HOCHSCHULE
DER MEDIEN

Infinite Adjacent Heightmap Generation with Image Outpainting

vorgelegt von

Ben Häußermann

Matr.-Nummer: 40034
E-Mail: ben.haussermann@gmail.com , bh070@hdm-stuttgart.de

an der Hochschule der Medien Stuttgart
am 28.06.2023

zur Erlangung des akademischen Grades eines Bachelor of Science

Erstprüfer: Prof. Dr. Johannes Maucher
Zweitprüfer: Johannes Theodoridis

„Hiermit versichere ich, Ben Häußermann, ehrenwörtlich, dass ich die vorliegende Bachelorarbeit (bzw. Masterarbeit) mit dem Titel: „¡Infinite Adjacent Heightmap Generation with Image Outpainting¿“ selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Ebenso sind alle Stellen, die mit Hilfe eines KI-basierten Schreibwerkzeugs erstellt oder überarbeitet wurden, kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.“ Ich habe die Bedeutung der ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen (§ 24 Abs. 2 Bachelor-SPO, § 23 Abs. 2 Master-SPO (Vollzeit)) einer unrichtigen oder Mai 2023 Prüfungsverwaltung 6 unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen.“

Zusammenfassung

Diese Bachelorthesis behandelt den möglichen Einsatz von diffusionsbasierten Bildgenerierungsalgorithmen zur prozeduralen Landschaftsgenerierung in Videospielen. Die Leitfrage der Thesis ist, ob es möglich ist mithilfe eines solchen Algorithmus realistischere heightmaps zu generieren als das bisher mit noise basierten Methoden möglich ist.

Heightmaps dienen als Grundlage der Generierung virtueller Gelände in Open-World- besonders prozedural generierten Open-World-Spielen. Obwohl noisebasierte Methoden der dominante Ansatz zur Generierung dieser Heightmaps sind, haben sie Einschränkungen bei der Erstellung realistischer Landschaften mit vielfältigen Merkmalen und können oft repetitiv wirken.

Die Arbeit führt das Konzept der Verwendung von diffusionsbasierten Algorithmen ein, um besagte Heightmaps zu erstellen. Diese Methode beinhaltet das Trainieren eines Diffusionsmodells auf einem Datensatz von realen Geländedaten, die Vorverarbeitung des Datensatzes für die Verwendung mit der spezifischen U-Net-Architektur von "Diffusion Models beat Generative Adversarial Network (GAN)s on Image Synthesis"[ND21a], das Trainieren des Diffusionsmodells, die Verwendung von RePaint [LDR+22] zur Erweiterung, das Erstellen größerer Gelände und das Hochskalieren mit Enhanced Super Resolution Generative Adversarial Network (ESRGAN) [WYW+] für eine verbesserte räumliche Auflösung.

Um die Wirksamkeit der diffusionsbasierten Methode zu bewerten, wurde neben einer visuellen Beurteilung eine Umfrage durchgeführt, die ermittelt wie die Teilnehmenden den Realismus der generierten Gelände im Vergleich zu noisebasierten und realen Geländen wahrnehmen. Die Umfrageergebnisse zeigen, dass diffusionsbasierte Gelände im Allgemeinen als realistischer wahrgenommen werden als noisebasierte Gelände. Wenn sie jedoch

direkt mit realen Geländen verglichen werden existiert ein spürbarer Unterschied im wahrgenommenen Realismus zugunsten dem realen Gelände.

Zusätzlich testet die Arbeit die Performance des diffusionsbasierten Ansatzes und hebt dessen aktuelle Einschränkungen in Bezug auf Geschwindigkeit im Vergleich zu noisebasierten Methoden hervor. Der Versuch einen Hochskalierungsalgorithmus in den Prozess einzubinden um Performance zu verbessern ist erfolgreich. Die Arbeit schließt mit einer Betrachtung der Herausforderungen und potenziellen Zukunft der Integration von diffusionsbasierten Methoden zur Heightmapgewinnung in Videospielen und anderen interaktiven Anwendungen.

Abstract

This bachelor's thesis explores the use of diffusion-based image generation algorithms for procedural landscape generation in video games. The primary objective is to determine if such algorithms can yield a qualitative improvement in the realism of heightmaps compared to conventional noise-based methods. Heightmaps serve as the foundation for generating virtual terrains in open-world and procedurally generated games. While noise functions have been the dominant approach for generating these heightmaps, they have limitations in creating realistic landscapes with diverse features and can often be repetitive.

The thesis introduces the concept of using diffusion-based algorithms, in order to create said heightmaps. This method involves training a diffusion model on a dataset of real-world terrain elevation data, dataset preprocessing for use with the specific U-Net architecture of "Diffusion Models beat GANs on Image Synthesis" [ND21a], training the diffusion model, using RePaint [LDR+22] to outpaint, creating larger terrains, and upscaling with ESRGAN [WYW+] for enhanced resolution.

To evaluate the effectiveness of the diffusion-based method, next to a visual assessment, a survey was conducted to assess how participants perceive the realism of generated terrains compared to noise-based and real terrains. The survey results indicate that diffusion-based terrains are generally perceived as more realistic than noise-based terrains. However, when compared directly to real terrains, there is still a noticeable difference in perceived realism.

Additionally, the paper measures the computational performance of this diffusion-based approach, highlighting its current limitations in terms of speed compared to noise-based methods and successfully uses an upscaling algorithm to decrease sampling time. The thesis concludes by addressing the challenges and potential future directions for integrating diffusion-based methods to create heightmaps in interactive applications, such as video games.

Contents

List of Figures

List of Tables

List of Abbreviations

1 Introduction

The present thesis is dedicated to the use of heightmaps for procedural landscape generation in video games and examines whether a qualitative improvement of these heightmaps is possible through the application of a diffusion-based image generation algorithm.

Heightmaps are essentially matrices in which each value equates to the elevation of its coordinate. They are used to create simple 3D models without overhangs, often representing topographical terrain. Most commonly, these heightmaps are depicted as grayscale images, where the brightness of each pixel corresponds to the height of the point in the coordinate system. In the video game industry, the depiction of realistic and detailed landscapes plays a crucial role, especially in open-world games and the subgenre of procedurally generated open-world games, where heightmaps serve as the initial step in creating these virtual worlds. They provide the topographic foundation upon which later features, like water bodies, flora, and all other procedurally generated elements are based on.

Currently, heightmaps in procedurally generated worlds are almost exclusively obtained from adapted gradient noise functions. The two most popular algorithms developed by Ken Perlin, namely Perlin and Simplex noise, are widely used because their 2-dimensional representation as a heightmap strongly resembles hills and valleys. By using various differing variables in the calculation of 3-dimensional terrain, a wide variety of different landscapes can be generated.

However, noise functions have limitations in creating realistic-looking landscapes. For instance, mountains often appear too round at their peaks or may not form mountain ranges but exist individually in the landscape. Additionally, heightmaps based solely on noise functions can quickly become repetitive without further procedurally generated details.

An alternative to gradient noise may be machine learning-based image generation algorithms. Particularly in recent years, impressive models, capable of generating high-quality images of various kinds have been presented. There are two underlying technologies that currently produce the best results: GANs and Diffusion Models. It is theoretically possible to train these technologies on a large dataset of topographic data to create

heightmaps that more closely resemble real topography.

However, these methods are not inherently capable of creating additional adjoining terrain to existing heightmaps. This problem can be solved using another technology: Outpainting. Outpainting allows extending an image beyond its original boundaries. By applying Outpainting iteratively to previously generated terrain, it is possible to create the desired infinitely sizeable heightmap.

The main question addressed in this work is therefore, whether these image generation algorithms can produce terrain that is perceived as more realistic than conventional noise-based methods are able to.

Additionally, it will be tried to answer the question whether a possible increase in performance can be reached by using an upscale algorithm in order for the model to be able to sample at a lower resolution than is needed.

This thesis is split into the sections: Theory, Method, Results, Discussion and Conclusion. The theory section will go over 2 previous works and explain the methodology chosen. The method section will cover all steps necessary to produce the infinitely scalable heightmap and describe the methodology of a survey conducted to measure its realism. The result section will review those results with a visual assessment and the results of the survey. In the discussion and conclusion section, possible shortcomings, alternative methods, and future work will be discussed and a final conclusion will be drawn.

2 Theory

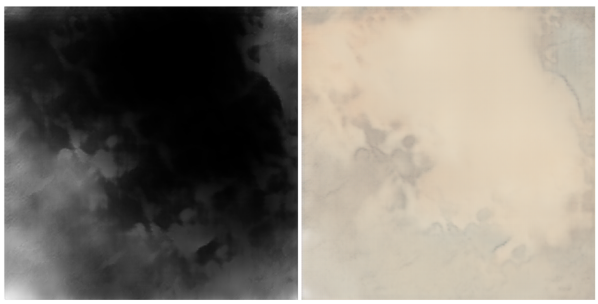
As stated, this chapter will cover two papers that can be classified as previous works. Namely "A step towards procedural terrain generation with GANs" [BP17] and "PTRM: Perceived Terrain Realism Metric"[RK22]. Afterwards the theory behind diffusion models, starting with Convolutional Neural Networks, and the inpainting method proposed in RePaint [LDR+22] will be explained in detail.

2.1 Previous Works

2.1.1 A step towards procedural terrain generation with GANs

Beckham and Pal (2017) [BP17] manage to synthesize individual procedural heightmaps by using a Deep Convolutional Generative Adversarial Network (DCGAN) also trained on satellite data to synthesize heightmaps. They also trained a pix2pix GAN to predict textures based on the synthesized heightmaps.

With both those objectives, they achieved relative success. The generated heightmaps look interesting and show different and distinct features that are neigh impossible to achieve with noise-based heightmap generation. However the results are lacking in certain aspects. The generated textures seemed to only learn very simplified correlations in terrain type and heightmap structure. When trying to generate dessert textures for example, areas of higher elevation where textured white as to represent snow. The GANs where also trained on a very large spacial resolution, with 1 px representing 1 square kilometer. Without details in smaller dimensions it is questionable how many practical applications exist for what they have tested. Sadly, the paper does not discuss the need of generating adjacent heightmap content. Without this feature, only maps of the size specifically trained for can be generated, as it is not possible to extend them once a potential player reaches its edges. It is however, a good proof of concept for heightmaps synthesized this way. The generated terrain, as mentioned, shows variety and does not appear to posses any unrealistic features that could not be imagined to be part of actual terrain. This shows a lot of promise for future experimentation as the GAN in the paper was only



(a)Heightmap (b)Texturemap

Figure 1: Generated Heightmap and the corresponding texture from "A step towards procedural terrain generation with GANs". Both images are 512×512, which corresponds to an area of 512km^2.

Source: [BP17]

trained on less than 900 images.

2.1.2 PTRM: Perceived Terrain Realism Metric

Deepak et al. (2022) [RK22] aim to introduce metrics to calculate the perceived realism of digital terrain. To do this, they categorized synthetic, noise-based terrain and real terrain in into different biome types. They then conducted a study comparing these terrains with each other, with each participant having to chose the more realistic looking one of two terrains. This first study confirmed the assumption that synthetic terrain gets perceived as less realistic than the real terrain. Furthermore, they theorized that in order for a terrain to be perceived as realistic it needs so called geomorphons, such as valleys, ridges, hollows and so forth. By applying a Convolutional Neural Network (CNN) based feature transfer, transferring geomorphons from real to synthetic heightmap data and conducting a second study, in which those newly synthesized terrains outperformed the traditional noise based ones, they were able to confirm that theory. Lastly they trained an MLR model on their dataset to be able calculate the effect of each geomorphon type on the realism score of the terrain. Using the coefficients of this model to weight the different geomorphons accordingly, they are able to calculate a realism score for any arbitrary heightmap, given the size allows for enough geomorphons. Using this approach it is however questionable if it would work on smaller spacial resolutions where geomorphons are less noticeable. All of their tests where done with a spa-

cial resolution of $200m^2$ per pixel. For the purpose of procedural terrain in video games, this might prove to be too large, requiring a lot of detail to be generated using other methods. It is however very beneficial that they could confirm a clear improvement in perceived realism from pure noise-based terrain to their CNN based feature transfer approach.

Since in this thesis, terrain with a spacial resolution of $30m^2$ per pixel will be synthesized, it is unlikely that the exact values they calculated for the geomorphon deltas can be used to accurately determine a realism score. Thus it will be necessary to conduct another independent study to accurately determine a realism score for the, in this thesis, generated terrains.

2.2 Deeplearning for Computer Vision and Generative Models

In recent years, machine learning has experienced remarkable development and proliferation. With increasingly large datasets on various topics, more powerful processors, and a tremendous growth in academic research, machine learning has evolved from niche applications into a significant tool in various fields. From image recognition and speech processing to generative systems for images or even videos, machine learning has made tremendous progress. The heightmaps in this thesis will be synthesized using a diffusion model. This decision was made after multiple failed attempts at training different GANs to outpaint synthesized heightmaps. This means in no way that GANs are not capable of successful outpainting, however when switching to a diffusion model, it took only a single attempt to successfully outpaint heightmaps beyond their original borders multiple times without a visible decay in quality. In the following section the principle behind diffusion models will be explained, starting at their most crucial component, the CNN.

2.3 Convolutional Neural Networks

Not only diffusion models but many classification and generation tasks, benefit from CNNs. Their advantage lies in the ability to independently

learn relevant features of the dataset. This capability of CNNs is achieved through the use of convolutional layers, which apply filters to input data in a localized manner. These filters allow the network to automatically extract important features, such as edges, textures, and patterns, from images or other types of data. By progressively combining multiple convolutional layers, the network can learn hierarchical representations, capturing increasingly complex and abstract features. [Mau]

2.3.1 Convolution Layer

In an image classification task, one such layer has a number of filters, also named kernels, $C_1, C_2, ..., C_k$ which are small arrays of size $(a \times b)$, of trainable floating point values. The input layer X of size $(w \times h)$ is taken and the dot product is calculated for a subsection the size of the kernel and the first kernel. This dot product is then placed on a new grid G_n of dimensions $(m \times n)$.[Mau] 2

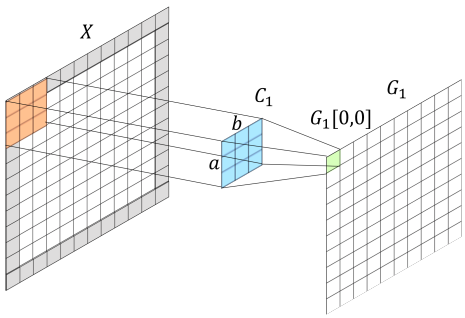


Figure 2: The kernel and a section of the input layer compute their dot product.

Source: Author

The kernel is then "moved" by a Stride S until all subsections of X have been filtered and the output grid is filled.3 Every $G[m, n]$ can be calculated using this formula:

$$G_n[m, n] = \sum_a \sum_b C_n[a, b]X[(m \times S) + a - P, (n \times S) + b - P]$$

This is then repeated for each kernel, resulting in K output grids, also called activation maps, meaning an output of depth K. Width and height of the output depend on the size of the input layer, if padding is used and

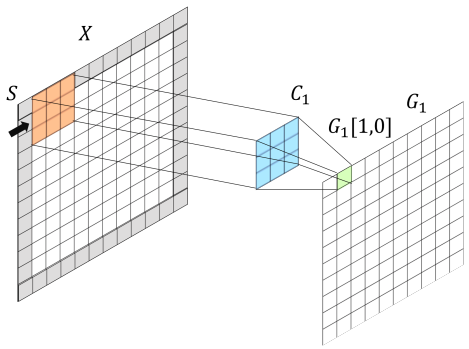


Figure 3: The kernel moves across the input layer.

Source: Author

on S. In the example shown, we use a padding $P = 1$ and $S = 1$, resulting in the width and height staying the same for a kernel of size (3×3). 4

In any other case, the output dimensions N_{out} can be calculated with this formula:

$$N_{out} = [\frac{N_{in}+2P-C}{S} + 1]$$

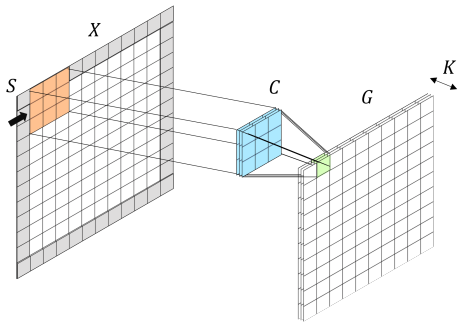


Figure 4: Each kernel is applied to the input, resulting in the full output grid.

Source: Author

The depth of each kernel corresponds to the depth of the layer before it. In the example, a grayscale image is used, meaning a vector with a depth of 1. If applied to a Red Green Blue (RGB) image, a vector of depth 3, each kernel needs to be depth 3 as well. This is also true for the next convolution layer, as every kernel for it has to have a depth of K. After every convolution layer, an activation function has to be used for the network to be able to learn non linear correlations.[KSH12] This activation function is applied to each $G_n[m, n]$ resulting in an output of the same dimensions.

The most common activation functions for CNNs are ReLU (Rectified Linear Unit) and Leaky ReLU. [Sha] 5

ReLU: $f(x) = max(0, x)$ Leaky ReLU: $f(x) = x$ when $X \geq 0$ and aX when $X < 0$ where $0 < a < 1$

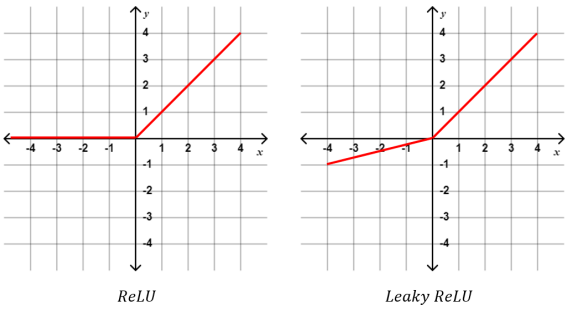


Figure 5: ReLu and Leaky ReLU.

Source: Author

2.3.2 Pooling Layer

Another type of layer, crucial to a CNN, are pooling layers. Pooling layers work to reduce width and height of the output by downsampling each grid. A single filter is applied to each subsection of each output grid, reducing a subsection of size 2×2 or 3×3 to a single value. The depth of the output stays the same. The most common pooling filters are Max-Pool, where the maximum of the subsection is taken, and Avg-Pool, where the average value is calculated. [Mau] 6

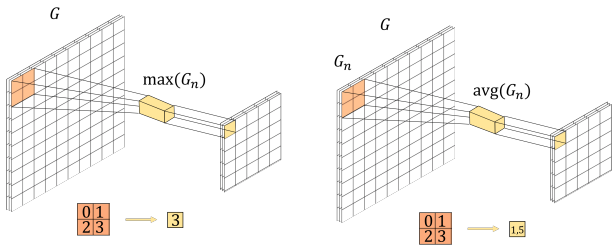


Figure 6: A Pooling Layer is applied to reduce the spatial dimensions of the tensor.

Source: Author

2.3.3 Fully Connected Layer

In the example case of image classification, after an amount of convolution and pooling layers, the tensor gets flattened into a one-dimensional array that can be fed into the neurons of a fully connected layer as known from non convolutional neural networks. [Mau] 7 8

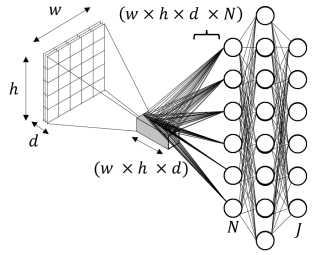


Figure 7: Every value of the flattened tensor is multiplied by a weight and fed into every node of the first fully connected layer.

Source: Author

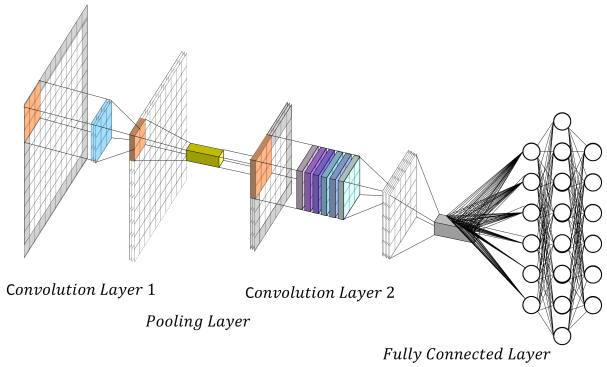


Figure 8: Full example of a convolutional neural network for classifying a 10x10 grayscale image into 6 distinct categories.

Source: Author

In image generation tasks however, a different kind of architecture is used. Since the desired output here is neither a classification nor a regression but an image of the same dimensions as the input image, there needs to be a learnable way to get the convoluted tensor back to the spatial dimensions of the input image.

2.3.4 U-Net Architecture

The U-Net architecture, originally invented for biomedical image segmentation, solves that by introducing transposed convolution layers. [UNe] By applying padding between the values of the convoluted tensor, also called dilating, its width and height are artificially sized up to the desired output plus two. It then applies a number of learnable kernels, equal to the desired output depth, to the padded image and moves them with a Stride of 1. Essentially this works just like a normal convolution layer where the output is dilated, to achieve an output of larger spatial dimensions. [DV18]9

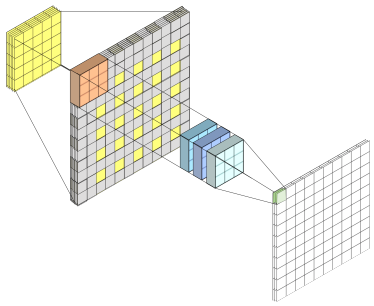


Figure 9: Transposed convolution layer where a tensor with dimensions $5 \times 5 \times 6$ is turned into a tensor with dimensions $10 \times 10 \times 3$

Source: Author

The U-Net architecture can be separated into the encoder and the decoder. The encoder works as known from image classification tasks by applying convolution and pooling layers. This increases feature information while reducing width and height of the tensor and increasing its depth. The decoder part roughly mirrors the encoder part. A mix of transposed and normal convolutions is used to return the tensors original spatial resolutions step by step, essentially creating the desired image from the extracted features.[UNe]

Another feature crucial to a U-Net, are skip connections. Skip connections work by concatenating tensors from the encoder layer onto their counterpart in the decoder layer, combining them depth wise. This way the decoder can recover positional information of features that have been lost in the encoding process.[UNe]10 11

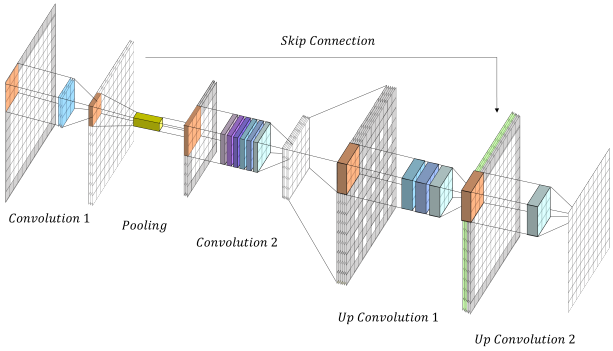


Figure 10: Example for a simple U-Net Architecture for 10 ×10 grayscale images.

Source: Author

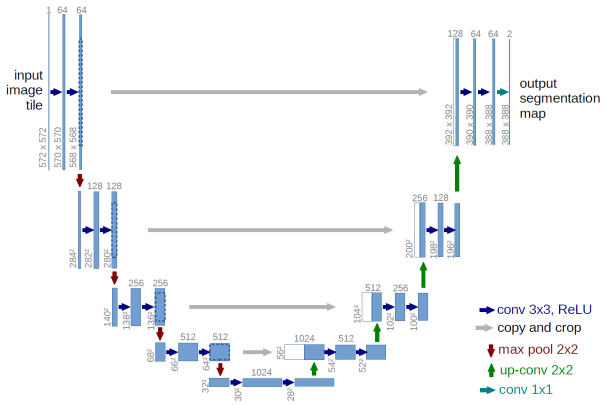


Figure 11: U-Net Example from the original U-Net Paper

Source: [UNe]

2.3.5 Backpropagation Convolution Layer

Up till now, only the way in which a CNN calculates its output, forward-propagation, has been described. To learn however, a CNN needs to back propagate, meaning to compare its output to the correct output and adjust its weights accordingly. To do this a CNN applies a loss function L and uses it to calculate $\frac{\delta L}{\delta G}$ for the Output of the last layer G. With $\frac{\delta L}{\delta G}$ it can now calculate $\frac{\delta L}{\delta C}$ for the filter of the last layer C and $\frac{\delta L}{\delta X}$ for the last layers input X. $\frac{\delta L}{\delta C}$ will be used to update the filter weights, while $\frac{\delta L}{\delta X}$ will be passed to the next layer, going back through the CNN. [Bac]

The chain rule can be applied to $\frac{\delta L}{\delta C}$, giving $\frac{\delta L}{\delta C} = \frac{\delta L}{\delta G} \times \frac{\delta G}{\delta C}$

In the following example the input X is of size (3×3), the filter $C(2 \times 2)$ and the output layer $G(2 \times 2)$ as well. Stride will be set to 1 and left out in the formulas. 12 Since G is defined from forward convolution as:

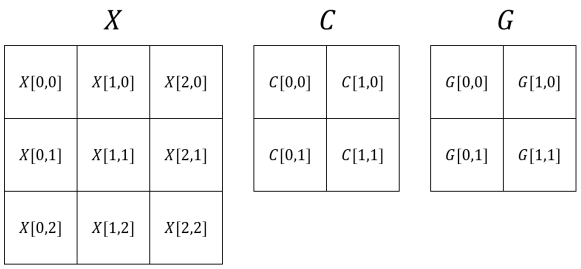


Figure 12: Example of a convolution layer with $X(3 \times 3)$, $C(2 \times 2)$ and $G(2 \times 2)$

Source: Author

$$G[m,n] = \sum_a \sum_b C[a,b]X[m+a,n+b],$$

this makes $G[1,1]$ for example:

$$G[0,0] = X[0,0]C[0,0] + X[0,1]C[0,1] + X[1,0]C[1,0] + X[1,1]C[1,1].$$

Giving the local gradients:

$$\frac{\delta G[0,0]}{\delta C[0,0]} = X[0,0], \frac{\delta G[0,0]}{\delta C[0,1]} = X[0,1], \frac{\delta G[0,0]}{\delta C[1,0]} = X[1,0], \frac{\delta G[0,0]}{\delta C[1,1]} = X[1,1].$$

After the local gradient is calculated, the chain rule can be applied to get $\frac{\delta L}{\delta C}$:

$$\frac{\delta L}{\delta C[a,b]} = \sum_m \sum_n \frac{\delta L}{\delta G[m,n]} \times \frac{\delta G[m,n]}{\delta C[a,b]}$$

Substituting in the calculated local gradients:

$$\frac{\delta L}{\delta C[a,b]} = \sum_m \sum_n \frac{\delta L}{\delta G[m,n]} \times X[a+m, b+m]$$

Which is simply performing a convolution operation on X with the loss gradient from the previous layer $\frac{\delta L}{\delta G}$. 13

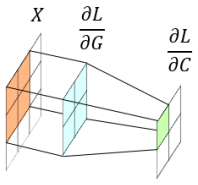


Figure 13: Performing convolution on X with $\frac{\delta L}{\delta G}$ to get $\frac{\delta L}{\delta C}$

Source: Author

$\frac{\delta L}{\delta X}$ can be calculated in a similar manner by calculating the local gradients $\frac{\delta G}{\delta X}$ first, which are:

$$\frac{\delta G[m,n]}{\delta X[w,h]} = C[a+w-m, b+h-n]$$

Applying the chain rule:

$$\frac{\delta L}{\delta X[w,h]} = \sum_m \sum_n \frac{\delta L}{\delta G[m,n]} \times \frac{\delta G[m,n]}{\delta X[w,h]}$$

Substituting with the local gradients:

$$\frac{\delta L}{\delta X[w,h]} = \sum_m \sum_n \frac{\delta L}{\delta G[m,n]} \times C[a+w-m, b+h-n]$$

Which is equal to applying a full convolution on the filter C, rotated by 180°, with the previous loss gradient. 14

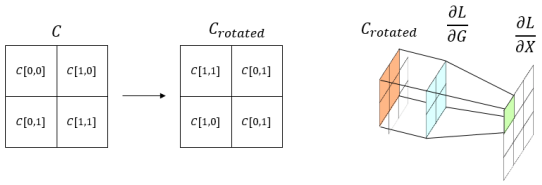


Figure 14: Performing a full convolution on the rotated filter C with $\frac{\delta L}{\delta G}$ to get $\frac{\delta L}{\delta X}$

Source: Author

$\frac{\delta L}{\delta X}$ is then given as $\frac{\delta L}{\delta G}$ to the next layer where the process is repeated. While $\frac{\delta L}{\delta C}$ is multiplied with a learning rate η and the filter C to get the updated filter weight. [Jef16]

$$C' = \eta \times \frac{\delta L}{\delta C} \times C$$

2.3.6 Backpropagation Pooling Layer

Although in a pooling layer there are no learnable parameters, the gradients have to pass through them to retain the appropriate size. In average pooling, each value of the output layer is placed in all slots in the input layer that were used to calculate it. In max pooling, a mask, created during the forward pass is used to tell the backwards pass where the values came from, the other values are filled with 0s. 15 [Jef16]

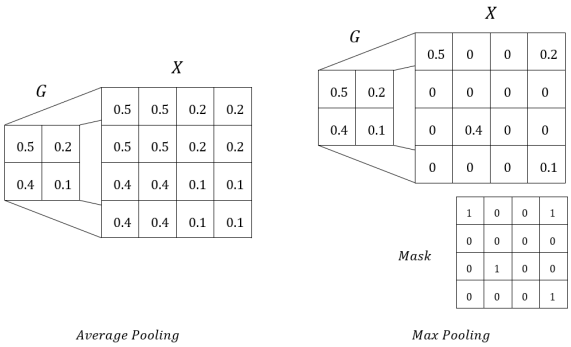


Figure 15: Example for a backwards pass on pooling layers.

Source: Author

2.4 Diffusion Models

Diffusion, in image generation, is a method to generate images, that implements a CNN as one of its components. Similar to a CNN, a diffusion model also has a forward and a reverse process, they are however, very different and not to be confused with each other. In the forward process, an image is taken and gradually diffused, meaning noise is added to the image one step at a time, until only noise is left, and the original image can no longer be discerned. In the backwards process, a CNN is used to denoise the image step by step, until all noise is removed and a full image has been generated from noise alone. [Mau22]

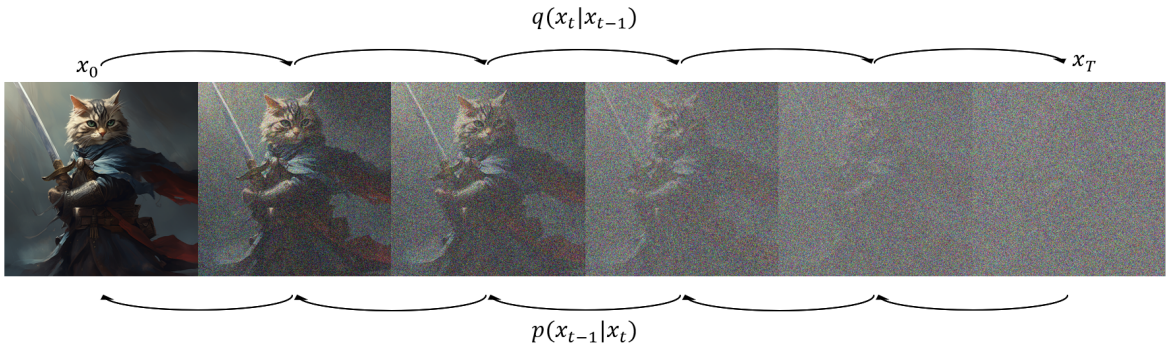


Figure 16: An image that is gradually having noise applied in the forward process and in the reverse process, an image is constructed out of noise.

Source: Author

2.4.1 Forward Process

In the forward process, an image taken from a training's data set x_0, is diffused with a small amount of Gaussian noise, giving x_1. This step is repeated T times with each iteration of x_t becoming noisier, generating us the sequence $x_0, ..., x_T$ with x_T being pure noise. Every x_t can be calculated given x_{t-1} with

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

where β_t is defined by a variance schedule, setting the amount of noise that is added at each step. β_t increases either linearly or with a cosine function, with the latter, information is kept longer into the diffusion process, leading to better results.

$$\{\beta_t \in (0,1)\}_{t=1}^{T}$$

When calculating an arbitrary x_t, one can avoid having to calculate each x_t coming before, by reparameterizing the formula with $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$ a formula for $q(x_t|x_0)$ can be devised as such. [Wen21]

$$\begin{aligned}
x_t &= \sqrt{\alpha_t x}_{t-1} + \sqrt{1-\alpha_t}\boldsymbol{\epsilon}_{t-1} && \text{;where } \boldsymbol{\epsilon}_{t-1}, \boldsymbol{\epsilon}_{t-2}, \cdots \sim \mathcal{N}(0, I) \\
&= \sqrt{\alpha_t\alpha_{t-1}}x_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\bar{\boldsymbol{\epsilon}}_{t-2} && \text{;where } \bar{\boldsymbol{\epsilon}}_{t-2} \text{ merges two Gaussians.} \\
&= \dots \\
&= \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}
\end{aligned}$$

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$$

2.4.2 Reverse Process

In the reverse process, also called denoising process, $q(x_{t-1}|x_t)$ needs to be calculated. This, however, is impossible. Instead a model p_θ can be learned to approximate x_{t-1}: $p_\theta(x_{t-1}|x_t)$. This can be expressed as:

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \boldsymbol{\mu}_\theta(x_t, t), \boldsymbol{\Sigma}_\theta(x_t, t))$$

Since the variance $\boldsymbol{\Sigma}_\theta(x_t, t)$ is fixed to a schedule, a CNN only has to calculate $\boldsymbol{\mu}_\theta(x_t, t)$. To be more precise, the CNN estimates the difference

between x_t and x_{t-1}, which is then subtracted from x_t to get x_{t-1}. This process starts at x_T, which is a randomly sampled Gaussian noise, and repeats T times to x_0, which should resembles an image similar to those in the training data. [Wen21]

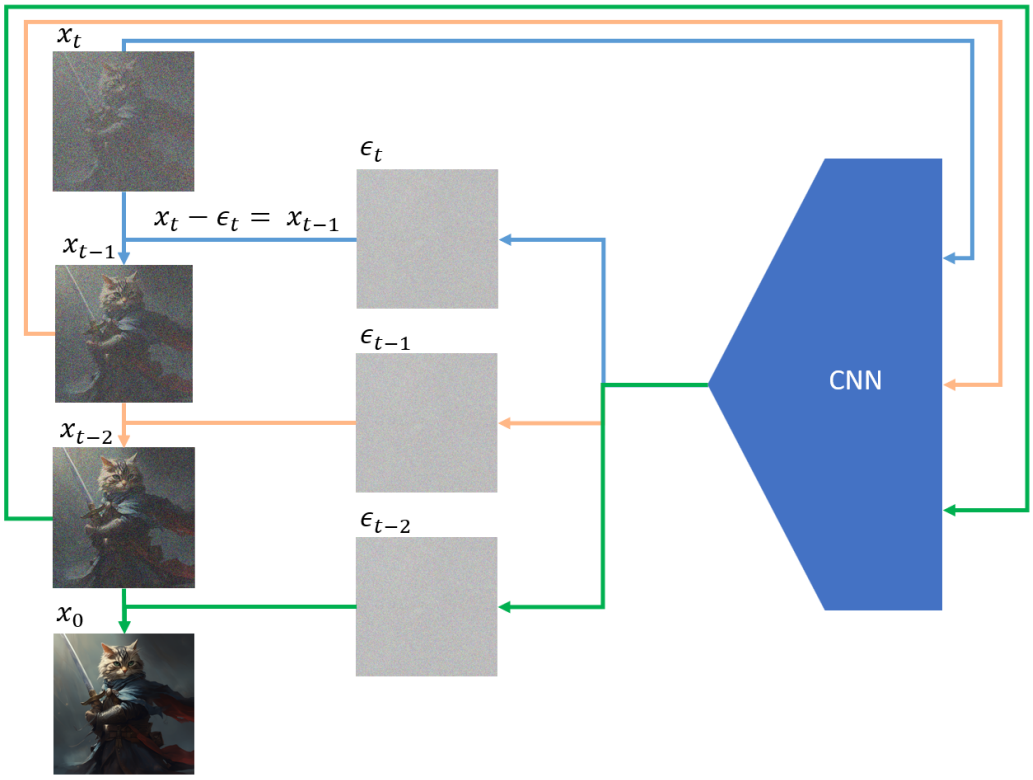


Figure 17: The noised image x_t is passed into the CNN, which calculates the noise increment ϵ_t. ϵ_t is then subtracted from x_t to get x_{t-1}. This process is then repeated until x_0 is reached.

Source: [Mau22], Recreated by Author

2.4.3 Training

The CNN is trained to minimize its loss function, the negative log likelihood.

$$-\log(p_\theta(x_0))$$

This however, is not easily computable as it depends on all other steps in the denoising process and would require keeping track of $T-1$ other variables, which is not feasible in practice. It is however possible to calculate the Variational Lower Bound. Since the Kullback-Leibler Divergence, measuring the statistical distance between $p_\theta(x_{1:t}|x_0)$ and $q(x_{1:t}|x_0)$: [Mau22]

[Wen21]

$$D_{KL}(q(x_{1:t}|x_0)||p_\theta(x_{1:t}|x_0))$$

is always positive, it is given that:

$$-log(p_\theta(x_0)) \leq -log(p_\theta(x_0)) + D_{KL}(q(x_{1:t}|x_0)||p_\theta(x_{1:t}|x_0))$$

which can be rewritten as:

$$L_{VLB} = \mathbb{E}_{q(x_{0:T})}\Big[\log \frac{q(x_{1:T}|x_0)}{p_\theta(x_{0:T})}\Big] geq \mathbb{E}_{q(x_0)} - \log p_\theta(x_0)$$

This can the be further rewritten to a sum of KL-Divergences:

$$L_{VLB}\mathbb{E}_q[\underbrace{D_{\mathrm{KL}}(q(x_T|x_0) \parallel p_\theta(x_T))}_{L_T} + \sum_{t=2}^{T} \underbrace{D_{\mathrm{KL}}(q(x_{t-1}|x_t, x_0) \parallel p_\theta(x_{t-1}|x_t))}_{L_{t-1}} \underbrace{- \log p_\theta(x_0|x_1)}_{L_0}]$$

split here into the cases L_T, L_{t-1} and L_0.

$$\begin{aligned} L_0 &= -\log(p_\Theta(x_0|x_1)) \\ L_{t-1} &= D_{KL}(q(x_{t-1}|x_t, x_0)||p_\Theta(x_{t-1}|x_t)) \\ L_T &= D_{KL}(q(x_T|x_0)||p(x_T)) \end{aligned}$$

Every L_t except L_0 compares two Gaussian distributions and can be computed in closed form. Since x_T is just randomly sampled Gaussian noise, L_T can be ignored.

The goal of this is still to train μ_θ to predict $\tilde{\mu}_t = \frac{1}{\sqrt{\alpha_t}}\Big(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_t\Big)$. The noise term can be re parameterized to predict $\boldsymbol{\epsilon}_t$, taking x_t as input:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}\Big(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(x_t, t)\Big)$$

The loss term L_t can be re parameterized to minimize the distance between

$\tilde{\boldsymbol{\mu}}_t$ and $\boldsymbol{\mu}_0$.

$$
\begin{aligned}
L_t &= \mathbb{E}_{x_0,\boldsymbol{\epsilon}}\left[\frac{1}{2\|\boldsymbol{\Sigma}_\theta(x_t,t)\|_2^2}\|\tilde{\boldsymbol{\mu}}_t(x_t,x_0)-\boldsymbol{\mu}_\theta(x_t,t)\|^2\right] \\
&= \mathbb{E}_{x_0,\boldsymbol{\epsilon}}\left[\frac{(1-\alpha_t)^2}{2\alpha_t(1-\bar{\alpha}_t)\|\boldsymbol{\Sigma}_\theta\|_2^2}\|\boldsymbol{\epsilon}_t-\boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}x_0+\sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t,t)\|^2\right]
\end{aligned}
$$

To further simplify this term, it has been found that the weighting term being ignored achieves better results. Resulting in the simplified loss term: [ND21b] [Wen21]

$$
L_t^{\text{simple}} = \mathbb{E}_{t\sim[1,T],x_0,\boldsymbol{\epsilon}_t}\left[\|\boldsymbol{\epsilon}_t-\boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}x_0+\sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t,t)\|^2\right]
$$

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1,\dots,T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$
5: Take gradient descent step on
$\nabla_\theta \left\|\boldsymbol{\epsilon}-\boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0+\sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon},t)\right\|^2$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0},\mathbf{I})$
2: **for** $t = T,\dots,1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0},\mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t,t)\right)+\sigma_t\mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Figure 18: Pseudocode of both the training and sampling process in a diffusion model.

Source: [HJA20]

2.5 Repaint: Diffusion based Inpainting

In order to create a procedurally growing heightmap it is not enough to generate individual heightmaps of a specific size. A way to use existing heightmaps as input and create new adjacent image parts is needed. This is called outpainting. The more commonly known inpainting refers to techniques that fill in missing spots in images and can, for example, be used to remove people from images and filling the empty space with generated content. Outpainting works very similar, the difference being that with outpainting an area outside the bounds of the input image is filled, expanding the image in the process. This means that it is generally possible to use inpainting techniques for outpainting tasks also, which is why in order to expand the heightmaps in this paper, a method originally developed

for inpainting has been used. Specifically: "RePaint: Inpainting using Denoising Diffusion Probabilistic Models" [LDR+22]. Examples from the RePaint paper can be found in figure 19.

Figure 19: Samples of Inpainted Images from the RePaint paper on the right. In the middle, different steps of the diffusion process for sample 1. On the left, The input image with the mask in translucent purple.

Source: [LDR+22]

RePaint denotes the image with a missing area x and a mask m with $m \odot x$ representing this unknown area and $(1 - m) \odot x$ representing the known area. Every x_{t-1}^{known} can simply be sampled using the forward process. For every $x_{t-1}^{unknown}$, the reverse process is applied. After every step, both are combined, creating x_{t-1}.

$$x_{t-1}^{known} \sim \mathcal{N}(\sqrt{\bar{\alpha}_t} x_0, (1 - \bar{\alpha}_t) I)$$

$$x_{t-1}^{unknown} \sim \mathcal{N}(\boldsymbol{\mu}_\theta(x_t, t), \boldsymbol{\Sigma}_\theta(x_t, t))$$

$$x_{t-1} = m \odot x_{t-1}^{known} + (1 - m) \odot x_{t-1}^{unknown}$$

This process alone however, only works partially and although the inpainted region is using the context of the known region, it is not harmonizing well with it.

As a solution, x_{t-1} is taken from the before mentioned operation and diffused back to x_t. Then a normal denoising step is applied to get back

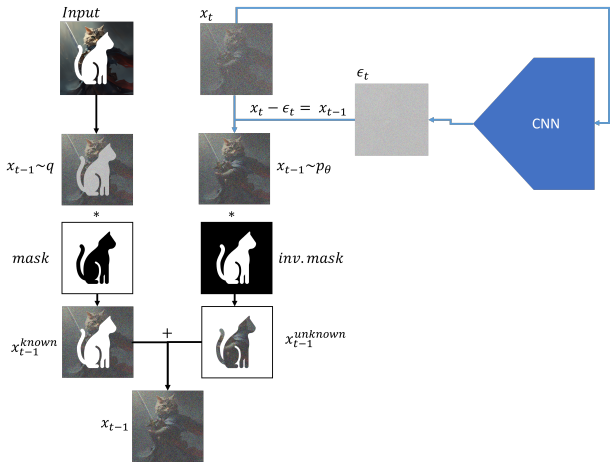


Figure 20: One step of the RePaint process without resampling

Source: [LDR+22], Recreated by Author

to x_{t-1}. Increasing the resample steps leads to better results, saturating at about 10 resamplings.

Notably this approach does not require the model to be separately trained for the outpainting process and allows for usage of any trained checkpoint from a standard diffusion model.

2.6 Enhanced Super-Resolution Generative Adversarial Networks

Since ESRGAN [WYW+] is only used to speed up performance by allowing the diffusion processes to work at a lower resolution, this thesis will not cover the specific methodology behind GANs. This super resolution GAN is trained on image pairs with one high resolution image and one low resolution image. The trained model can then synthesize a high resolution image when given a low resolution one. To do this it also employs a Deep Convolutional Neural Network.

3 Method

The full method, proposed in this paper, to synthesize inifinitely tileable heightmaps can be split into the following steps: Obtaining the dataset, preprocessing the dataset, training the model and training an upscale algorithm, sampling the model, outpainting the sample to the desired size and finally using the upscale algorithm to upscale the heightmap. In the following section, each of these steps will be described in detail.

3.1 Dataset

In order to train a diffusion model to produce a high quality output, it requires a large amount of images as training data. For a good dataset in this usecase counts: The smaller the spatial resolution the better, so less details have to be added via different processes. This obviously has a lower limit, however no large dataset of Digital Elevation Model (DEM)s falls below that. One such dataset, and the one opted for in this thesis, is ASTER GLOBAL DEM (Global Digital Elevation Model (GDEM)) V3, hereby referenced to as GDEM. [AYC22] GDEM provides sattelite based DEMs and covers all land surface between 83 degrees north and 83 degrees south. The data is packaged in ∼23000 GeoTIFF, image files with each file covering an area of roughly 110km^2, or 1×1 arc degree. The spatial resolution of GDEM is 30 × 30 meters per pixel, with each file being 3601×3601 pixels. The depth of each file is 16bit grayscale with 0 being sea level, and a 1 to 1 translation of increase in value to increase in elevation in meters.

Since the current dimensions of the files are too large for this use case, they are each segmented into 196 smaller images of size 256 × 256, covering a terrain of approximately 7.6 × 7.6km^2. Thus the final dataset consists of ∼4.6 million images of dimension 256 × 256 × 1 with a bit depth of 16.

3.2 Preprocessing

As previously mentioned, approaches to use GANs to generate the heightmaps did seem unstable in testing and especially underperformed when trying

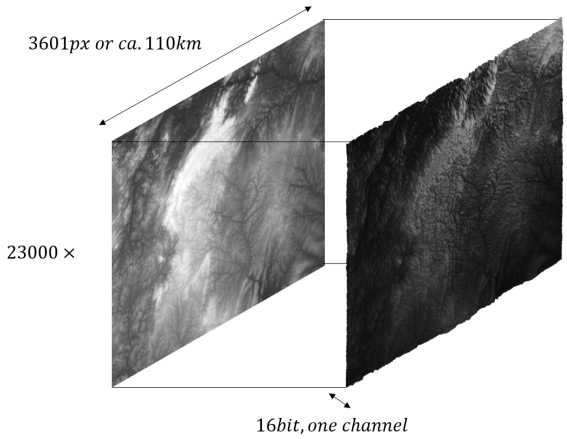


Figure 21: On the left: One GeoTIFF file from GDEM with dimensions 3601× 3601 pixels, covering an area of ca. 110×110km^2, resulting in a spacial dimension of about 30×30m^2 per pixel. On the right: The slightly exaggerated, three dimensional representation of the heightmap on the left.

Source: Author

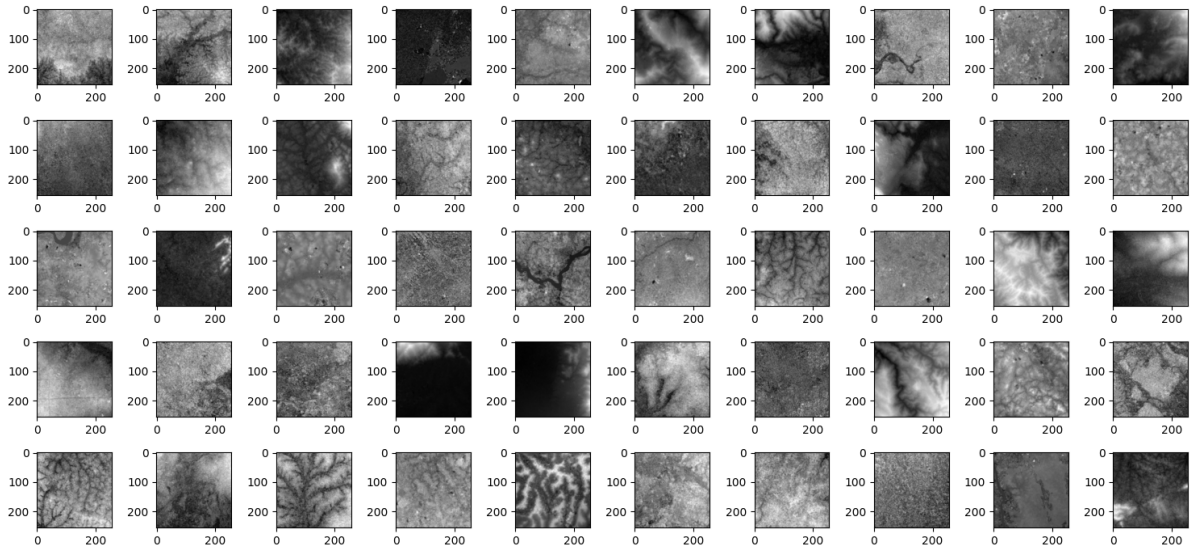


Figure 22: 50 randomly sampled images from the training set with automatically adjusted contrast for visibility.

Source: Author

to continuously outpaint data. When switching to a diffusion based approach, RePaint was chosen since, even though it was technically developed for inpainting tasks, the outpainting examples in the paper seemed very promising. After some successful testing with pretrained model checkpoints, the decision was made to switch. To work without modifications, RePaint requires a pretrained checkpoint from OpenAI's guided-diffusion [ope23], which is the codebase for "Diffusion Models Beat GANs on Image Synthesis" [ND21a]. However, this architecture does only work with 8 bit RGB images and does not support the 16 bit, single channel, data format of the dataset. To overcome this, the images are encoded into RGB using a special algorithm to not lose any data. Every pixel value of the 16 bit image is taken and split into its first and last 8 bit. The first 8 bit get stored in the red channel while the last 8 bit will populate the green channel. Since earths elevation does only very rarely exceed 8191m above sea level, those values can be cut to allow for more contrast in the red channel, by multiplying it to fill out the whole 8 bit. Images featuring this encoding will hereby be refereed to as terrarium images. One such image can be seen in figure 23

$$8191m = \underbrace{000}_{discarded}\ \underbrace{11111}_{rB}\underbrace{11111111}_{green}$$

$$\text{red} = rB \times 8$$

$$RGB = [255, 255, 0]$$

The spread out distribution of bits in the red channel made it easier for the model to correctly learn the connection between the red and green value. This terrarium configuration was able to generate the best results, out of all that were tested.

The main issue of this approach, next to the added time needed to fully transform the full dataset, is the increase in small but important features. Especially areas that feature greater differences in elevation suffer from

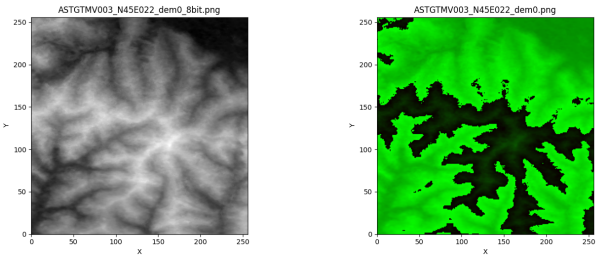


Figure 23: Left: The grayscale image with enhanced contrast for visibility. Right: The image, transformed as terrarium file. Even though it looks like the darker area in the middle of the terrarium image is of lower elevation, the value in its red channel is higher than in its neighboring area, giving it a higher elevation.

Source: Author

this. This makes it more difficult to train the model on a downscaled version of the image set with the intention of upscaling them back later, because downscaling the image loses some of this added detail and will not result in a logical heightmap when reversing the terrarium encoding. To overcome this, the grayscale image needs to be downscaled to 64 × 64 first, before being transformed into RGB. Then both images, the 256 × 256 and 64 × 64 versions, have to be encoded to terrarium individually. The upscaler can then be trained on data pairs that have been generated using this method. 24 This method of up and downscaling is however neccessary to save performance at every following step of the process and it would be a major problem if it would have to be avoided.

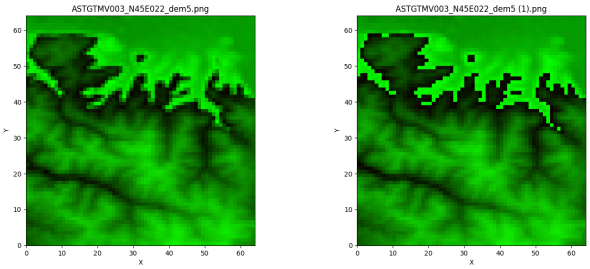


Figure 24: The left image was created by first downscaling the image to 64 × 64 and then applying the terrarium encoding. The right image was created the other way around. The diffusion model was not able to correctly learn realistic elevation features with the left version even after more than 600.000 training steps.

Source: Author

Next to this method without data loss, a secondary method of trans-

forming the images to 8 bit RGB was employed. Since many areas smaller than 7.68km^2 do not feature a larger elevation difference than 256m, they can actually be stored in 8 bit by automatically adjusting the contrast to fit the full value range and then sizing them down to 8 bit format. All figures showing gray scale heightmaps in this thesis were adjusted with the same process for better visibility of features. This process will however lose data with any terrain featuring a greater elevation change than 256m and might be unable to accurately create terrain larger than the learned image size by outpainting it, due to every image having an individually adjusted contrast value. Since this process does not add any details, it only has to be done once to the 256 × 256 version, which can then be downscaled to 64 × 64. 25

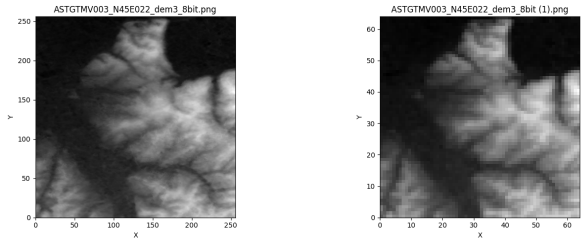


Figure 25: Left, an autocontrasted 256 ×256 slice of terrain data. Right, the same slice downscaled to 64 ×64.

Source: Author

3.3 Training

The next step is to train 4 diffusion models on the datasets created in the previous section. Two for the 64 × 64 and two for the 256 × 256 versions. Additionally the upscaler has to be trained on the terrarium and grayscale datasets separately as well. All models were trained on a NVIDIA A6000 GPU.

3.3.1 Diffusion Models

The 64 × 64 terrarium model was trained for about 16 hours, or 528 thousand steps. The 256 × 256 terrarium model had to be trained for a lot longer to produce comparable results. About 48 hours, or 345 thousand steps. The relevant training and model parameters are shown in table

1. It is possible that different parameters may result in better results or a shorter training time, however those where able to reliably generate adequate results with a variety of differently modified training sets. The

Parameter	Terrarium 64	Terrarium 256
Image Size	64×64	256×256
Channels	128	128
Resolution Blocks	2	2
Diffusion Steps	1000	1000
Noise Schedule	Linear	Linear
Learn Rate	0.0001	0.0001
Batch Size	4	2
Head Channels	4	4
Attention Resolution	32,16,8	32,16,8
Resolution Block Up Down	True	True
Learn Sigma	True	True

Table 1: Table showing all relevant parameters set in training the 2 terrarium models.

Source: Author

grayscale models were able to generate results, visually indistinguishable from real data, in less time than the terrarium models. The 64 × 64 version trained for about 7 hours, or 230 thousand steps and the 256 × 256 version for about 24 hours, or 117 thousand steps. Their parameters were almost identical to the terrarium models, yet they are nonetheless listed in table 2.

3.3.2 Upscaling

To upscale the results from the 64 × 64 terrarium and grayscale models, ESRGAN needs to be trained on a high resolution and low resolution dataset. Both the grayscale and terrarium upscale models were not trained on the full dataset available, but instead on a batch consisting of 196 thousand images. This batch was created by randomly sampling 1000 3601 × 3601 images from the original DEM dataset and slicing them into 196 256 × 256 images each. As mentioned in the preprocessing section, for the terrarium model, the images need to be downscaled first, then both the low

Parameter	Grayscale 64	Grayscale 256
Image Size	64×64	256×256
Channels	128	128
Resolution Blocks	2	2
Diffusion Steps	1000	1000
Noise Schedule	Linear	Linear
Learn Rate	0.0001	0.0001
Batch Size	8	4
Head Channels	4	4
Attention Resolution	32,16,8	32,16,8
Resolution Block Up Down	True	True
Learn Sigma	True	True

Table 2: Table showing all relevant parameters set in training the 2 grayscale models.

Source: Author

and high resolution version can be encoded separately. The images for the grayscale method can be transformed to 8 bit first and then downscaled. Both were trained for about 8 hours, or 150 thousand steps. However, there was no visually discernable quality difference after step 40 thousand. The parameters were identical for both methods and can be seen in table 3.

Parameter	Grayscale	Terrarium
Low Res	64×64	64×64
High Res	256×256	256×256
Workers	4	4
Batch Size	8	8
Crop Size	256	256
Image Channels	3	3
Pixel Weight	0.01	0.01
Feature Weight	1	1

Table 3: Table showing all relevant parameters set in training the 2 ESR-GAN models.

Source: Author

3.4 Outpainting

Since in most procedurally generated open worlds, in video games, it is not sufficient to just create a single square of terrain but instead, more terrain needs to be added once the player explores further. A way to generate new terrain based on already existing one is needed. With RePaint [LDR+22] it is possible to inpaint missing regions of an image as seen in figure 19. This can be used to oupaint an image by simply shifting the image to, for example, the left and creating a new image out of its right half and the new empty space that has formed. Now a simple mask, marking its right half as the missing pixels can be used and the new right half can be inpainted. The generated image is then stitched back together with its original left half. This process is shown in figure 26. Diagonal extensions, corners, can be generated in a similar manner and are visualised in figure 27. In testing it was found that decreasing the known area to values lower than 50%, quickly starts to worsen the similarity of the added terrain and the original. Meaning that the terrain would get increasingly less consistent over a larger area. Utilizing this method of outpainting, an area of any size can now be generated iteratively. As a starting point, an image sampled, using just the standard sampling process, can be used. To make calculations easier, chunks with side lengths of half the generated images, are introduced. Then the side length L ,in chunks, of the desired area is used to calculate the number of middle expansions I_m with $I_m = \frac{L-2}{2}$ and corner expansions I_c with $I_c = (L-2)^2$. Middle expansions are expansions on the same axis as the starting image and expand the images, using masks that cover either the top, bottom, left or right half of the image. Every middle expansion uses two already existing chunks to generate 2 new ones, adding an area of 2×1 chunks. Corner expansions use a mask marking a single corner as missing pixels. Corner expansions are less efficient than middle expansions since every corner expansion only adds a single new chunk instead of two. All 4 possible middle expansions on the starting image are calculated first, then those newly generated areas are used as input for another iteration of middle expansions. This is then repeated until a plus like shape, with width and height of the desired output area, is created. Afterwards, the 4 areas, each in one corner of the image, are filled using the same iterative process

but with corner expansion. Since expansions in different directions don't build upon another, using this method, there can always be 4 expansions that are calculated simultaneously. This process is displayed for an area of 6×6 chunks in figure 28.

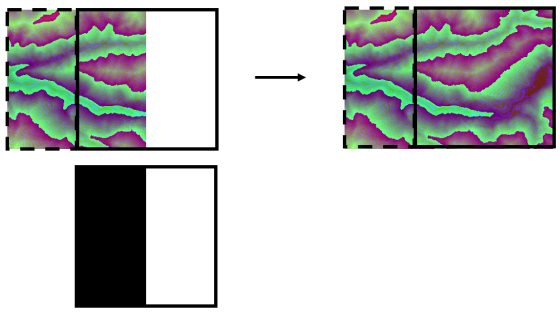


Figure 26: The input image gets shifted to the left by half its width. The resulting image now only consists of the former right half and empty space. A mask is then created, marking this newly formed space on the right as missing pixels. With the mask and the input image, the missing pixels are inpainted. The resulting image is then stitched back together with the former left half, forming a new, larger image. This operation will be referred to as "middle expansion".

Source: Author

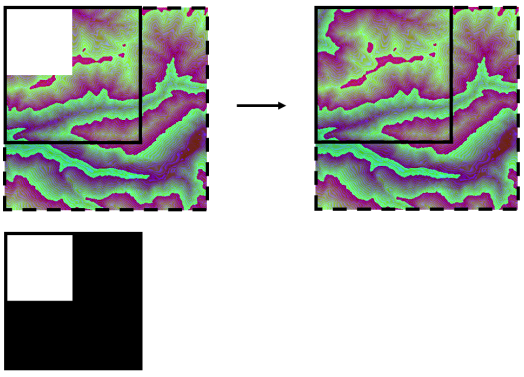


Figure 27: In a similar manner to the middle expansion, corners can be expanded as well. They are notably less efficient since they only add 1 new chunk instead of 2.

Source: Author

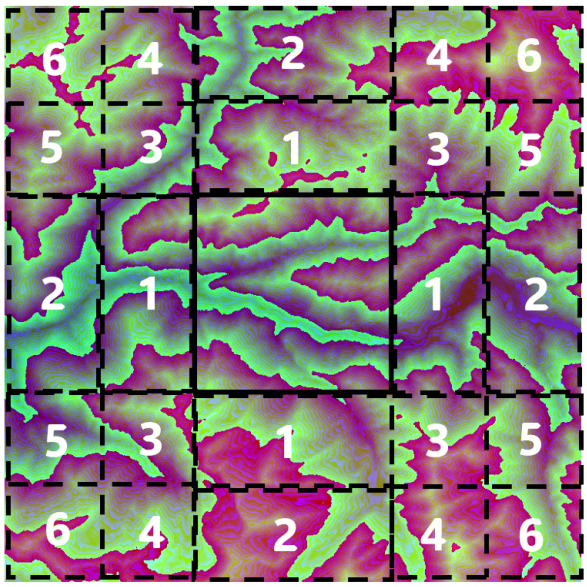


Figure 28: Outpainting order of an area of size 6 × 6 chunks. The middle 8 expansions are calculated first, then, iteratively, the corner expansions follow. This area takes 6 iterations in total with each iteration calculating 4 areas simultaneously.

Source: Author

3.5 Upscaling

As a final step, applied only to the 64 × 64 models, the full heightmap is split into areas of 2 × 2 chunks, or 64 × 64 pixel. Those images are then used as input for their respective upscaling model. The upscaler then generates the 256 × 256 versions of the images, which are then stitched back together to form the fully upscaled area.

The upscaled terrarium heightmap of the desired size can now be transformed back into a 16 bit grayscale image by simply reversing the terrarium encoding done in preprocessing. The resulting grayscale heightmap can then be turned into the 3D model of a terrain in any 3D editing software like Blender 3D or the Unreal Engine Game Editor.

The upscaled grayscale heightmap can be turned into a 3D model without any additional steps.

3.6 Assessing Quality

A rough quality assessment of the results is easily done visually. However, since there is no objective method of measuring the perceived realism of

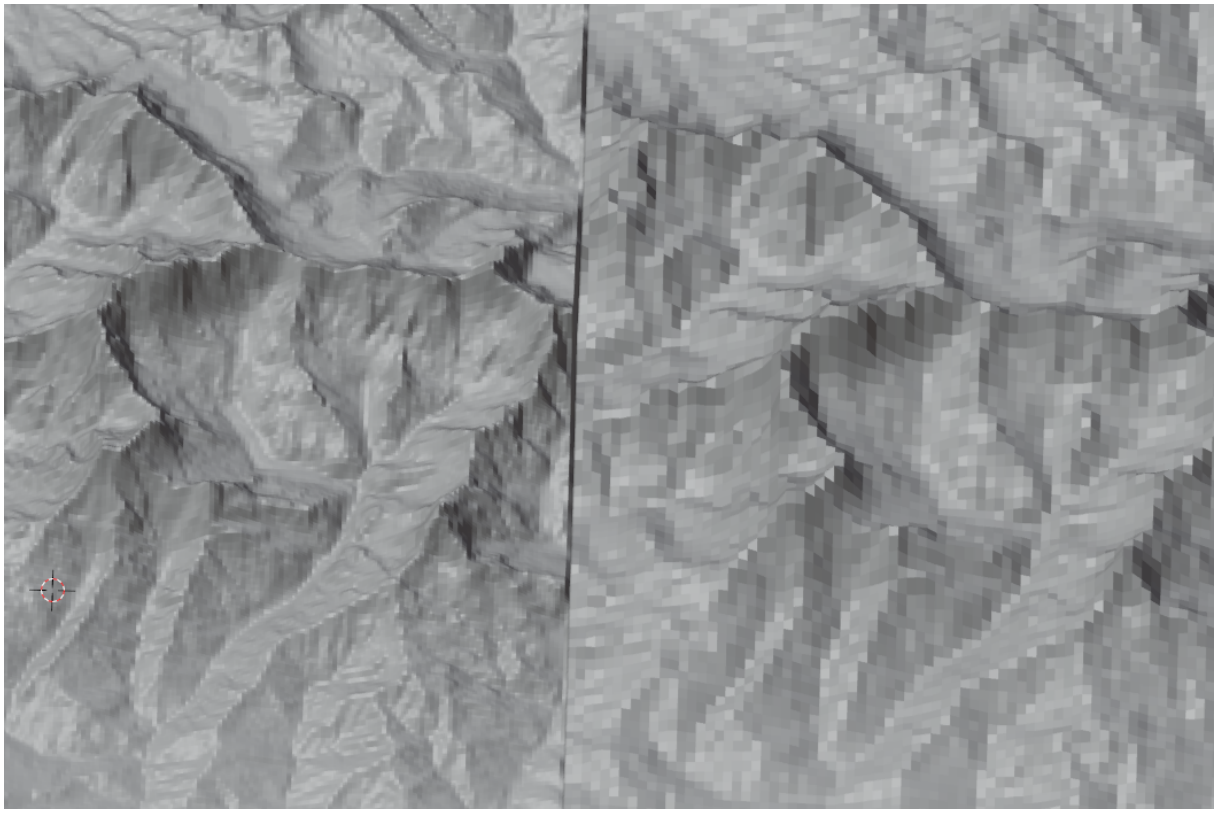


Figure 29: On the left, terrain created by sampling and outpainting in 64 $\times$ 64 and then upscaled to 256 $\times$ 256. On the right, the same sampled terrain before upscaling. Notable the left terrain features new details and is not simply the right version but sharper.

Source: Author

terrain with this spatial resolution, it is neccesary to approximate the perceived realism by surveying a group of people about their opinions. The objective of this survey is: To find out how the perceived realism of the generated terrain compares to noise based and real terrain. For this purpose, 3 types of terrain imagery is needed. Real terrain, noise based terrain and terrain generated using the methods in this paper. Real terrain can simply be obtained by sampling random areas from the training dataset. The terrain generated using the trained diffusion model is obtained using the method previously described. The noise-based terrain is generated using Blender 3Ds built in cloud and perlin noise functions. Since these do not look like terrain with their default variables, the parameters were adjusted to make multiple different repeatable terrain types. From these repeatable terrains, random coordinates were chosen to end up with distinct areas to compare with. Examples for noise-based, real and diffusion-based terrains as they were shown in the survey can be found in figure 30. The areas to compare where set to 7.68km^2, which equals heightmaps of 256 $\times$ 256 pixels. Although this is small enough to be generated without outpainting, random 256 $\times$ 256 areas from larger outpainted images were chosen for the diffusion based terrains. In total, the survey featured 30 images of each type. 15 of the diffusion based terrains were sampled from a heightmap made with the terrarium method, the other 15 were sampled from a grayscale based heightmap. Since the 64$\times$64 models yielded better results in less time than the 256$\times$256 models. All diffusion-based terrains for the study were sampled using these models and upscaled to 256$\times$256. All 90 images were rendered in Blender and feature the same lighting and monotone green coloring.

In the survey, the respondents were first shown an area with the same dimensions on a map and the corresponding terrain to give a better reference for what constitutes a realistic size. They were then shown 45 pairs of 2 images and tasked with marking the one that looked more realistic to them. The 45 image pairs were made up of the 3 possible pairings: Noise-Diffusion, Noise-Real and Real-Diffusion. Due to the odd number of terrarium and grayscale based diffusion terrains, an even distribution into the 2 comparisons was not fully possible. So the terrains being compared

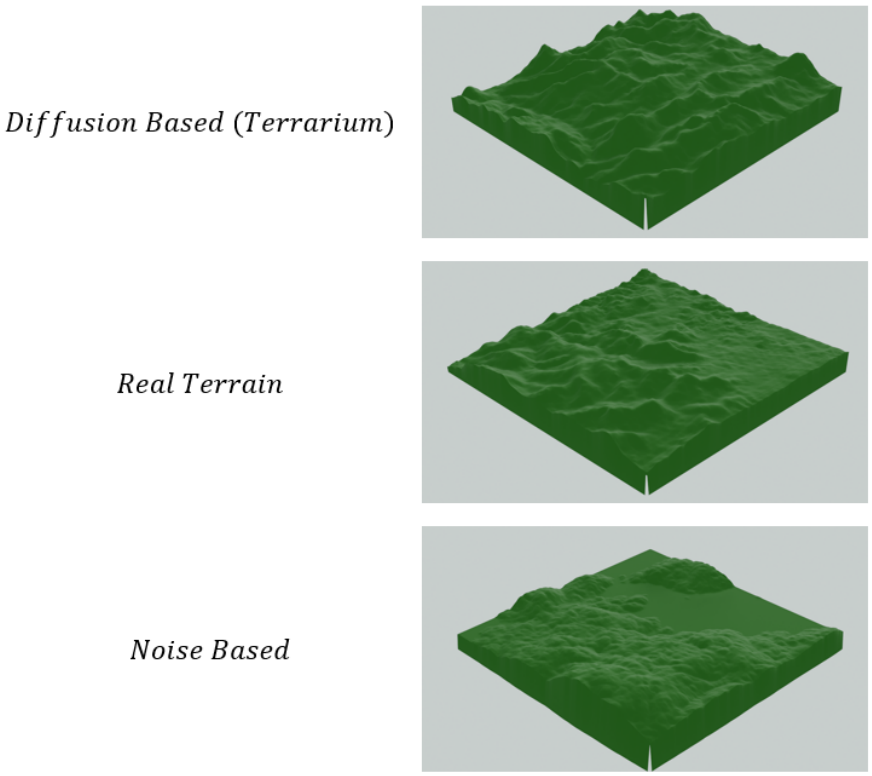


Figure 30: Examples of the different types of terrain exactly as shown in the survey. Respondents were shown 2 such images in each question and asked which one they thought looked more realistic.

Source: Author

with the real terrain are comprised of 8 terrarium based and 7 grayscale based ones, while the terrains being compared with noise based terrain are made up of 7 terrarium and 8 grayscale terrains. To gain further insights into the respondents, they were also asked the following three questions: Have you ever played a videogame? Have you ever played an open world videogame? And have you used a 3D Modelling software more than a couple of times.

4 Results

In this section there will be a review of the methods computational performance, followed by a first visual assessment of the results and a thorough analysis of the surveys results.

4.1 Performance

The main downside of using a diffusion-based approach instead of a traditional noise-based one will likely be the great difference in performance. This paper however, is not focused on improving performance but on proving the concept in general and will leave the task of bettering performance for actual usage during a running video game to future works. Here are the current performance stats of the different processes. All data has been collected on a NVIDIA A6000 GPU.

64×64 Samples	Time
1	34s
4	1m 5s
16	3m 52s

Table 4: Table showing the times for sampling from the diffusion model trained on terrarium or grayscale heightmaps.

Source: Author

256×256 Samples	Time
1	1m 56s
4	5m 27s
16	15m 10s

Table 5: Table showing the times for sampling from the 64×64 diffusion model trained on terrarium or grayscale heightmaps.

Source: Author

64×64 RePaint Expansion	Time
4 Simultaneous Expansions	10m 2s

Table 6: Table showing the average time for one outpainting iteration consisting of 4 simultaneously calculated 64×64 areas.

Source: Author

256×256 RePaint	Time
4 Simultaenous Expansions	25m 20s

Table 7: Table showing the average time for one outpainting iteration consisting of 4 simultaneously calculated 256×256 areas.

Source: Author

Upscaling 64×64 to 256×256	Time
1	0.8s

Table 8: Table showing the average time for one 64×64 image to be upscaled to 256×256. It is notably faster than sampling the diffusion model on 256 ×256.

Source: Author

4.2 Visual Assessment

At first glance, the generated terrarium terrain looks quite similar to real terrain. It features a variety of geomorphic structures that look similar to those created by glacial movements. Even mountains seem to appear in realistic groupings instead of only individually, like is the case in noise based terrain. Terrain that is outpainted on already outpainted iterations does not seem to show a loss in quality, which makes sense given the way in which the outpainting method works. Issues do seem to arise when looking at larger stretches of outpainted terrain. Some mountain ranges end relatively abruptly and quickly turn into terrain that seems illogical next to a mountain. An even more glaring issue with the terrarum terrain is however a very small evaluation change at every chunk seem in the outpainted terrain. This could be fixed with the implementation of a blending algorithm.

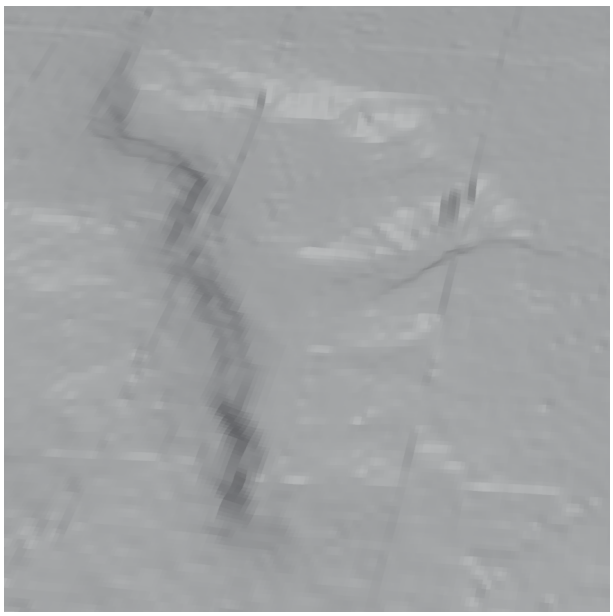


Figure 31: Small visible seam that weirdly only appears in some of the generated terrarium terrain.

Source: Author

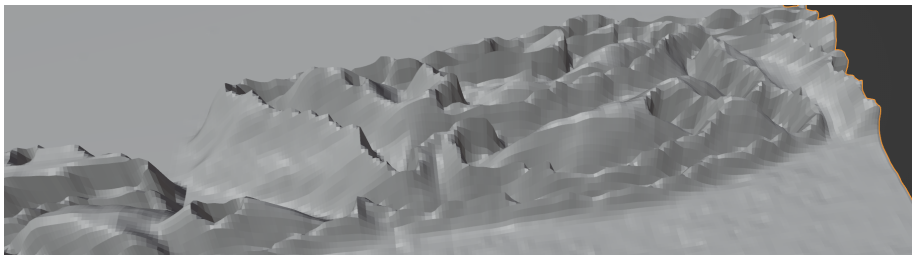


Figure 32: This should be one mountain, but because the red value does not get adjusted correctly, the change in elevation is not correctly calculated.

Source: Author

Very perplexingly however, this only affected some of the generated terrain. Possible explanations for this might be an issue in the code transforming the dataset to terrarium which only affects heightmaps that were on the edge of the 3601 × 3601 image before being sliced. An example of this can be found in figure 31. Another issue with the terrarium terrain is that it often does not correctly adjust the red value. This leads to mountains and other areas of higher elevation change to not have a consistent slope, but instead lose height when approaching their peak. One especially visible example of this can be seen in figure 32.

As expected, the autocontrasted grayscale terrain was prone to different issues. Although any individual 256 × 256 slice seems to be of similar

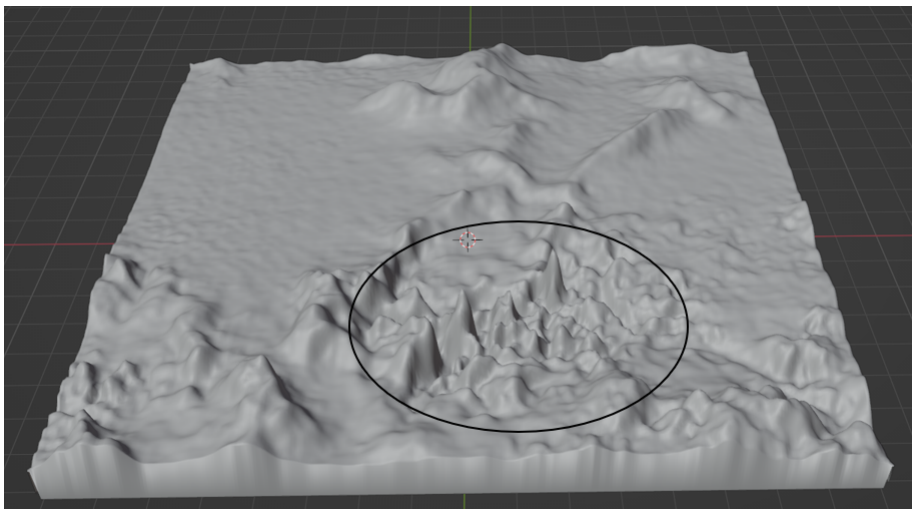


Figure 33: While most of this grayscale based terrain slice looks quite realistic, the circled area clearly needs a different adjustment value than the rest and thus looks not like something that would exist in real terrain.

Source: Author

quality as real terrain, featuring complex geomorphic features that would be very hard to replicate with a noise-based approach, when trying to convert the heightmap to 3D terrain, a factor to multiply the elevation has to be chosen. Since this factor correlates to the contrast of the heightmap and every 256×256 slice technically has its own contrast, this is impossible without compromise. For the results featured in the survey, a factor was chosen, limiting the maximum elevation to 1km. This however still meant that some mountains appeared way to tall for their diameter, while other areas were flatter than they should have been. A good example of this is shown in figure 33.

There is a very clear gap between these results and real terrain taken from the dataset. However many of the results do look more realistic than noise-based terrain, at least at first glance.

4.3 Survey Results

When the survey had ended, it had 27 responses. Although it is hard to infer any statistical significance from this number of respondents, since every respondent answered 45 questions, there are 1215 datapoints. This should be enough to give some insight into how the generated terrain is being perceived compared to noise-based and real terrain. The main hypothesis

is: Diffusion-based terrain is perceived as more realistic than noise-based terrain (A). However the survey also tests: Real terrain is perceived as more realistic than noise-based terrain (B) and real terrain is perceived as more realistic than diffusion based terrain (C).

	Noise	Diffusion	Real
Noise		73,5%	82%
Diffusion	26,5%		58,7%
Real	18%	41,3%	

Table 9: Table showing how often one method (columns) was picked as more realistic when compared with another (rows). When comparing real and noise based terrain, for example, the real terrain was picked 82% of the time.

Source: Author

Due to the large number of answers, the p-values calculated would actually be grounds for statistical significance. Thus the survey suggests that Hypotheses A, B and C are true for the terrain images compared in the survey.

$$P(O|A_0) = \sum_{i=298}^{405} \binom{405}{i} \times 0.5^{405} \approx 3.05 \times 10^{-22}$$

$$P(O|B_0) = \sum_{i=332}^{405} \binom{405}{i} \times 0.5^{405} \approx 7.59 \times 10^{-41}$$

$$P(O|C_0) = \sum_{i=238}^{405} \binom{405}{i} \times 0.5^{405} \approx 2.44 \times 10^{-4}$$

Filtering the data by respondents however shows that 8 of the 27 respondents only perceive the diffusion based method as more realistic than noise based terrain about half of the time. 34 Where as this number shrinks to 1 when comparing noise based to real terrain. 35 This is likely due to imperfections and uncommon geomorphic structures in the diffusion based terrain, which are just not present in real data and are more easily avoided

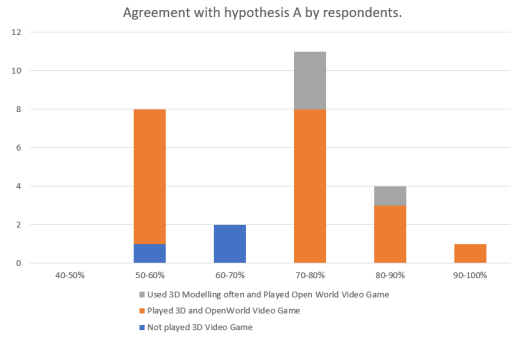


Figure 34: Diagram showing, by what percentage, how many respondents, picked diffusion-based terrain over noise-based terrain. Even though, on average, the former was picked in 73% of cases, 8 of the 27 respondents did not consistently identify it as more realistic than the latter.

Source: Author

in noise based data. However when comparing real data to diffusion based heightmaps, real heightmaps were only picked 58% of the time, with most respondents hovering around a 60% accuracy. 36 Which seems contradictory given that respondents found it to be way easier to distinguish real from noise-based terrain than diffusion from noise-based. Another interesting fact that this data may provide is that there seems to be a tendency for those with experience in 3D modelling and open world video games to avoid picking the noise-based terrain more often. This is likely due to the fact that they have been exposed to noise-based terrain before and can distinguish its look more easily. 34 35 36

The survey shows only a slight difference in pick rate between the two different diffusion methods. This result differs from the results of the visual assessment. A possible explanation would be that due to the low number of samples from both diffusion-based approaches, the quality difference between the two methods was lower in the samples used in the study. The survey suggests however, no large quality difference between the two terrain types. 37

When sorting by image comparisons instead, some interesting but expected patterns appear. Average pickrates of diffusion-based terrains have a far wider spread than those of real terrains. This is likely due to the drastic differences in quality between individual slices of diffusion-based terrain. Such differences are simply not present in real terrain. 38

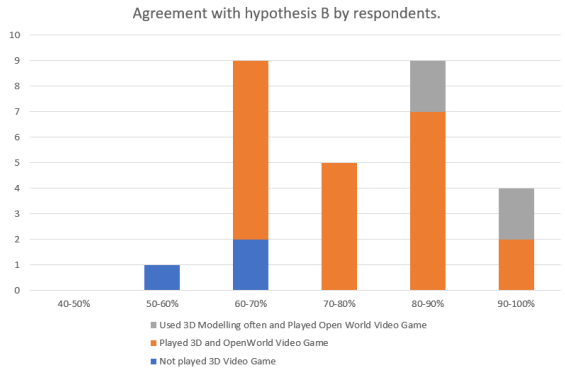


Figure 35: Diagram showing, by what percentage, how many respondents, picked real terrain over noise-based terrain. Although compared to the questions with diffusion based terrain, the overall pickrate has only increased by 10 percentage points, there are almost no respondents in the 50-60% range.

Source: Author

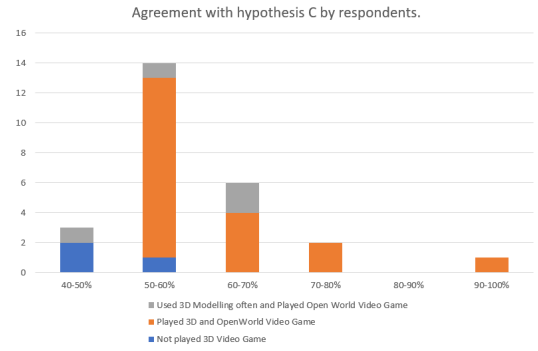


Figure 36: Diagram showing, by what percentage, how many respondents, picked real terrain over diffusion-based terrain. Although real terrain was preffered in 58% of the cases, most respondents seem to struggle with differentiating the two terrain types.

Source: Author

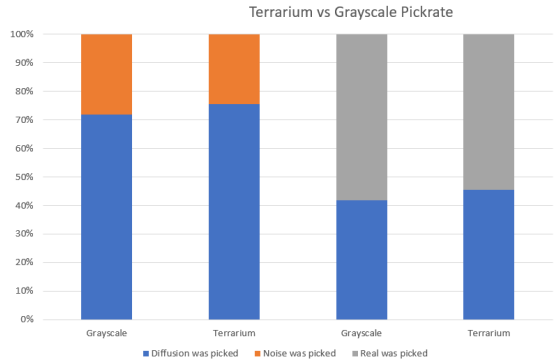


Figure 37: Comparison in pickrate of grayscale terrain and terrarium terrain in percent. In both categories there is a slight tendency towards terrarium terrain.

Source: Author

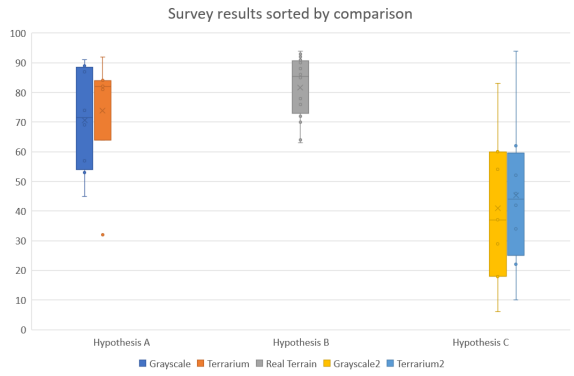


Figure 38: Survey results sorted by comparison. When comparing diffusion-based terrain with noise-based terrain, diffusion-based terrain was picked more often to be more realistic. However there is a significant spread in the values, especially compared with the results regarding hypothesis B. The results regarding hypothesis C however, show an even larger spread, indicating that outcomes were largely dependent on the specific image shown.

Source: Author

5 Discussion

The discussion section delves into the nuanced outcomes of the research, providing an in-depth analysis of the findings and their potential implications for the field of procedurally generated terrain.

The survey outcomes revealed intriguing insights into how observers perceive the generated terrain. Most participants consistently rated the diffusion-based terrain as more realistic than the noise-based terrain, indicating that the method successfully captures certain qualities of authentic landscapes. However, when comparing directly to real terrain there was still, albeit smaller than to noise-based terrain, a noticeable advantage towards the authentic landscapes.

There is the chance that someone with even more experience in synthesizing realistic looking noise-based terrain would have been able to make samples that would have performed better in the survey. This possibility is however unavoidable.

There is one definitive flaw in the survey method however. An optimal survey would have provided the respondents with a way to fully experience the generated terrain in a video game. This way respondents could have seen the terrain from the perspective that it would be experienced in and could perceive changes in it by walking around in it. Due to the large areas of generated terrain, this would likely be very unpractical and would be outside of the time constraints of any potential respondent as well as this thesis. Although the real terrain being successfully picked by far the most out of the 3 different terrain types grants the method a certain validity.

Even though the survey results did suggest a level of success in generating terrain that closely approximates real-world features. Many generated areas, especially those synthesized with the terrarium method, still feature abnormal geomorphic features. And while individual heightmaps synthesized with the grayscale method were less prone to those, it seems impossible to create larger areas than those sampled by outpainting, without a vastly distorted elevation.

Interestingly, the survey comparison between the two diffusion-based methods demonstrated that the difference in perceived realism was not as striking as observed during the visual assessment. It also speaks for the use

of the grayscale method in usecases where only a specific slice of terrain is needed or different slices do not need to fit together. Also in any use case where no higher elevation than 256m is needed could the grayscale approach be a better solution by only training it on such data. This would eliminate the need for autocontrasting and would work without data loss.

A possible third method, which was not explored in this paper, would be to use a different U-Net architecture, built for 16 bit grayscale images and train it directly on the sliced dataset. This would likely result in the "best of both worlds" and will hopefully be explored in future works.

The research's contribution also extends to its examination of computational performance. It is crucial to acknowledge that the current implementation of the diffusion-based approach lags behind traditional noise-based methods in terms of speed, even when downsampling the dataset to a fourth before the process. This performance discrepancy will pose challenges for applications that demand real-time terrain generation, like video games. And even with a drastically improved performance, any additional usage of a GPU will be contested, since during a video game the GPU is already experiencing a heavy workload. While optimizing performance was not the primary focus of this study, the usage of an upscaler to speed up the sampling process improved performance drastically. However, it remains a crucial avenue for future work to enable the integration of diffusion-based methods into interactive applications.

6 Conclusion

The visual assessment as well as the results of the survey do indicate that diffusion-based methods are able to generate terrain that is perceived as more realistic than noise-based terrain.

Despite this success, it's worth noting that a definitive discrepancy still exists when directly comparing the generated terrain to actual real-world landscapes, emphasizing the challenges in achieving absolute realism. Using similar methods with a diffusion model capable of synthesizing 16 bit grayscale data might be able to eliminate many of shortcomings of the results of this paper.

The survey's limitations were acknowledged. Nonetheless, the preference for real terrain by survey participants lends credibility to the method's effectiveness.

Despite its promising outcomes, the diffusion-based methods do currently lag behind noise-based methods in terms of computational performance, which is crucial for real-time applications such as video games. This performance gap raises challenges for integrating diffusion-based techniques into interactive applications. Balancing the computational demands with the benefits of realism remains a consideration for future development in this field.

However a successfull step towards optimization of the method has been taken by using an upscaling algorithm to allow the diffusion model to work with heightmaps of a lower resolution than is required.

In conclusion, this study demonstrates the potential of diffusion-based terrain generation methods to produce realistic landscapes, albeit with certain limitations. By testing different data encodings, computational performance considerations, and the possibility of hybrid approaches, the research contributes to the growing body of knowledge in procedurally generated terrain. As technology advances and the field evolves, it is hoped that these findings will inform future endeavors in creating realistic and captivating virtual environments.

References

[AYC22] M. Abrams, Y. Yamaguchi, and R. Crippen. Aster global dem (gdem) version 3. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLIII-B4-2022:593–598, Jun 2022.

[Bac] Convolutions and backpropagations. https://pavisj.medium.com/convolutions-and-backpropagations-46026a8f5d2c. Accessed: 2023-07-24.

[BP17] Christopher Beckham and Christopher Pal. A step towards procedural terrain generation with gans. Jul 2017.

[DV18] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. 2018.

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. Dec 2020.

[Jef16] Jefkine. Backpropagation in convolutional neural networks, Sep 2016.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[LDR+22] Andreas Lugmayr, Martin Danelljan, Andreas Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. Aug 2022.

[Mau] Johannes Maucher. Convolutional neural networks. https://maucher.pages.mi.hdm-stuttgart.de/mlbook/neuralnetworks/03ConvolutionNeuralNetworks.html. Accessed: 2023-07-24.

[Mau22] Johannes Maucher, 2022.

[ND21a] Alex Nichol and Prafulla Dhariwal. Diffusion models beat gan on image synthesis. Jun 2021.

[ND21b] Alex Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. Feb 2021.

[ope23] openai. Github - openai/guided-diffusion, 2023.

[RK22] Suren Deepak Rajasekaran, Hao Kang, Martin Čadík, Eric Galin, Eric Guérin, Adrien Peytavie, Pavel Slavik, and Bedrich Benes. Ptrm: Perceived terrain realism metric. *ACM Transactions on Applied Perception*, 19(2):1–22, Apr 2022.

[Sha] Sagar Sharma. Activation functions in neural networks. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6. Accessed: 2023-07-24.

[UNe] U-net: Convolutional networks for biomedical image segmentation. https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/. Accessed: 2023-07-24.

[Wen21] Lilian Weng. What are diffusion models?, Jul 2021.

[WYW+] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Chen Loy, Yu Qiao, and Xiaoou Tang. Esrgan: Enhanced super-resolution generative adversarial networks.