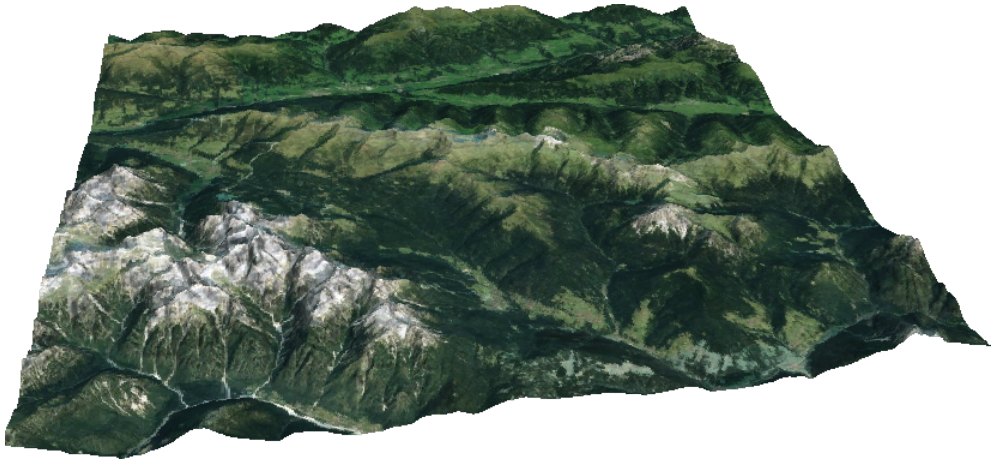
MSE | MASTER OF SCIENCE IN ENGINEERING

Lucerne University of Applied Sciences and Arts
HOCHSCHULE LUZERN

**Master of Science in Engineering**
**Master Thesis**

# Landscape Generation with Generative Adversarial Networks




**Student:** Manuel Jordan
manuel.jordan@stud.hslu.ch

**Advisor:** Prof. Dr. Thomas Koller
thomas.koller@hslu.ch


**Submission Date:** 30.05.2020

# Declaration of Originality

I hereby declare that I wrote my work independently and did not use any other sources and aids than those indicated. All used text excerpts, quotations or contents of other authors were explicitly marked as such. I hereby declare that this thesis represents my original work and that I have used no other sources except as noted by citations.

All data, tables, figures and text citations which have been reproduced from any other source, including the internet, have been explicitly acknowledged as such.

| | |
|---|---|
| Place, Date | Signature |

# Acknowledgements

Hereby I would like to thank all persons who supported me during the work and provided me helpful advice.

Special thanks go to my advisor Prof. Dr. Thomas Koller, who supported and advised me during this time.

Furthermore, I would also like to thank all survey participants for their contribution.

# Abstract DE

Die prozedurale Erzeugung von realistischen Landschaften für Computerspiele ist ein lange erforschtes Thema. Klassische Algorithmen wie Fraktale Rauschfunktionen sind in der Lage, einigermassen realistisch aussehende Landschaften zu erzeugen, die aber nicht alle Merkmale von tatsächlichen Landschaften erhalten. Modernere Verfahren modellieren daher geologische Prozesse wie Erosion, was aber wiederum aufwändig ist, oder überlassen es dem Designer, manuell ein realistisches Gelände zu erzeugen.

Im Rahmen der Master Thesis des Studiengangs Master of Science in Engineering beschäftigt sich die vorliegende Arbeit mit der Erzeugung von 3D Landschaften mittels neuronalen Netzen. In den letzten Jahren wurden große Fortschritte bei der Erzeugung von Bildern, Videos und Text erzielt, insbesondere durch generative kontradiktorische Netzwerke (GAN) und Autoencoder.

Dieser Arbeit führt die Idee von Christopher Beckham and Christopher Joseph Pal [3] weiter, mittels eines GAN texturierte Höhenmodell zu generieren. Das bestehende System wurde abgewandelt und optimiert, weiter wurde auch ein hochwertigerer Datensatz zusammengestellt. Das Modell wurde Satellitenbildern und Höhendaten diverser Faltengebirge trainiert. Um grössere Landschaften zu erzeugen, wurde ein Algorithmus definiert, welcher mehrere Erzeugnisse des Modells zusammenfügen kann.

Um die Qualität der erzeugten Landschaften zu messen, wurde eine Umfrage durchgeführt, in welcher die Probanden echte und generierte Landschaften von einander unterscheiden sollten. Die Ergebnisse zeigen, dass das Modell grossteils sehr realistische Bilder erzeugt. Die zusammengesetzten Landschaften weisen jedoch zum Teils starke Artefakte auf und wirkten auf die Probanden klar weniger realistisch.

Durch eine Vergrösserung des Datensatzes und des Modells, sowie durch Optimierung des zusammensetzenden Algorithmus sollte es zukünftig möglich sein, dieses Problem zu lösen.

# Abstract EN

The procedural generation of realistic landscapes for computer games is a long researched topic. Classical algorithms like fractal noise functions are able to create reasonably realistic looking landscapes, but they do not preserve all features of real landscapes. More modern methods therefore model geological processes such as erosion, which in turn is complex, or leave it to the designer to manually create a realistic terrain.

In the context of the master thesis of the Master of Science in Engineering program, the present work deals with the generation of 3D landscapes using neural networks. In recent years, great progress has been made in the generation of images, videos and text, especially by generative adversarial networks (GAN) and autoencoders.

This work continues Christopher Beckham and Christopher Joseph Pal's [3] idea of generating textured elevation models using a GAN. The existing system was modified and optimized, and a higher quality data set was compiled. The model was trained using satellite images and elevation data of various folded mountains. In order to generate larger landscapes, an algorithm was defined, which can combine several outputs of the model.

In order to measure the quality of the generated landscapes, a survey was conducted in which the participants were asked to distinguish between real and generated landscapes. The results show that the model generates very realistic images for the most part. However, some of the composed landscapes show strong artifacts and clearly appear less realistic to the test subjects.

By enlarging the data set and the model, and by optimizing the composing algorithm, it should be possible to solve this problem in the future.

# Acronyms

**DCGAN** deep convolutional generative adversarial network. 6–8, 11

**GAN** generative adversarial network. 1, 6, 7, 13, 14, 21, 24

**GPS** Global Positioning System. 1

**LR** learning rate. 16

**NASA** National Aeronautics and Space Administration. 7, 9, 24

**WGS** World Geodetic System. 1, *Glossary:* World Geodetic System

# Glossary

**Alpide belt** a orogenic and seismic belt that includes a series of mountain ranges along the southern margin of Eurasia. 10

**Keras** open-source neural-network library written in Python running on top of TensorFlow. 12, 13, 24, 26

**Tensorflow** a free and open-source software library for dataflow and differentiable programming. 11, 24, 26

**World Geodetic System** 3-dimensional coordinate reference system for the earth. *Abbrev.:* WGS

# Contents

**Declaration of Originality** I

**Acknowledgements** II

**Abstract DE** III

**Abstract EN** IV

**Acronyms** V

**Glossary** VI

**1. Introduction** 1
- 1.1. Motivation . . . 1
- 1.2. Objective . . . 1
- 1.3. Terrain Representation . . . 1
  - 1.3.1. World Geodetic System . . . 1
  - 1.3.2. Web Mercator Projection . . . 1
  - 1.3.3. Tiled Web Map . . . 2

**2. State of the Art** 4
- 2.1. Introduction . . . 4
- 2.2. Classical Approaches . . . 4
  - 2.2.1. Fractal Noise . . . 4
  - 2.2.2. Diamond-Square . . . 5
  - 2.2.3. Hydraulic Erosion . . . 5
- 2.3. Novel Approaches . . . 6
- 2.4. Conclusion . . . 7

**3. Methods** 8
- 3.1. Concept . . . 8
- 3.2. Training Data . . . 8
  - 3.2.1. Data Acquisition . . . 8
  - 3.2.2. Data Preprocessing . . . 11
- 3.3. Models . . . 11
- 3.4. Tile composition . . . 14
- 3.5. Quality evaluation . . . 15

**4. Results** 16
- 4.1. Training . . . 16
- 4.2. Survey Results . . . 16
  - 4.2.1. Single Tiles . . . 17
  - 4.2.2. Composed Tiles . . . 18

**5. Discussion** 21
- 5.1. Evaluation . . . 21
- 5.2. Problems . . . 21
- 5.3. Future Work . . . 21

**Bibliography** 23

**List of Figures** **24**

**List of Tables** **25**

**A. Appendix** **26**
A.1. Used software . . . . . 26
A.1.1. Modeling and Training . . . . . 26
A.1.2. Data Acquisition . . . . . 26
A.1.3. Data Visualization . . . . . 26
A.2. Curriculum vitae . . . . . 27

# 1. Introduction

## 1.1. Motivation

The procedural creation of realistic landscapes for computer games is a long researched topic. Classical algorithms like fractal noise functions are able to create reasonably realistic looking landscapes, but they do not preserve all the features of real landscapes. More modern methods therefore model geological processes such as erosion, which in turn is complex, or leave it to the designer to manually create a realistic terrain.

In this thesis we will investigate how generative adversarial networks (GANs) or similar methods can be used to create 3D terrains. A special problem is that large models should be generated, which often cannot be achieved by a single application of the method. Several terrains should therefore be able to be joined together.

## 1.2. Objective

In this thesis, methods are to be developed to generate realistic terrains with height and texture data using machine learning approaches. For this purpose, a data set consisting of real world data will be searched for and models will be learned from it. The focus should be on mountain regions, as they offer diversified elevation models. The generated landscapes will be compared with outputs from classical methods.

In a second step, procedures are to be developed based on this data set in order to be able to join several models together.

Furthermore it is to be examined how realistic the generated landscapes are in comparison to real world data.

## 1.3. Terrain Representation

### 1.3.1. World Geodetic System

The World Geodetic System (WGS) is a 3-dimensional coordinate reference frame for establishing latitude, longitude and heights for navigation, positioning and targeting. The latest revision is WGS84, established and maintained by the National Geospatial-Intelligence Agency since 1984. WGS84 represents the best global geodetic reference system for the earth available at this time for practical applications of cartography, geodesy, and satellite navigation including the Global Positioning System (GPS). This standard includes the definition of the coordinate system's fundamental and derived constants. [2]

### 1.3.2. Web Mercator Projection

Web Mercator or WGS84/Pseudo-Mercator is a variant of the Mercator projection and is the de facto standard for Web mapping applications. It rose to prominence when Google Maps adopted it in 2005. It is used by virtually all major online map providers.

Web Mercator is a slight variant of the Mercator projection. It uses the same formulas as the standard Mercator as used for small-scale maps. However, the Web Mercator uses the spherical formulas at all scales whereas large-scale Mercator maps normally use the ellipsoidal form

of the projection. The discrepancy is imperceptible at the global scale but causes maps of local areas to deviate slightly from true ellipsoidal Mercator maps at the same scale. While the Web Mercator's formulas are for the spherical form of the Mercator, geographical coordinates are required to be in the WGS 84 ellipsoidal datum. This discrepancy causes the projection to be slightly non-conformal.

Web Mercator shares some of the same properties of the standard Mercator projection: north is up everywhere, meridians are equally spaced vertical lines, but areas near the poles are greatly exaggerated.

Unlike the ellipsoidal Mercator and spherical Mercator, the Web Mercator is not quite conformal due to its use of ellipsoidal datum geographical coordinates against a spherical projection. The benefit is that the spherical form is much simpler to calculate. [1]

### 1.3.3. Tiled Web Map

A tiled web map (or "slippy map" in OpenStreetMap terminology) is a map displayed in a browser by seamlessly joining dozens of individually requested image or vector data files over the Internet. It is the most popular way to display and navigate maps. Google Maps was one of the first major mapping sites to use this technique. Most tiled web maps follow certain Google Maps conventions and the de facto OpenStreetMap standard: [12]

1. Tiles are 256x256 pixels
2. An X and Y numbering scheme
3. The origin is top left
4. PNG images for tiles
5. At the outer most zoom level, 0, the entire world can be rendered in a single map tile.
6. Each zoom level doubles in both dimensions, so a single tile is replaced by 4 tiles when zooming in.
7. The Web Mercator projection is used.

Figure 1.1.: Different zoom levels of a tiled map, represented as a pyramid, Source: MapTiler [1]

[1] https://www.maptiler.com/tools/tiles-a-la-gmaps/img/pyramid.png

To convert from the WGS84 geographic latitude and longitude coordinates to X and Y, the following formulas are used: [16]

$$x = \left\lfloor \frac{lon + 180^\circ}{360^\circ} * 2^z \right\rfloor \tag{1.1}$$

$$y = \left\lfloor \left( 1 - \frac{\ln\left(\tan(lat * \frac{\pi}{180^\circ}) + \frac{1}{\cos(lat * \frac{\pi}{180^\circ})}\right)}{\pi} \right) * 2^{z-1} \right\rfloor \tag{1.2}$$

$$lon = \frac{x}{2^z} * 360^\circ - 180^\circ \tag{1.3}$$

$$lat = \arctan\left(\sinh\left(\pi - \frac{y}{2^z} * 2\pi\right)\right) * \frac{180^\circ}{\pi} \tag{1.4}$$

$lat$ = Latitude

$lon$ = Longitude

$z$ = Zoom

Tiled web maps are limited at a latitude of around 85 degrees to make them square. This can be calculated with the formula 1.4 with $y = 0$:

$$lat = \arctan\left(\sinh\left(\pi\right)\right) * \frac{180^\circ}{\pi} \approx 85^\circ \tag{1.5}$$

# 2. State of the Art

## 2.1. Introduction

The procedural creation of realistic landscapes for computer games is a long researched topic. Classical algorithms like fractal noise functions are able to create reasonably realistic looking landscapes, but they do not preserve all the features of real landscapes. More modern methods therefore model geological processes such as erosion, which in turn is complex, or leave it to the designer to manually create a realistic terrain.

## 2.2. Classical Approaches

The basic idea of the classical approaches is to use handcrafted algorithms to model structures and processes from nature. On the one hand it is tried to reproduce the self-similarity of landscapes with fractal functions. On the other hand geological processes like erosion are simulated to make the landscapes more realistic.

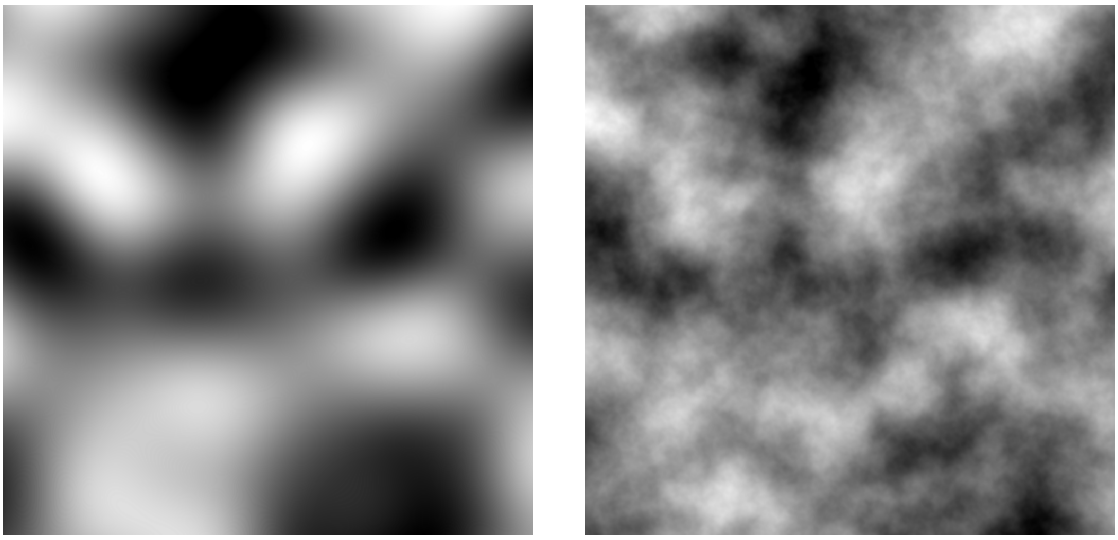
### 2.2.1. Fractal Noise

Fractal structures can be created by scaled superposition of a basic function. For landscapes or textures a gradient based noise function is often used. Well known examples are the Perlin-Noise and the Simplex-Noise functions, developed by Ken Perlin [14]. Both are n-dimensional noise functions, but the simplex noise requires less computational effort and is much more performant, especially in higher dimensions, because its complexity increases with $O(n^2)$ more slowly than the $O(2^n)$ with the Perlin noise. Compared to Perlin noise, simplex noise also has fewer directional artifacts and is therefore isotropic. [6]

Noise functions can be transformed to an $\frac{1}{f}$-noise by superposition. The value for each pixel is: [17, p. 169ff]

$$P_{xy} = \sum_{i=0}^{n} \frac{Noise(x * 2^i, y * 2^i)}{2^i} \tag{2.1}$$

Figure 2.1 demonstrates the application to the Simplex-Noise.

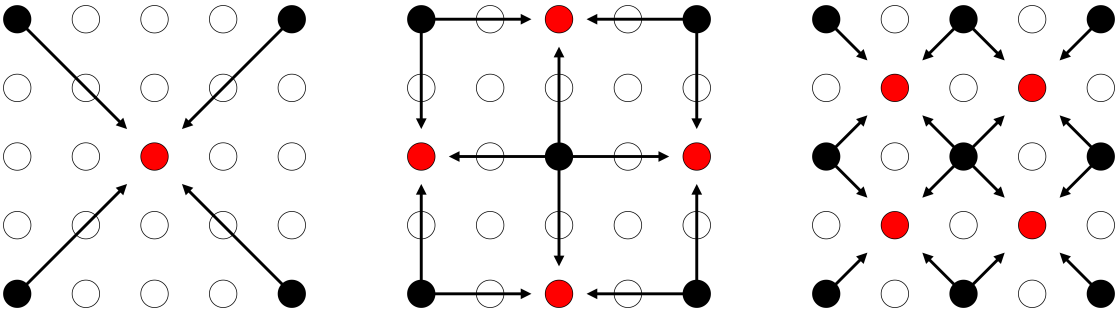


(a) Simplex-Noise (b) $\frac{1}{f}$ Simplex-Noise

Figure 2.1.: Conversion of the Simplex-Noise to an $\frac{1}{f}$-noise

### 2.2.2. Diamond-Square

A simple alternative way to create two-dimensional fractal structures is the Diamond-Square-Algorithm by Fournier et al.[5]. In this algorithm, the center of the outermost square is determined starting from four corner points with a random value. This corresponds to the mean value of the corner points plus a random value. This step is carried out alternately diagonally and orthogonally for the newly created square and diamond shapes (see figure 2.2).[17, p. 169ff]



(a) Diamond step (b) Square step (c) Diamond step

Figure 2.2.: Procedure of the Diamond-Square-Algorithm

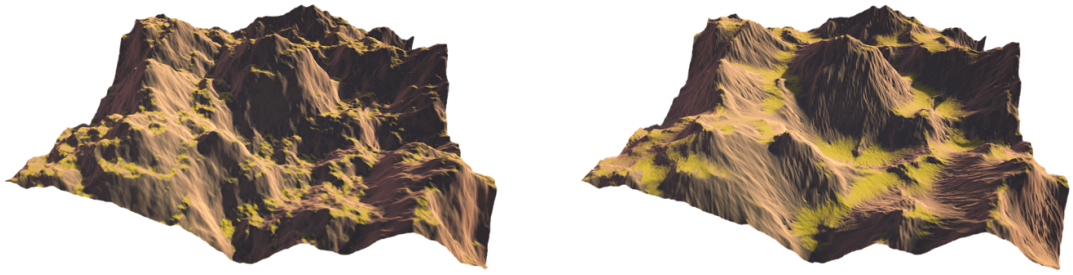
### 2.2.3. Hydraulic Erosion

Both presented methods were accused of being flawed since generated mountains and valleys showed similar features and the terrain has the same visual characteristics when mirrored upside-down. This particular feature can not be found in nature because of the effects of erosion.

The first algorithm to simulate hydraulic erosion on a terrain was introduced by Musgrave et al. [13] in 1989. The introduced procedure adds water to the vertices of a terrain mesh generated from a heightmap. This water then flows to all lower neighbouring vertices. During this

process sediment is taken from higher vertices and distributed to lower ones as it occurs in nature.

A sample implementation by Sebastian Lague shows the effect of the erosion. Figure 2.3 shows the comparison of an uneroded landscape to an eroded one.



(a) Fractal noise (b) Eroded fractal noise

Figure 2.3.: Effect of a simulated erosion, Source: Sebastian Lague [1]

## 2.3. Novel Approaches

Recent years have seen major progress in generating images, videos and text, notably using GANs. Along with autoencoders, GANs are nowadays popular models for generating images. Jaydeep T. Chauhan [4] provides a comparative analysis of these two generative models on basis of their objective, performance and architecture. Both models have their own pros and cons. In general it can be said that a GAN tends to produce sharper and more realistic images than a autoencoder. But a GAN is also more difficult to train, a it is based on the zero-sum non-cooperative game, also called minimax game. In game theory, the GAN model converges when the discriminator and the generator reach a Nash equilibrium.

$$loss_D = -\log(D(x)) - \log(1 - D(G(z))) \tag{2.2}$$

$$loss_G = \log(1 - D(G(z))) \tag{2.3}$$

A special form of the GAN is normally used for image generation: a deep convolutional generative adversarial network (DCGAN) [15]. It uses convolutions and transposed convolutions instead of fully connected layers.

[1] `https://sebastian.itch.io/hydraulic-erosion`

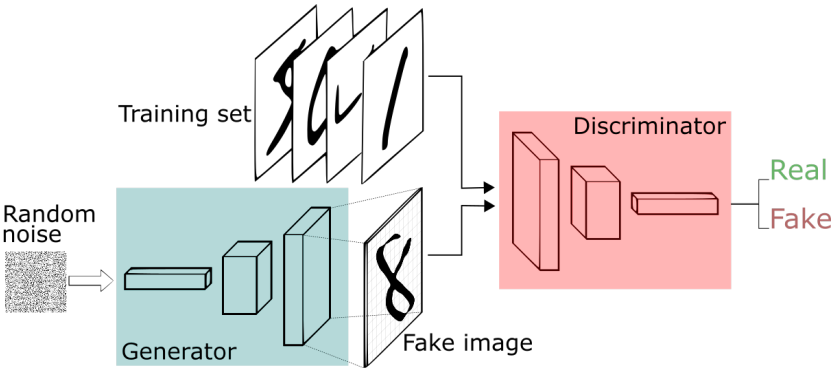



Figure 2.4.: Structure of a GAN

For the generation of 3D landscapes Christopher Beckham and Christopher Joseph Pal [3] have already established a foundation. They divided the generation into two steps. First, a heightmap is generated with a DCGAN. In a second step a picture-to-picture network [8] (also called pix2pix) is used to generate a matching texture from the heightmap.

As training data they used high-resolution terrain and heightmap data provided by the National Aeronautics and Space Administration (NASA) "Visible Earth" project[2]. This data is addressed again in chapter 3.2.1.

According to the authors the most obvious next step for future work would be to jointly train the DCGAN and pix2pix GANs for performance and quality optimizations.

## 2.4. Conclusion

In their work Beckham an Pal have achieved a reasonable first step toward procedural generation of terrain based on real-world data. A major advantage over classical approaches is that the texture of the landscape can also be generated. In addition, the new approach allows to generate much more diverse landscapes, which, however, depends strongly on the training data. But the authors also mention problems such as the need to train two models. Furthermore, landscapes are hardly scalable with the current solution, while the classical approaches make this possible, albeit with a strong increase in computing effort.

[2] https://visibleearth.nasa.gov/

# 3. Methods

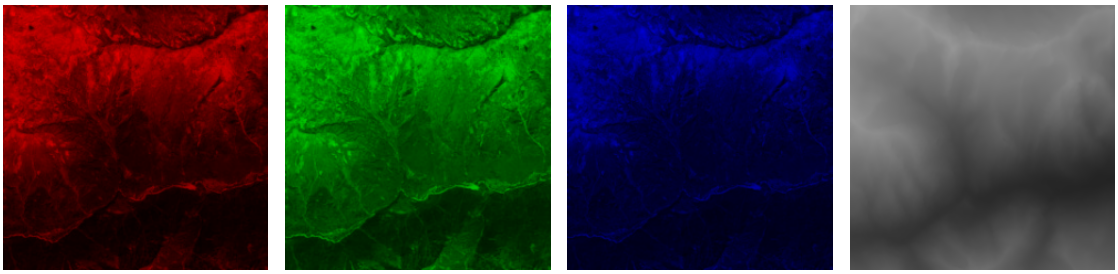
## 3.1. Concept

The basis for this study is the work of Beckham and Pal [3]. This thesis tries to solve the known problems of the original model and make the output scalable. In order not to train two models in parallel, we suggest to process the height and texture data in a single network. The height data can be considered as an additional color channel in the texture image. For this purpose, the usual RGBA format can be used, where the transparency simply represents the elevation. With this approach, the system can be reduced to a single DCGAN that uses an RGBA image as input and output.

The exact model is described in chapter 3.3 a possible method for upscaling the images is presented in chapter 3.4. To measure the quality of the generated landscapes a survey is carried out, which is described in chapter 3.5.

## 3.2. Training Data

To train the neural network, a data set must be created that consists of RGBA images containing height and texture data as in figure 3.1. The acquisition and preparation of the training data is explained below.



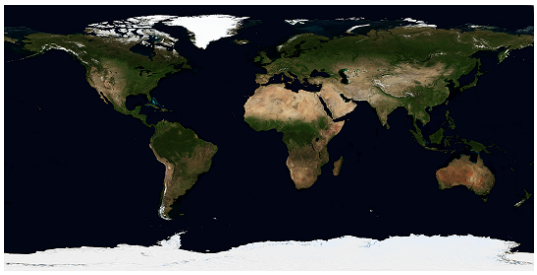
(a) red (b) green (c) blue (d) alpha

Figure 3.1.: Channels of a sample tile

### 3.2.1. Data Acquisition

The training data (figure 3.2) that Beckham and Pal [3] used are not well suited, as they only show very macroscopic landscapes. Due to the spatial resolution of 1 square km per pixel no fine details can be displayed. Furthermore the maps are only available in an equirectangular projection, which is not conformal in contrast to the Mercator projection. Thus, the further away areas are from the equator the more they are distorted.

(a) world elevation data [1]



(b) world texture data [2]

Figure 3.2.: Heightmap and texture provided by NASA's "Visible Earth" project

As an ideal data source are providers of tiled web maps, because they allow different zoom levels and therefore more details. The following providers were found who offer freely accessible height and/or texture data - albeit with a limited number of requests:

**MapTiler:** satellite imagery, RGB encoded elevation data (zoom level 0-10) [11]

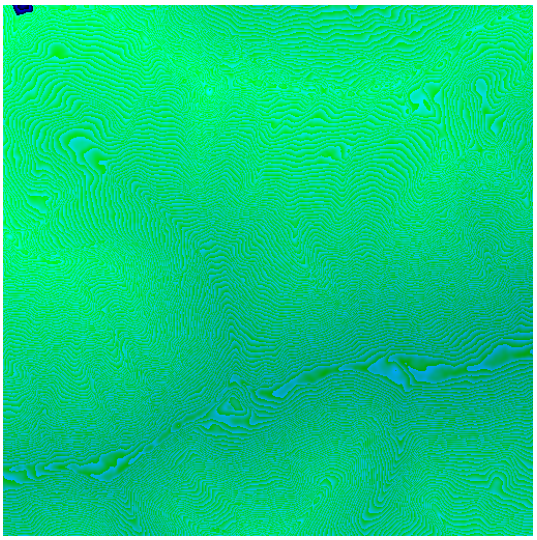
**MapBox:** satellite imagery, RGB encoded elevation data (zoom level 0-14) [10][9]

**Here:** satellite imagery [7]

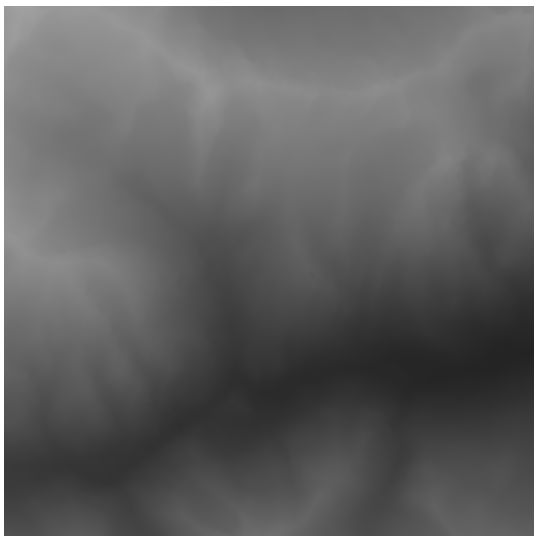
The elevation data is usually encoded with RGB values. The resulting height in meters can be calculated with the following formula: [9]

$$height = -10000 + ((R * 256 * 256 + G * 256 + B) * 0.1) \tag{3.1}$$

Figure 3.3 shows a sample heightmap from the data provided with its conversion to a grayscale image.



(a) RGB encoded

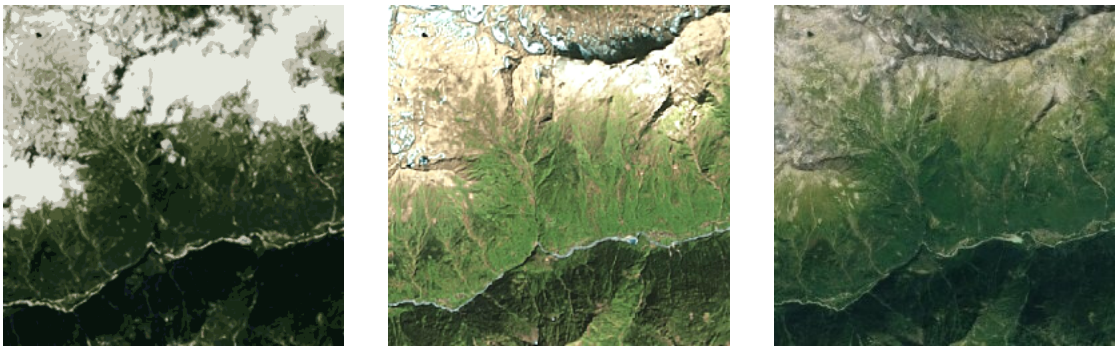


(b) converted to grayscale

Figure 3.3.: Sample heightmap

[1] https://visibleearth.nasa.gov/images/73934/topography
[2] https://visibleearth.nasa.gov/images/74092/july-blue-marble-next-generation

All three data providers use different satellite images and thus offer a varying quality. The matching textures from the tile in figure 3.3 is shown in figure 3.4. It is obvious to see, that MapBox provides the poorest quality, as the pictures have only little contrast and also show clouds. “Here” offers much more details, but the colors seem unnaturally bright and oversaturated. Also in this case some clouds are still visible. MapTiler provides the most suitable images, as the colors appear natural, the image is not blurred and no clouds are visible.



(a) MapBox (b) Here (c) MapTiler

Figure 3.4.: Satellite image quality

When selecting the areas for the training, care was taken to use similar mountain ranges. The training data consists mainly of folded mountains from the Alpide belt.

| Name | Location | Rectangle Coordinates (Latitude / Longitude) |
|---|---|---|
| Western Alps | France | (46.6/5.3), (43.95/7.9) |
| Eastern Alps | Austria / Italy | (47.9/11.4), (46.1/15) |
| Central Alps | Switzerland | (47.5/7.3), (45.9/12.1); (47.2/7), (45.6/8.5) |
| Pyrenees | France / Spain | (43/-0.9), (42.15/2.5) |
| Caucasus | Georgia / Russia | (43.9/40.3), (43.2/41.5); (43.5/41.5), (42.7/43.7); (43/43.3), (42.2/46.4) |
| Jotunheimen | Norway | (62.4/5.6), (60.5/9); (61.9/5.2), (59/7.4) |
| Southern Alps | New Zealand | (-45.2/167), (-46.1/167.4); (-44.4/167.9), (-45/169) |

Table 3.1.: Mountain ranges included in the training data

With the exception of Jotunheimen, all the regions used are so-called fold mountains. Other well-known and large mountain ranges were not included in the data set, for example:

**Northern Rocky Mountains:** The elevation data shows strong artifacts

**Himalaya:** The elevation data are far beyond the range of the other mountains in the data set. The Himalaya with altitudes up to 8848 meters would be the only mountain range in the data set with altitudes above 6000 meters.

### 3.2.2. Data Preprocessing

For the training, the raw data from chapter 3.2.1 must still be prepared. First the heightmap is converted into a grayscale image. For this, each pixel is assigned a value between 0 and 255 according to its height (from equation 3.1) with the following formula:

$$pixelvalue = \max\left(\min\left(\left\lfloor \frac{height * 255}{5000} \right\rfloor, 255\right), 0\right) \tag{3.2}$$

After texture and elevation data have been combined into one image, a data augmentation is performed. The definition of the function is shown in figure 3.5 and performs the following operations:

1. random crop (no rotation)
2. random 90 degree rotation
3. random horizontal and vertical flip

```
def augment(img):
   img = tf.image.random_crop(img, [IMG_WIDTH, IMG_HEIGHT, 4])
   img = tf.image.random_flip_left_right(img)
   img = tf.image.random_flip_up_down(img)
   img = tf.image.rot90(img, tf.random.uniform(shape=[], minval=0, maxval=4,
       dtype=tf.int32))
   return img
```

Figure 3.5.: Definition of the data augmentation using Tensorflow

## 3.3. Models

The base model is a DCGAN. The following definitions are based on the recommendations of Radford et al. [15] and Beckham et al. [3].

The generator consists of six upsampling stages. One stage consists of:

1. transposed convolution to upsample by the factor 2
2. batch normalization
3. ReLU activation layer

The complete definition of the generator is given in figure 3.6.

```python
def gan_g_layer(model, nFilter, resizeConv=False):
if resizeConv:
model.add(layers.UpSampling2D(size=(2, 2), interpolation='bilinear'))
model.add(layers.Conv2D(nFilter, (5, 5), strides=(1, 1), padding='same'))
else:
model.add(layers.Conv2DTranspose(nFilter, (4,4), strides=(2, 2), padding='same',
    use_bias=False))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())

def make_generator_model():
model = tf.keras.Sequential()
model.add(layers.Dense(4*4*512, use_bias=False, input_shape=(noise_dim,)))
model.add(layers.BatchNormalization())
model.add(layers.ReLU())
model.add(layers.Reshape((4, 4, 512)))

gan_g_layer(model, 512)
gan_g_layer(model, 512)
gan_g_layer(model, 256)
gan_g_layer(model, 256)
gan_g_layer(model, 128)
gan_g_layer(model, 128)

model.add(layers.Conv2D(4, (5, 5), strides=(1, 1), padding='same', use_bias=False,
    activation='tanh'))
assert model.output_shape == (None, IMG_WIDTH, IMG_HEIGHT, 4)

return model
```

Figure 3.6.: Definition of the generator in python using Keras

The discriminator consists of five downsampling stages. One stage consists of:

1. convolution to downsample by the factor 2
2. batch normalization
3. leaky ReLU activation layer
4. dropout with a rate of 30%

```python
def gan_d_layer(model, nFilter):
model.add(layers.Conv2D(nFilter, (5, 5), strides=(2, 2), padding='same'))
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU())
model.add(layers.Dropout(0.3))

def make_discriminator_model(channels):
model = tf.keras.Sequential()
model.add(layers.Conv2D(32, (5, 5), strides=(2, 2), padding='same',
    input_shape=[IMG_WIDTH, IMG_HEIGHT, channels]))
model.add(layers.LeakyReLU())
model.add(layers.Dropout(0.3))

gan_d_layer(model, 64)
gan_d_layer(model, 128)
gan_d_layer(model, 256)
gan_d_layer(model, 512)
gan_d_layer(model, 512)

model.add(layers.Flatten())
model.add(layers.Dense(1, activation='sigmoid'))

return model
```

Figure 3.7.: Definition of the discriminator in python using Keras

The overall model can be represented as follows:

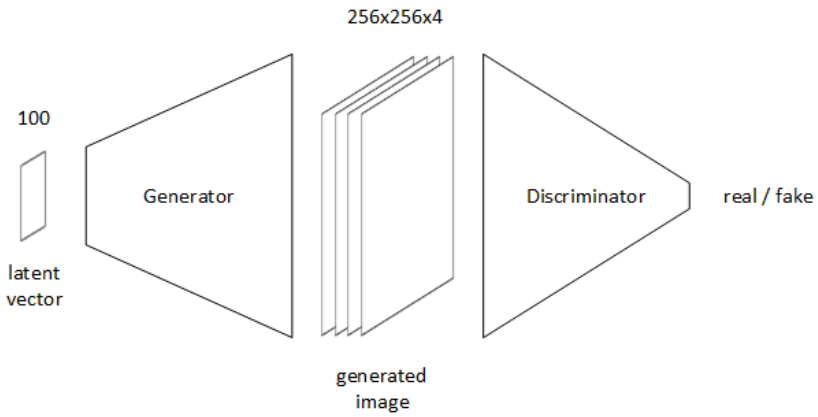



Figure 3.8.: Simple GAN model

During the test phase of this model the problem arose that the heightmap was always very blurry, so another model was defined. This advanced model includes a second discriminator which only evaluates the heightmap. Both discriminators together determine the loss of the generator in equal parts.

In contrast to the simple model, the heightmap now receives a higher weighting, because otherwise it would be underrepresented with only one channel compared to three channels for the texture.

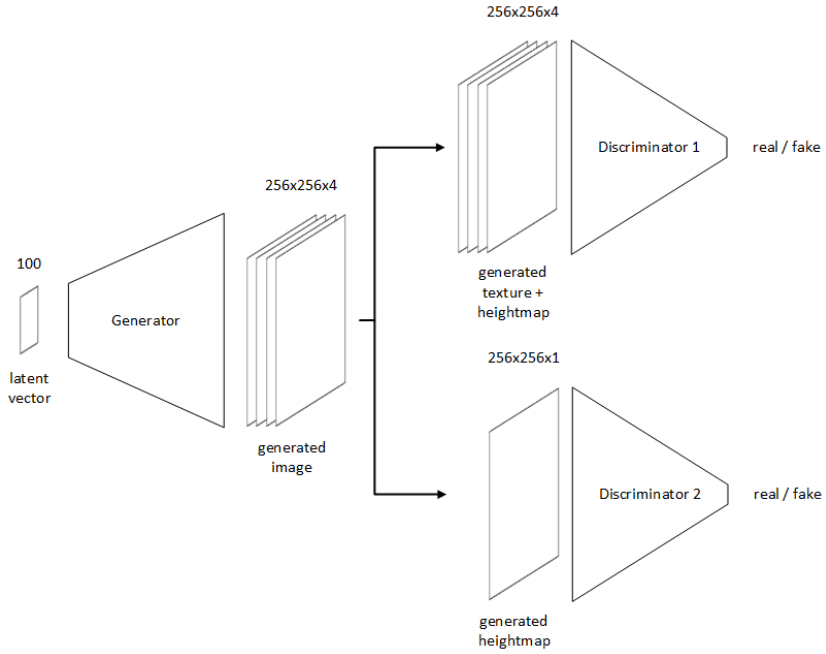



Figure 3.9.: Advanced GAN model with two discriminators

## 3.4. Tile composition

To combine several tiles and create a larger map, we can use existing algorithms. Since our model is only a single GAN, image painting methods from Yeh et al. [18] can be applied. They formulate an optimization problem in which the latent vector z is adjusted in such a way that the loss of the generated image is as small as possible.

$$\hat{z} = \arg\min_{x} L(z) \tag{3.3}$$

The loss is composed of:

**Contextual loss:** distance between known parts of image and reconstruction

**Perceptual loss:** log likelihood of $G(z)$ not being real according to D

$$L(z) = L_{contextual}(z) + \lambda * L_{perceptual}(z) \tag{3.4}$$

$$L_{contextual}(z) = (M \odot G(z) - M \odot x_{corrupted})^2 \tag{3.5}$$

$$L_{perceptual}(z) = \log(1 - D(G(z))) \tag{3.6}$$

The reconstructed image can be found as

$$x_{reconstructed} = M \odot x_{corrupted} + (1 - M) \odot G(z) \tag{3.7}$$

where $M$ is a binary mask (same size as $x_{corrupted}$, 0 for missing/corrupted pixels) and $\odot$ is the element wise product of two matrices).

For the application to our case only the contextual loss has to be reformulated. The optimizer has to find matching height profiles and textures along one or two edges. The loss is the

distance between the pixels of the overlapping region of two adjacent tiles (marked as purple in figure 3.10)

$$L_{contextual}(z_{m+1,n}) = (RightBorder(G(z_{m,n})) - LeftBorder(G(z_{m+1,n})))^2 \tag{3.8}$$

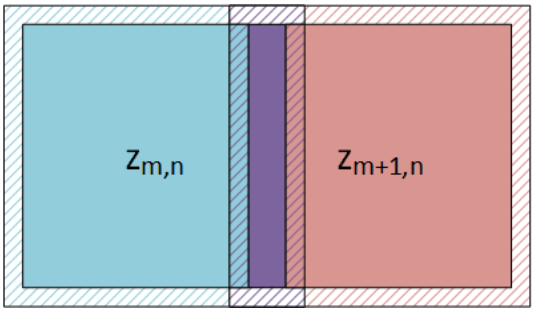



Figure 3.10.: Overlapping tiles

If a tile has several neighbors, the loss function is extended to the corresponding edges.

$$\begin{aligned} L_{contextual}(z_{m+1,n+1}) = (RightBorder(G(z_{m,n+1})) - LeftBorder(G(z_{m+1,n+})))^2 + \\ (LowerBorder(G(z_{m+1,n})) - UpperBorder(G(z_{m+1,n+})))^2 \end{aligned} \tag{3.9}$$

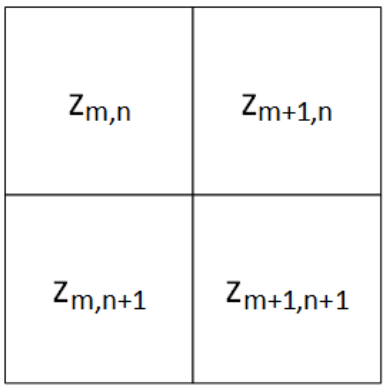



Figure 3.11.: Composed tiles

## 3.5. Quality evaluation

In order to measure the quality of the generated landscapes, a survey is carried out in which the probands are asked to distinguish between real and generated landscapes. The survey consists of two parts:

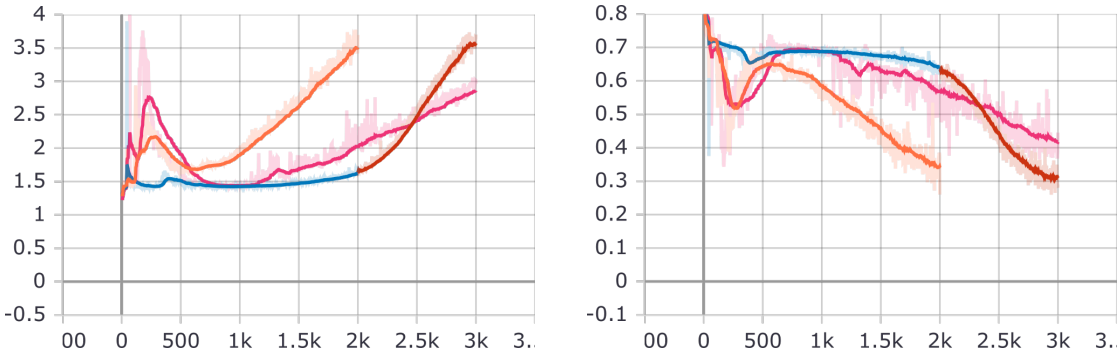
1. single tiles
2. composed tiles using the procedure from chapter 3.4

In both parts, the participants have to assess 50 landscapes each, half of which are generated. In order to avoid biases as much as possible, the probands are not told how many of the given images are generated. Furthermore, the images are presented to each subject in random order.

# 4. Results

## 4.1. Training

For training the Adam optimizer was always used with an learning rate (LR) of 1e−5, if not stated otherwise. The results of the simple model are not listed, as it was replaced by the more advanced model with two discriminators during the course of the work. The loss curves of the best models are listed in figure 4.1. It should be noted that, for simplicity, only the loss from the first discriminator is shown, since both are practically identical. The legend of the graph is listed in table 4.1.




(a) generator loss

(b) discriminator 1 loss

Figure 4.1.: Loss curves

| Color | Generator config | $LR_G$ | # Generator Params | Latent space size |
|---|---|---|---|---|
| Orange | 512-256-256-128-128-64 | 0.0001 | 9,323,584 | 100 |
| Blue | 512-512-256-256-128-12 | 0.0002 | 13,172,736 | 100 |
| Purple | 512-512-256-256-128-128 | 0.0002 | 14,450,688 | 256 |

Table 4.1.: Best performing models

The specified configuration refers to the depth of the respective convolutional layer.

## 4.2. Survey Results

The results of the survey are listed and interpreted below. A total of 120 people took part in the survey. The images to be evaluated are all derived from the most recent and largest model (purple model in table 4.1).

In the optimal case there would be an overall accuracy (= proportion of correctly identified images) of 50%. This would mean that the images are not distinguishable from each other and on average every second image is correctly guessed.

Another important measure is the false positive rate, which shows how many of the generated images were considered real. In the best case this would be 100%.

In the following chapters the samples are presented as in figure 4.2. The corresponding percentage always corresponds to the percentage of subjects who rated the corresponding landscape as real.

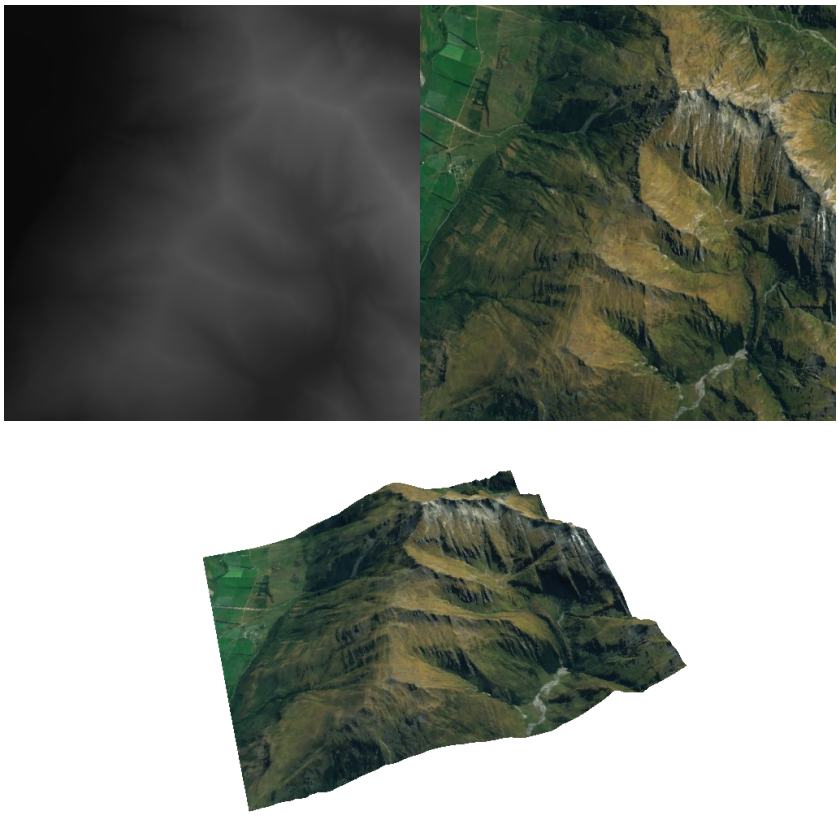
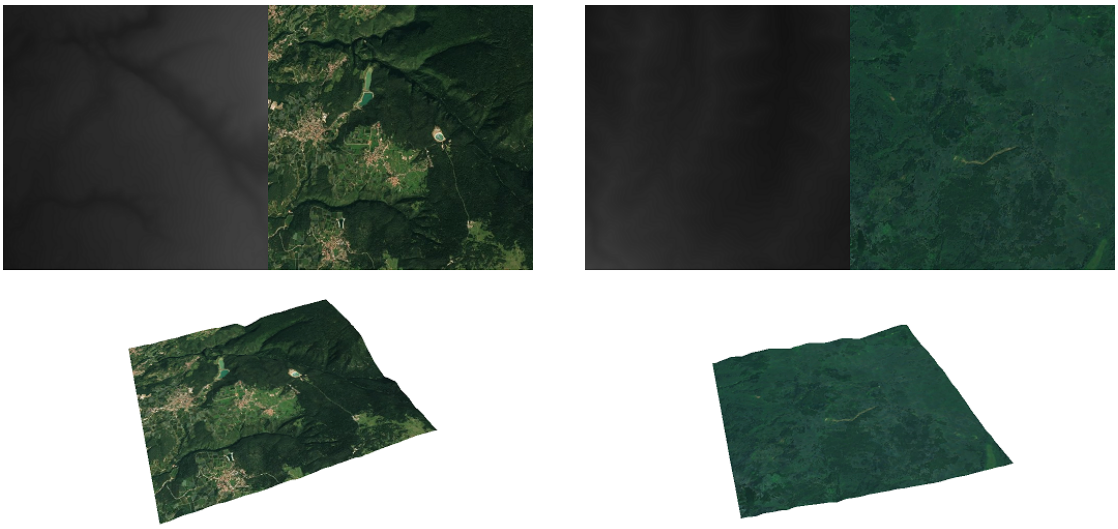


Figure 4.2.: Sample output image for the survey with heightmap (top-left), texture (top-right) and 3D rendered mesh (bottom), rating = 62.5%

### 4.2.1. Single Tiles

In the first part of the survey the quality of the individual tiles was examined. The test persons achieved an overall accuracy of 58.9%. This means that in many cases they had great difficulty in distinguishing the images. The false positive rate is 44.7%. Thus, almost every second generated tile was considered real. The very low accuracy also shows that many real tiles were rated as fake.

Figures 4.3 and 4.4 show the best and worst rated samples of the generated and real tiles.

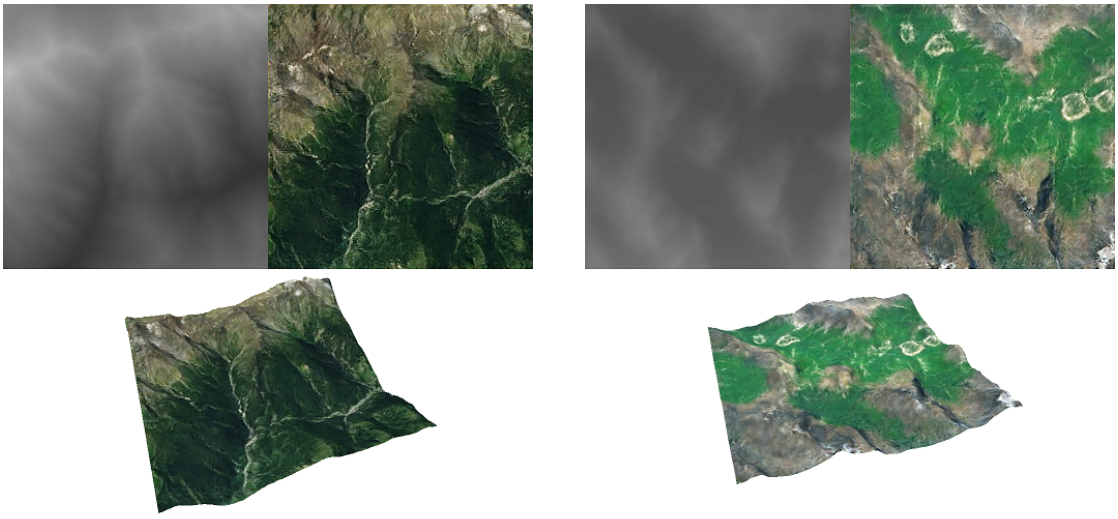


(a) best rated (81.7%)



(b) worst rated (32.5%)

Figure 4.3.: Real world sample



(a) best rated (76.7%)



(b) worst rated (13.3%)

Figure 4.4.: Generated sample

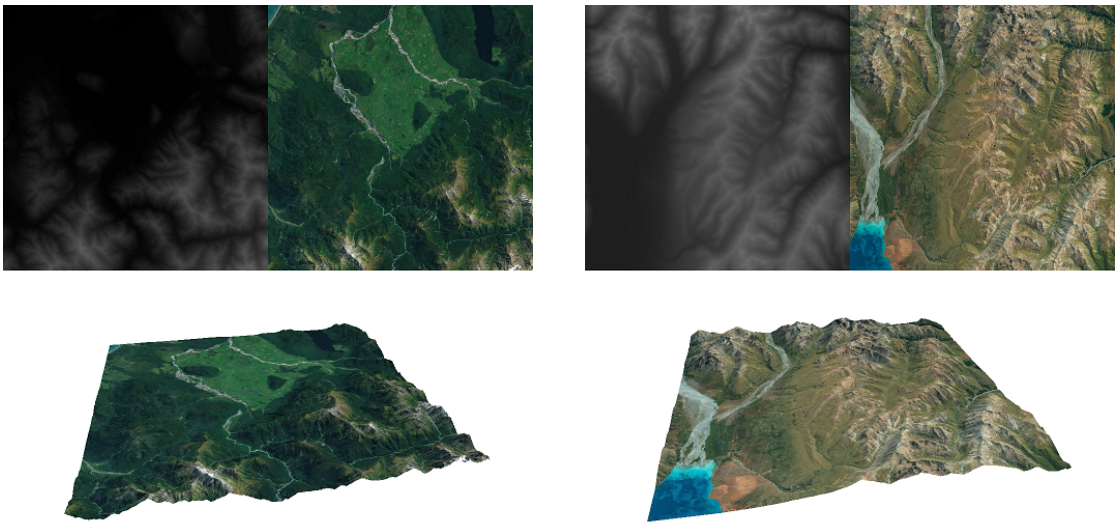
Some generated tiles show artifacts like repetitive patterns. Some test persons have also noticed the generally poorer quality (e.g. blurry). On closer examination, a few probands also noticed illogical topographies or wrong shadow casts.

### 4.2.2. Composed Tiles

In the second part of the survey the quality of the composed tiles was examined. The test persons achieved an overall accuracy of 69.7%. This means that in many cases they had only little difficulty in distinguishing the images. The false positive rate is 29.7%. Thus, barely every third

generated tile was considered real.

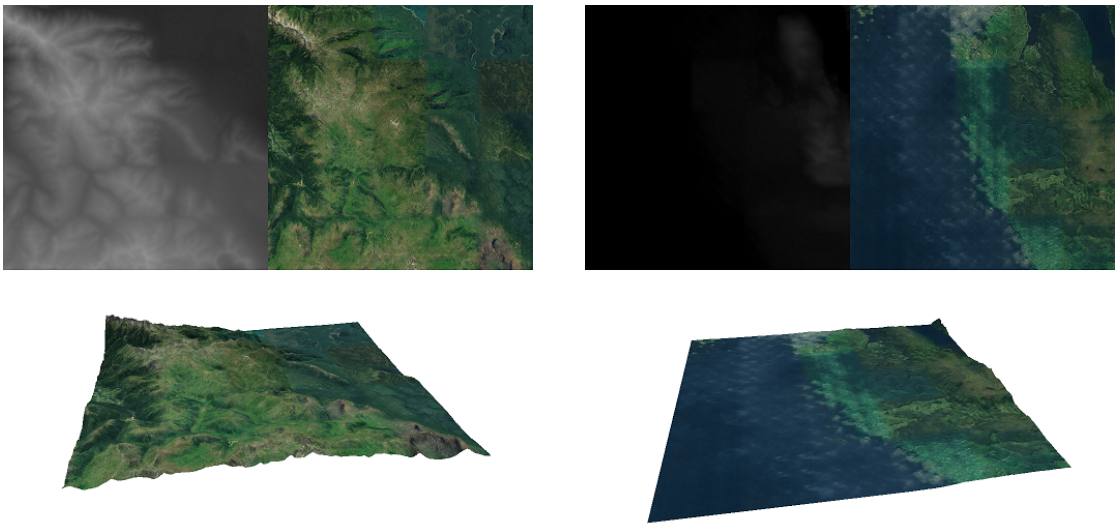
Figures 4.5 and 4.6 show the best and worst rated samples of the generated and real tiles.



(a) best rated (85%)

(b) worst rated (43.3%)

Figure 4.5.: Real world sample



(a) best rated (50.8%)

(b) worst rated (9.2%)

Figure 4.6.: Generated sample

Most criticised were the visible borders of the composite tiles. Often the colors do not match either. Figure 4.7 shows a map where both flaws are strongly visible.

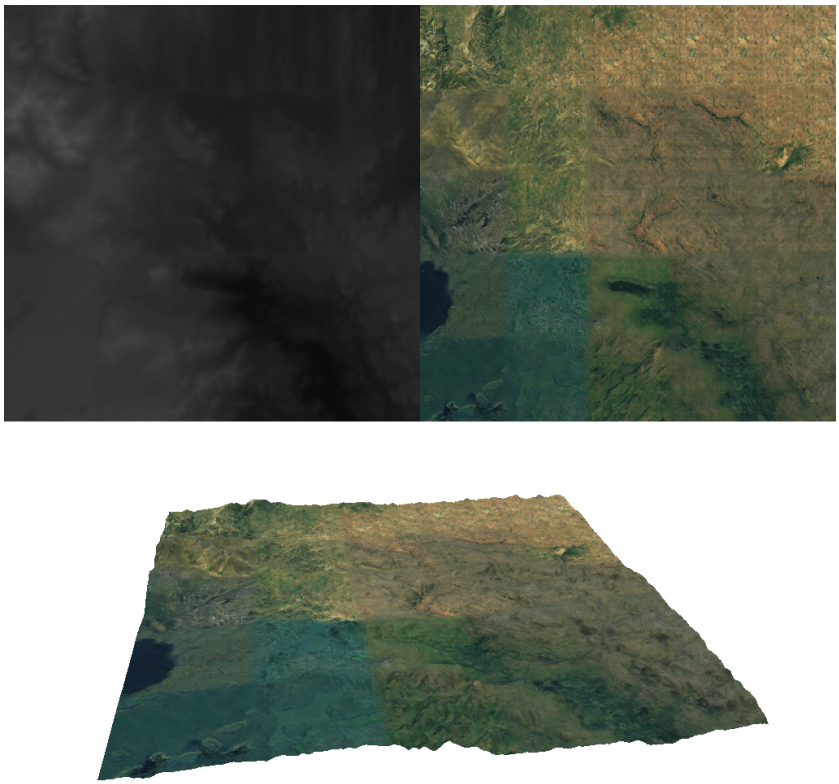


Figure 4.7.: Badly rated (17.5%) map with strongly visible edges and unfitting color hues

# 5. Discussion

## 5.1. Evaluation

The results from chapter 4 show that the model is quite capable of producing realistic landscapes. Even if only 45% of the tiles were found to be real, some of them could fool up to 77% of the test persons. It is interesting to note that several real pictures were also classified as false. These are rather flat and barren landscapes. From this it could be deduced that for humans diversified landscapes are generally more realistic.

The composite tiles perform much worse, more than two thirds of the images were correctly classified. The generated images still show many artifacts and the composing algorithm has problems to follow all restrictions (matching height profiles and texture along two edges). Either the search space is too small or the optimizer gets stuck too early in a local optimum.

In summary, it can be said that the new approach allows to generate much more diverse landscapes, which, however, depends strongly on the training data. In principle, it would be possible to combine several tiles with the proposed algorithm, but it works very unreliably.

## 5.2. Problems

The biggest problem is the compositional algorithm. It rarely manages to match texture and height profile on all edges. Different approaches like a larger latent space did not show any effect.

The choice of training dates may also be disadvantageous, as the tiles may have very different color shades, but not all of them are equally frequent in the dataset.

## 5.3. Future Work

The most obvious point to work on in the future is the composing algorithm. It must be able to work more reliably. Maybe the whole model has to be adapted and conditional GANs or similar have to be used.

Another aspect is the training data. These should be more diversified so that the model has enough freedom to find a suitable neighbour for each tile.

# Bibliography

[1] National Geospatial-Intelligence Agency. Wgs 84 and the web mercator projection nga office of geomatics. https://earth-info.nga.mil/GandG/wgs84/web_mercator/%28U%29%20NGA_SIG_0011_1.0.0_WEBMERC.pdf, 2014. [online; accessed on 15.04.2020].

[2] National Geospatial-Intelligence Agency. Nga geomatics - wgs 84. https://earth-info.nga.mil/GandG/update/index.php?dir=wgs84&action=wgs84, 2020. [online; accessed on 15.04.2020].

[3] Christopher Beckham and Christopher Joseph Pal. A step towards procedural terrain generation with gans. *ArXiv*, abs/1707.03383, 2017.

[4] Jaydeep T. Chauhan. Comparative study of gan and vae. *International Journal of Computer Applications*, 182(22):1–5, Oct 2018.

[5] Alain Fournier, Don Fussell, and Loren Carpenter. Computer rendering of stochastic models. *Commun. ACM*, 25(6):371–384, June 1982.

[6] Stefan Gustavson. Simplex noise demystified. url: http://staffwww.itn.liu.se/ stegu/simplexnoise/simplexnoise.pdf, März 2005.

[7] HERE. Guide - map tile api. https://developer.here.com/documentation/map-tile/dev_guide/topics/resource-base-maptile.html, 2020. [online; accessed on 27.02.2020].

[8] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2016.

[9] Mapbox. Access elevation data. https://docs.mapbox.com/help/troubleshooting/access-elevation-data/, 2020. [online; accessed on 27.02.2020].

[10] Mapbox. Maps service api. https://docs.mapbox.com/api/maps/#raster-tiles, 2020. [online; accessed on 27.02.2020].

[11] MapTiler. Maptiler cloud tiles. https://cloud.maptiler.com/tiles/, 2020. [online; accessed on 27.02.2020].

[12] MapTiler. Tiles à la google maps. https://www.maptiler.com/google-maps-coordinates-tile-bounds-projection/, 2020. [online; accessed on 06.03.2020].

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

[14] Ken Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3):287–296, July 1985.

[15] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

[16] OpenStreetMap Wiki. Slippy map tilenames. https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames, 2020. [online; accessed on 07.03.2020].

[17] Georgios N. Yannakakis and Julian Togelius. *Artificial Intelligence and Games*. Springer, 2018. `http://gameaibook.org`.

[18] Raymond Yeh, Chen Chen, Teck Lim, Mark Hasegawa-Johnson, and Minh Do. Semantic image inpainting with perceptual and contextual losses. 07 2016.

# List of Figures

1.1. Different zoom levels of a tiled map, represented as a pyramid . . . . . 2

2.1. Conversion of the Simplex-Noise to an $\frac{1}{f}$-noise . . . . . 5
2.2. Procedure of the Diamond-Square-Algorithm . . . . . 5
2.3. Effect of a simulated erosion . . . . . 6
2.4. Structure of a GAN . . . . . 7

3.1. Channels of a sample tile . . . . . 8
3.2. Heightmap and texture provided by NASA's "Visible Earth" project . . . . . 9
3.3. Sample heightmap . . . . . 9
3.4. Satellite image quality . . . . . 10
3.5. Definition of the data augmentation using Tensorflow . . . . . 11
3.6. Definition of the generator in python using Keras . . . . . 12
3.7. Definition of the discriminator in python using Keras . . . . . 13
3.8. Simple GAN model . . . . . 13
3.9. Advanced GAN model with two discriminators . . . . . 14
3.10. Overlapping tiles . . . . . 15
3.11. Composed tiles . . . . . 15

4.1. Loss curves . . . . . 16
4.2. Sample output image for the survey with heightmap (top-left), texture (top-right) and 3D rendered mesh (bottom), rating = 62.5% . . . . . 17
4.3. Real world sample . . . . . 18
4.4. Generated sample . . . . . 18
4.5. Real world sample . . . . . 19
4.6. Generated sample . . . . . 19
4.7. Badly rated (17.5%) map with strongly visible edges and unfitting color hues . . 20

# List of Tables

3.1. Mountain ranges included in the training data . . . . . 10

4.1. Best performing models . . . . . 16

# A. Appendix

## A.1. Used software

### A.1.1. Modeling and Training

`https://github.com/Major94/Terrain_Generation_GAN`

Tensorflow and Keras using Jupyter Notebooks on Python

1. Model definition
2. Model training
3. complete dataset

### A.1.2. Data Acquisition

`https://github.com/Major94/TerrainGeneration`

Windows Forms Application on C#

1. image download
2. image conversion

### A.1.3. Data Visualization

`https://github.com/Major94/3D_Terrain`

Unity 3D Engine on C#

1. 3D mesh render from grayscale or RGB heightmap
2. create tile preview with heightmap, texture und 3D mesh

# A.2. Curriculum vitae

## Personal Data


| | |
|---|---|
| Name: | Manuel Jordan |
| Place and Date of Birth: | Brig, Switzerland \| 13 December 1994 |
| Address: | Napoleonstrasse 64, 3902 Glis |
| email: | manuel.jordan@gmx.net |


## Work Experience

| | |
|---|---|
| 2018 - 2019 | ABB Switzerland AG, Lenzburg<br>PLC programming<br>Version control concept for pLC software |
| 2017 | Maxon Motor AG, Sachseln<br>Conception of voltage control for electromotors<br>Bachelor Thesis |
| 2017 | Mimedis AG, Muttenz<br>Development of an application for document generation |

## Scientific Education

| | |
|---|---|
| 2018 - 2020 | Master of Science in Engineering<br>Lucerne University of Applied Sciences and Arts<br>Master thesis: Landscape Generation with Generative Adversarial Networks |
| 2014 - 2018 | Bachelor of Science FHNW in Mechatronics Trinational<br>FHNW University of Applied Sciences and Arts Northwestern Switzerland |
| 2010 - 2014 | Automatiker EFZ<br>Ecole des métiers du Valais, Sion |