proposed by Wang et al. [25] and bicubic interpolation. We evaluated the methods considering metrics such as the Peak Signal-to-Noise Ratio (PSNR) and the Structural Similarity Index (SSIM), defined below.

$$\mathbf{SSIM}(\mathbf{x}, \mathbf{y}) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} + \frac{2\sigma_{xy} + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \quad (3)$$

PSNR measures the ratio of the maximum possible power of a signal to the power of corrupting noise that affects the fidelity of its representation. In eq. (1), the maximum pixel value of the evaluated image is denoted as $MAX_I$, with the minimum level assumed to be 0. The noise level is modelled as square root of Mean Squared Error (MSE).

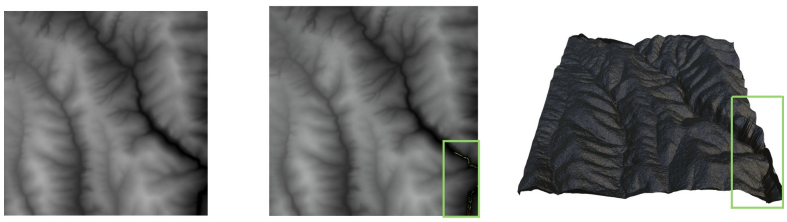
The Mean Squared Error (MSE) is given by equation 2, where $x_i$ represents the value of a pixel in the original image and $\hat{x}_i$ represents the corresponding pixel value in the processed image. A higher PSNR value indicates a lower level of noise in the reconstructed image, which in turn indicates a higher image quality.

The SSIM compares the luminance, contrast, and structure of the two images by taking into account their mean values ($\mu_x$ and $\mu_y$) and variances ($\sigma_x^2$ and $\sigma_y^2$), as well as their covariance ($\sigma_{xy}$). It consists of two terms: the first term computes the similarity of the luminance and contrast between the two images, while the second term computes the similarity of their structure. The constants $C_1$ and $C_2$ are used to avoid division by zero errors and are typically small positive values. A higher SSIM value indicates a higher similarity between the two images.

We evaluated the algorithms using a real heightmap database consisting of 1000 1024 x 1024 pixel heightmaps that were previously downsized to 256 pixels. These downsized images represent examples of images that could potentially have been generated by the GAN model. We intentionally downsized the images to evaluate the ability of the enhancing techniques and the quality of the results when working with lower resolution images. Both algorithms were employed to enhance these downsized images and restore them to their original size.

The evaluation results indicate that, in 54% of cases, bicubic interpolation outperformed ESRGAN with an average difference of 0.5dB. Even though most images generated by ESRGAN are acceptable, it occasionally produced small unexpected artifacts, such as regions with colored pixels, as it can be seen in the example of Figure 5(a), which resulted in slight lumps or peaks in the final mesh and a decrease of up to 9 dB in the PSNR, like in Figure 5(b). Only a 1.2% of the images had a significant impact. The difference in SSIM values between the two techniques was found to be on the order of $10^{-3}$. The time taken to produce results was also considered, with bicubic interpolation taking 2 minutes and ESRGAN taking less than a second.

Despite the artifacts produced by ESRGAN in a subset of images, we decided to use it to enhance the resolution of the generated heightmaps since most of the images were acceptable, and ESRGAN was significantly faster. However, further investigation and refinement may be necessary to address the artifacts in the subset of images where a significant impact was observed. The code used followed the original paper. It was obtained from the TensorFlow Hub and was initially developed by Dey [4].



(a) (b) (c)

Figure 5: Example of a real heightmap (a), the output after enhancing it with ESRGAN (b) and the rendered result (c). The green bounding boxes indicate the perceivable artifacts generated by the ESRGAN.

# 5 Texture generation

As depicted in Figure 1, after the heightmap is generated and its resolution is improved, the next step is to generate its texture. The ability to customize textures based on height and position is a key feature of this application, as it allows users to create terrains that suit their specific needs. Two methods have been studied to achieve a realistic result that fits the topography of the surface, which are presented below.

## 5.1 Pix2Pix

The first approach used is Pix2Pix, a GAN presented by Isola et al. [11] that is used in image-to-image translation, which involves generating an output im-

age that is a modification of a given input image. In this case, image-to-image translation refers to generating a texture that fits the topography of the environment, having the heightmap as an input of the network. This can lead to realistic results, as it is expected to make good associations between heights and material.

To implement this technique, we utilized the Pix2Pix network provided by TensorFlow [23]. The network requires a database consisting of images that contain both the heightmap and its associated real texture. Following the same approach as in Section 3.2, we created the necessary database using web-scraping techniques. However, such dataset is not diverse nor balanced enough. There are more textures for the low height areas than for the high height areas. This leads to a bias that favours materials from flat areas over those in rugged areas, as shown in Figures 6(a) and 6(b), which show a plain and a rugged texture generated by the Pix2Pix network, respectively.

The generated plain texture has similar features to the ground truth image, such as the brownish perturbations in parts A and B of Figure 6. However, the generated rugged terrain doesn't show any of the perturbations of the ground truth image, as shown in part C, making it an unacceptable result.

Another disadvantage of using Pix2Pix is the difficulty users may face when attempting to personalize the generated textures. The network can only replicate the textures present in the training database, and there is a limited amount of textures available. This makes it difficult to customize the map according to the user's specific needs, as the network cannot generate new textures that were not present in the training data. As a result, the user may be limited to using only the available textures, which may not be suitable for all use cases.

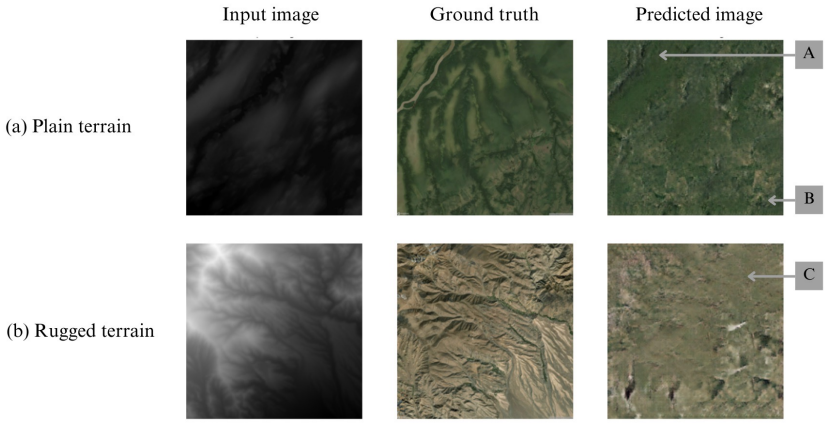
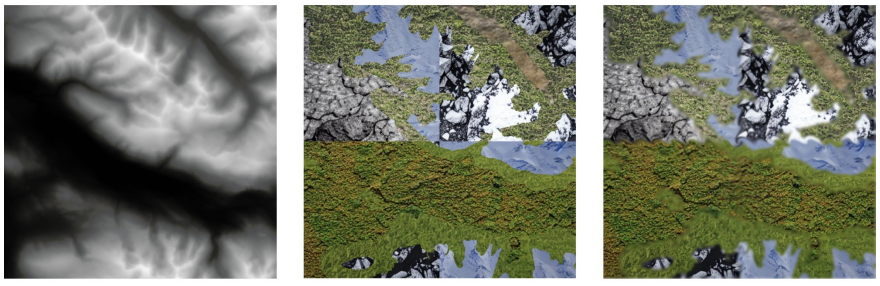



Figure 6: Comparison between the ground truth textures of a plain and a rugged terrain with the predicted textures produced by the pix2pix network, given the corresponding heightmap as input.

## 5.2 Chroma keying

The second approach studied was the application of chroma keying. This technique involves using color information to select and isolate pixels that fall within a range of colors previously defined, known as the "key colors". In our case, the application of chroma keying involved dividing the heightmap into different ranges of grey levels and assigning a different material to each range. To allow for user customization, the user selected the range of greys in which each material should be placed. To create varied combinations of textures, we have chosen eight materials and found an image from the Internet for each one of them. The selected materials were rock, grass, snow, clay and moss, grass with rock, rock with snow and grass with sand. Then, the material images were resized to 1024x1024 so that they could cover the entire heightmap in case needed. Additionally to this height division, a feature was added to include position as a parameter to further customize the texture of the heightmap. However, the application of this technique resulted in evident contours between neighboring textures, creating an unnatural look, as shown in Figure 7(b). To address this issue, we applied post-processing to the image. Firstly, we added a Gaussian blur filter from the OpenCV library [14] to a copy of the texture image. The key parameters, $\sigma_x$ and $\sigma_y$, were set to 2 to achieve moderate smoothing. Next, a binary mask was created using the detected contours from the original image, which was then dilated with a kernel of (5,5) pixels to select the surrounding areas of the contours. Then, we replaced the pixels in the original image that were selected by the mask with their corresponding pixels in the blurry image. These procedures resulted in an image with smoothed contours. However, the axis changes still appeared unnaturally straight. To remedy this, we added a swirl effect in both axes using 16 swirls uniformly distributed. The strength of the swirl was set to 2, and the radius to 100 pixels to avoid over-prominence. The resulting image is shown in Figure 7(c).



(a) (b) (c)

Figure 7: Example of the different phases involved in texture generation: (a) the original heightmap used as input for generating the texture, (b) the texture after applying chroma keying, and (c) the texture after post-processing.

Table 1: Combination of values that the user needs to input to generate the textures given the corresponding heightmaps depicted in Figure 8.

| Figure | Height Value | Position (px) | Texture |
|---|---|---|---|
| 7(a) | (0, 55) | (0, 1023), (0, 1023) | clay |
| | (55, 105) | (0, 1023), (0, 1023) | clay and moss |
| | (105, 155) | (0, 1023), (0, 1023) | rock |
| | (135, 255) | (511, 1023), (0, 511) | snow |
| | (155, 255) | (0, 1023), (0, 1023) | rock with snow |
| 6 and 7(b) | (0, 75) | (0, 511), (0, 511) | rock |
| | (0, 75) | (511, 1023), (0, 511) | rock with sand |
| | (0, 75) | (0, 511), (511, 1023) | clay and moss |
| | (75, 150) | (0, 511), (0, 1023) | grass and rock |
| | (75, 150) | (0, 1023), (0, 511) | grass |
| | (150, 255) | (0, 255), (0, 255) | snow |
| | (150, 255) | (255, 511), (0,255) | rock with snow |
| | (150, 255) | (255, 511), (0, 255) | rock with snow |
| | (150, 255) | (255, 511), (255, 511) | snow |
| 7(c) | (0, 35) | (0, 1023), (0, 1023) | clay |
| | (35, 105) | (0, 1023), (0, 1023) | grass and rock |
| | (35, 105) | (511, 1023), (511, 1023) | clay and moss |
| | (105, 165) | (0, 1023), (0, 1023) | rock with snow |
| | (165, 255) | (0, 1023), (0, 1023) | snow |

# 6 Integration and rendering

To integrate the texture to the mesh, we used the Blender API [5]. However, instead of integrating the texture as a whole, it is integrated material by material. In this way, the map can be exported in pieces to the chosen simulator and assign different properties to each type of material. The resulting files are exported with the extension .obj and .mtl. Some results can be seen in Figure 8.

# 7 Conclusions

Simulation environments are commonly utilized in robotics, but the process of creating realistic 3D terrains for robotic simulation purposes can be challenging and laborious. In this paper, we introduce a system that automatically generates 3D terrains from scratch, generating both heightmaps and textures. The system is composed of four main parts: heightmap generation, heightmap resolution enhancement, texture generation, and integration. After comparing Diffusion Models and DCGANs, we used DCGANs for heightmap generation, as it is computationally less expensive and faster, making it accessible to a wider range of users. For heightmap resolution improvement, we employed ESRGAN, which

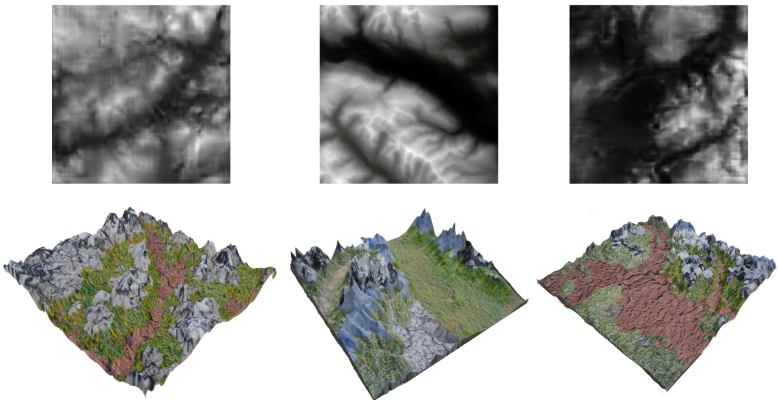


Figure 8: Final rendering of three different terrains, generated by integrating the corresponding heightmaps shown in the top row of this Figure with the textures defined in Table 1.

outperformed bicubic interpolation in execution time and most of the outputs were adequate. For texture generation, we studied two methods: Pix2Pix and chroma keying. Although Pix2Pix was limited in terms of available textures and biased towards flat areas, chroma keying allowed for greater customization by dividing the heightmap into different grey ranges, differentiating between positions, and assigning a different material chosen by the user to each range. To achieve a more natural look, we also used post-processing to smooth the contours and add swirl effects. Finally, we integrated and rendered both the mesh and texture, resulting in .obj and .mtl files that together, represent the generated terrain in the chosen robotics simulator engine. In this way, the user can customize the parameters of each material separately.

As future work, we suggest exploring new techniques for customizing the terrain, so the result can be more similar to the user's idea. Additionally, enhancing the databases to provide the neural networks with more and better quality data could result in further improvements. Besides, addressing the artifacts observed in the ESRGAN's outputs may require further investigation and refinement.

## Acknowledgment

This work is partially supported by the Spanish Ministry of Science and Innovation under contracts PID2021-124463OB-IOO and PID2019-107255GB, by the Generalitat de Catalunya under grants 2021-SGR-00326 and 2021-SGR-00478. Finally, the research leading to these results also has received funding from the European Union's Horizon 2020 research and innovation programme under the HORIZON-EU VITAMIN-V (101093062) project.

# References


[1] Cities: Skylines online heightmap generator. https://heightmap.skydark.pl/, accessed on April 5, 2023

[2] Procedural worlds. https://www.procedural-worlds.com/, accessed on March 13, 2023

[3] Barratt, S., Sharma, R.: A note on the inception score (2018)

[4] Dey, A.: Image enhancing using esrgan. https://github.com-/tensorflow/hub/blob/master/examples/colab/image_enhancing.ipynb (2019), accessed on March 31, 2023

[5] Foundation, B.: Blender (Accessed on March 19, 2023), https://www.blender.org/

[6] Gasch, C., Chover, M., Remolar, I., Rebollo, C.: Procedural modelling of terrains with constraints. Multimedia Tools and Applications **79**, 31125–31146 (2020)

[7] González-Medina, D., Rodríguez-Ruiz, L., García-Varea, I.: Procedural city generation for robotic simulation. In: Robot 2015: Second Iberian Robotics Conference: Advances in Robotics, Volume 2. pp. 707–719. Springer (2016)

[8] Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks (2014)

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

[10] Ho, J., Jain, A., Abbeel, P.: Denoising diffusion probabilistic models (2020)

[11] Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 5967–5976 (2017). https://doi.org/10.1109/CVPR.2017.632

[12] Mojang Studios: Minecraft. https://www.minecraft.net/en-us/, accessed on March 11, 2023

[13] Musgrave, F.K., Kolb, C.E., Mace, R.S.: The synthesis and rendering of eroded fractal terrains. ACM Siggraph Computer Graphics **23**(3), 41–50 (1989)

[14] OpenCV: Opencv library. https://opencv.org/, accessed on April 6, 2023

[15] Panagiotou, E., Charou, E.: Procedural 3d terrain generation using generative adversarial networks. arXiv preprint arXiv:2010.06411 (2020)

[16] Perlin, K.: An image synthesizer. ACM Siggraph Computer Graphics **19**(3), 287–296 (1985)

[17] Reallusion: iclone8, https://www.reallusion.com/iClone/, accessed on March 19, 2023

[18] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X.: Improved techniques for training gans (2016)

[19] Smelik, R.M., Tutenel, T., Bidarra, R., Benes, B.: A survey on procedural modelling for virtual worlds. In: Computer Graphics Forum. vol. 33, pp. 31–50. Wiley Online Library (2014)

[20] on Software, E.: Vue - overview (Accessed on March 19, 2023), https://info.e-onsoftware.com/vue/overview

[21] Software, P.: Terragen (Accessed on March 19, 2023), https://planetside.co.uk/

[22] Tang, George: Dcgan256. https://github.com/t0nberryking/DCGAN256, accessed on March 27, 2023

[23] TensorFlow: pix2pix: Image-to-image translation with a conditional gan. https://github.com/tensorflow/docs/blob/master/site/en/tutorials/generative/pix2pix.ipynb, accessed on April 3, 2023

[24] Unity Technologies: Unity real-time development platform. https://unity.com/, accessed on March 27, 2023

[25] Wang, X., Yu, K., Wu, S., Gu, J., Liu, Y., Dong, C., Qiao, Y., Change Loy, C.: Esrgan: Enhanced super-resolution generative adversarial networks. In: Proceedings of the European conference on computer vision (ECCV) workshops. pp. 0–0 (2018)