MASTERPROJEKT MUSTERERKENNUNG
TERRAIN GENERATION WITH GENERATIVE ADVERSARIAL NETWORKS

Denise Moussa, ca97siwo, 21954294

1 Motivation

Terrain generation plays an important role in different types of computer games but also VR worlds or simulations (e.g. flight simulators). It should be both realistic and provide a certain degree of diversity.

At the moment, there are basically two main procedures to generate terrains that can also be combined. One option is to use maps that are designed by artists and/or developers in advance, for which a great amount of time and human resources may be needed. On the other hand, maps can be generated automatically by an algorithm.

Then, two problems have to be solved. First, a height map is needed that defines the elevation of the terrain. This is mostly done on a basis of different noise functions, like perlin or fractal noise. The well known game *Minecraft* serves as an example for this kind of generation [1]. To provide a short insight into the principle, look at Figure 1 as an example. Here, differently scaled 2D perlin noise maps were added up to form an elevation map that can be rendered as a mesh forming the terrain.

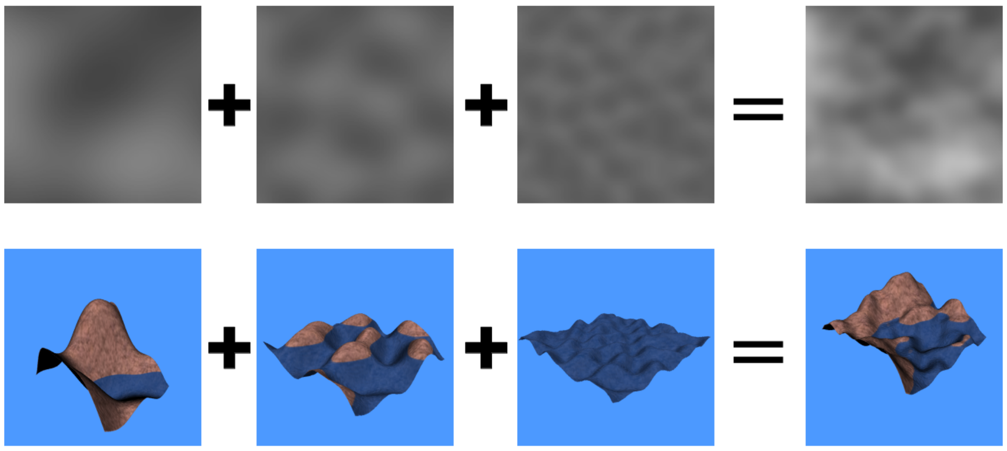


Figure 1: Artificially generated height maps and the corresponding results using a simple terrain renderer (own implementation)

Note that the texture was added later and in fact shows the second problem of terrain generation: defining it's landscape, like for example the location of forests,

[1]https://minecraft-de.gamepedia.com/Weltgenerierungext

grasslands, lakes and many more. This can again be done by custom design or be calculated artificially. When computed, the task is often solved by using a set of biomes representing different landscapes. These biomes can for example be defined by textures and forms. To generate the final map, the biomes are blended together using a set of rules. For example, one may define a constraint forbidding a region of sand directly next to a glacier. For more information about terrain generation with biomes, take a look at [2]. Again,*Minecraft* serves as an example here, as it provides this possibility of generating maps defining the game's landscape.[1]

A drawback of this method is that one has to define a possibly big set of rules in order to describe a terrain in a realistic fashion, since principles of the real world, like climate, timberlines, prevailing winds and many more have to be taken into account. Similarly, the approach of using noise functions for height map generation is limited in producing realistic terrains, since it cannot take into account geological processes. Of course, the algorithms can be extended arbitrarily, to simulate erosion and other natural processes [7].

However, we have to take into account that when performance is an important factor, at least resources consuming, complex simulation processes are not possible, as should be the case for most computer games.

2 Task and Problem Formulation

The aim of this project is to avoid complex simulation algorithms for the generation of realistic terrains by applying the technique of deep learning to the task. To reach this aim, a generative adversarial network shall be designed to generate plausible terrain data with both elevation and landscape information per pixel. As an example, the output of the generator could give us a height map that defines the location of mountains, valleys or flat areas, but also provides the structure of the terrain like regions of rock, grassland or water and the information on where those regions are located. Formally, our task can be described as following:

We draw a k-dimensional sample $\tilde{z} \sim p(z)$ from a noise distribution as input for our neural network denoted as $N(\tilde{z})$. We then obtain our result $\tilde{x} = N(\tilde{z})$ which is a composition of the desired outputs height map h and label map l. Defining the generator of our DCGAN as $G(\cdot)$ and our discriminator as $D(\cdot)$ our goal of training can be defined as the following minimization problem:

$$min_G = L(D(G(\tilde{z})), c_r) \tag{1}$$

$$min_D = L(D(x), c_r) + L(D(G(\tilde{z})), c_f) \tag{2}$$

L is a GAN specific loss function which is in our case the standard binary cross-entropy. c_r denotes the label for real samples, where we used 0.9 (label smoothing) and c_f the label for faked samples (outputs of the generator), in our case 0. x describes a real sample of the training set.

3 Related Work

[1] use openly available satellite imagery from NASA that depict height map and texture data in order to train two generative adversarial networks (see Figure 2). The first network is a DCGAN which generates height maps from noise and provides the input for the second GAN, a 'pix2pix' GAN [4], which translates these height maps to texture maps. [1] claim that this configuration is able to produce relatively faithful elevation maps that are similar to the original data, despite some training instabilities. The texture maps generated by the image-to-image translation GAN further show different textural properties. Especially, regions of relatively higher elevation seem to have different textures. However, they also note some side effects that seem to appear because the two GANs are not trained jointly. First of all, some textures are completely white. Secondly, textures generally seem to differentiate faithfully to relative heights but not to absolute heights. For example, when generating deserts the pix2pix GAN still assigns higher elevations a snow texture (See Figure 3.).

Contrary to [1], we only train one GAN on both height data and the corresponding label data as joint input. The idea is that the neural network may be able to learn rules or correlations between certain height and landscape occurrences that way. As suggested by the name, we also do not use continuous texture data but discrete label data that assigns each pixel in the elevation map to one class (e.g. grass, snow, etc.). This is due to the fact that our output shall serve as so called *splatmaps* for rendering, meaning that there are clear instructions (discrete

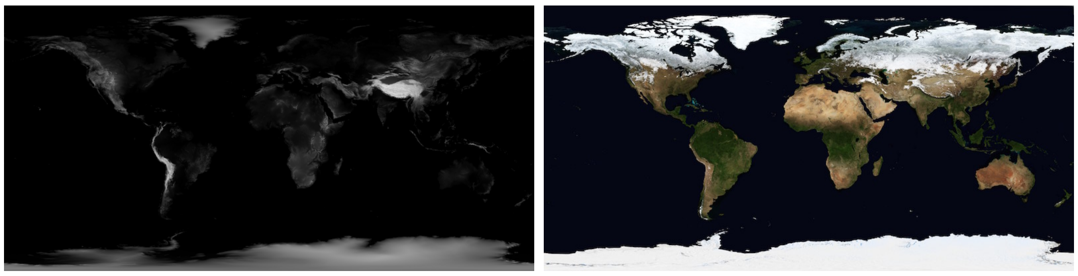


Figure 2: Height map (left) and texture map (right), both with dimensions 21600px x 10800px, provided by the NASA Visible Earth project with a spatial resolution of 1 square km per pixel [1].

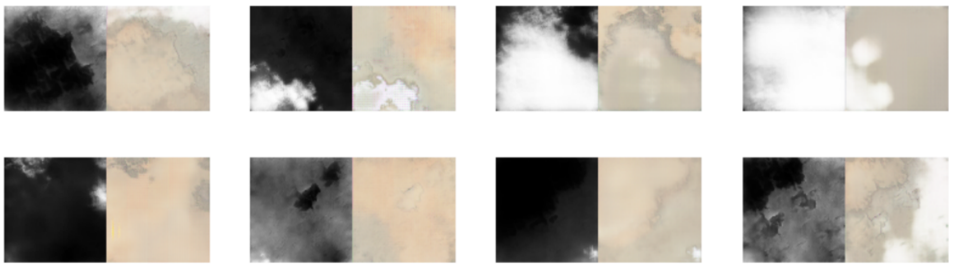


Figure 3: An assembly of results from [1]. The examples portray desert regions with height maps (left) and coresponding texture maps (right). The 'pix2pix' GAN predicts snowy textures for higher elevations even in desert regions.

labels per pixel) where to render forests, deserts and other biomes. Furthermore, we decided to train on data with higher spatial resolution and in case of elevation data, higher height value resolution than [1]. Details about our training data are presented in the next section.

4 Training Data

For training, we use datasets provided by the service *Mapbox* [2], which offers free access to map data to a certain degree and allows for custom map design that enabled us to design training samples fit for our task.

[2]www.mapbox.com

4.1 Elevation Data

Mapbox provides elevation data as raster tiles that encode height values in the Red, Green and Blue channels of PNG files. The data set, called *Terrain-RGB*, uses each color channel as a position in a base-256 numbering system and therefore is able to store 16777216 different elevation values. These are further mapped to 0.1 meter height increments as shown by equation 3.[3]

$$\text{height\_value} = -10000 + ((R \cdot 256^2 + G \cdot 256 + B) \cdot 0.1) \tag{3}$$

Elevation data can be accessed at different zoom levels, specifying the resolution. For this task, data was requested at a zoom level of 11, which results in a resolution of circa 76 meters per pixels for equator. For a standard screen with 90dpi and zoom level 11, 1cm corresponds to 2.7 km in reality[4].

Using the RGB-Data directly as input for the GAN proved to be problematic due to the encoding of the actual height values as a base-256 number. Minor noise in the prediction of the R channel, for example, results in huge height differences. However, when training on the real height data we encountered the problem of floating point errors, because the images are getting normalized to the range $[0, 1]$, which is not suitable for 16777216 different height values. Since we noted, that the R channel of our training set is 1 for all samples, we decided to only take note of the G and B channels. As a result, the height values are more equally distributed in the value range and we only have to consider 65536 different possible values. Equation 4 shows the simple preprocessing step, we used for each RGB-Sample.

$$\text{normalized\_training\_height} = \frac{G \cdot 256 + B - 32767.5}{32767.5} \tag{4}$$

4.2 Label Data

Since it is the aim of our task to learn both the elevation of a terrain and its properties, we designed a data set that assigns each pixel of the RGB raster tiles a label. With the help of *mapbox studio* [5], one can design own map styles via editor

[3]https://docs.mapbox.com/help/how-mapbox-works/mapbox-data/#mapbox-terrain-rgb
[4]https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames#Resolution_and_Scale
[5]https://studio.mapbox.com/

and later request terrain raster tiles using that style. We decided upon twelve different classes that a pixel can be exclusively assigned to. The classes were chosen based on the data *Mapbox* provided for describing the landscape of the terrain data set (see table 1) [6]. The twelve labels are encoded as equidistantly distributed gray levels $\in [0, 255]$ in one channeled PNG raster tiles.

gray level	class
0	agriculture
23	grass
46	scrub
69	glacier
92	wood
115	rock
138	sand
161	park
184	water
207	hillshade
230	landcover
252	background

Table 1: Classes one pixel can exclusively belong to

4.3 Training Samples

Using the Mapbox-API, a data set of 249.066 training samples was obtained. We focused on samples of the regions Alps, Pyrenees and Scandinavian Mountains in order to obtain images with great diversity in height values. One sample consists of one elevation tile with dimensions 64x64x3 and the corresponding label tile (64x64x1). See Figure 4 for some examples. For training, the elevation data has to be extracted from the RGB-channels as mentioned in Section 4.1. Elevation and their corresponding height samples are then stacked to form the resulting input of the dimension 64x64x2 for the neural network.

[6]https://docs.mapbox.com/vector-tiles/reference/

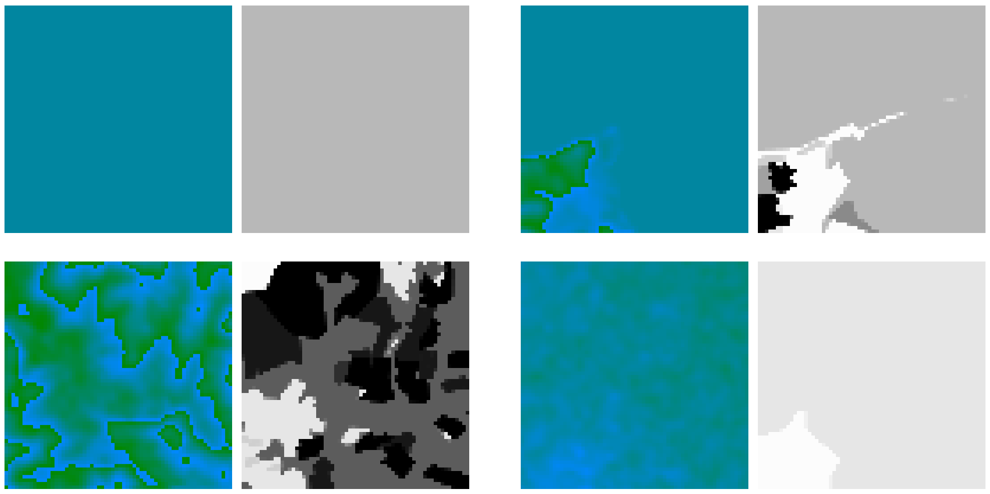


Figure 4: Four training samples, each consisting of an elevation tile (left ones) and label tile (right ones).

5 Architecture

For this task, we coarsely followed the principle of the DCGAN architecture [8]. See Figure 5 for a detailed visualization of our model. Note that the noise seed taken as input by the generator is sampled from a normal distribution $\mathcal{N}(0, 1)$. As dimension of the convolutional filters (5x5) was chosen for all convolutional layers. Since we encountered noisy results when generating elevation data, we exchanged each transposed convolutional layer by an upsampling layer followed by a convolutional layer as proposed by [6], which greatly improved the quality of the resulting predicted images. As proposed by [8], each convolutional layer in the generator is followed by batch normalization The generator and discriminator both use Leaky ReLu as activation function for their layers and in addition the discriminator uses Dropout with factor 0.3 after each layer. Training is performed with a batchsize of 128. Both the generator and discriminator use a learning rate of 0.0001 with the optimizer *Adam* [5] and its default parameters ($\beta_1 = 0.9, \beta_2 = 0.999$). We also tried *RMSProp* [9] as alternative to the momentum based optimizer. However, for our task, *Adam* led to faster convergence and better results. Furthermore, we applied one sided label smoothing, assigning the real (not generated) samples a value of 0.9 instead of 1.

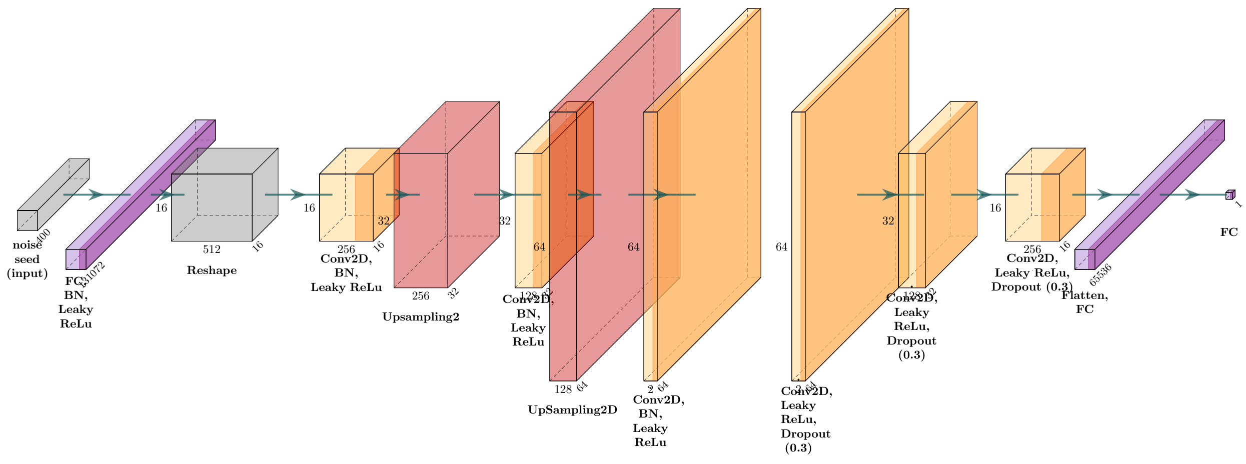


Figure 5: Visualization of GAN architecture

We further experimented with the wasserstein loss with gradient penalty [3] as alternative loss function to the binary cross-entropy. However, this function proved to be not suitable for our task. We were not able to generate convincing label data using it.

6 Results

Figure 6 depicts gray scale label maps and their corresponding elevation maps as generated by the neural network. For the human eye, the results seem plausible compared to the original training data. However, noise can be noticed in both the label and elevation data which is a common problem in GAN architectures.

When using the label data as input information, it is further necessary to apply thresholding in order to identify the correct label per pixel. During prediction, we just map each predicted value v $\in [0, 1]$ to an integer $[0, 255]$, so some values can vary from the defined 12 labels due to noise or prediction errors. Therefore, thresholding is necessary to force all values to be in the set of the previous defined 12 label values. Figure 7 shows the result of first applying a median kernel of size 3 and then processing the result with a simple threshold segmentation.We took the mean of each pair of adjoining label values in the range $[0, 255]$ as boundary for twelve label intervals. In order to provide a more intuitive visualization, Figure 7

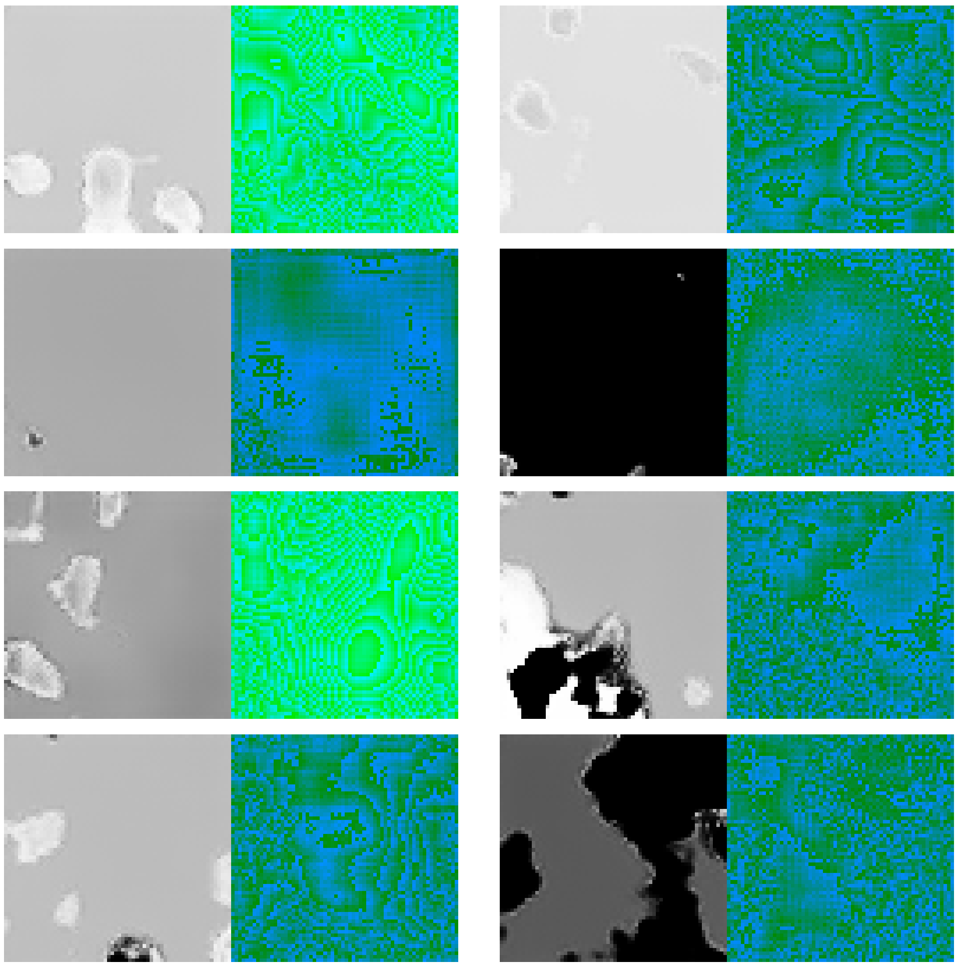


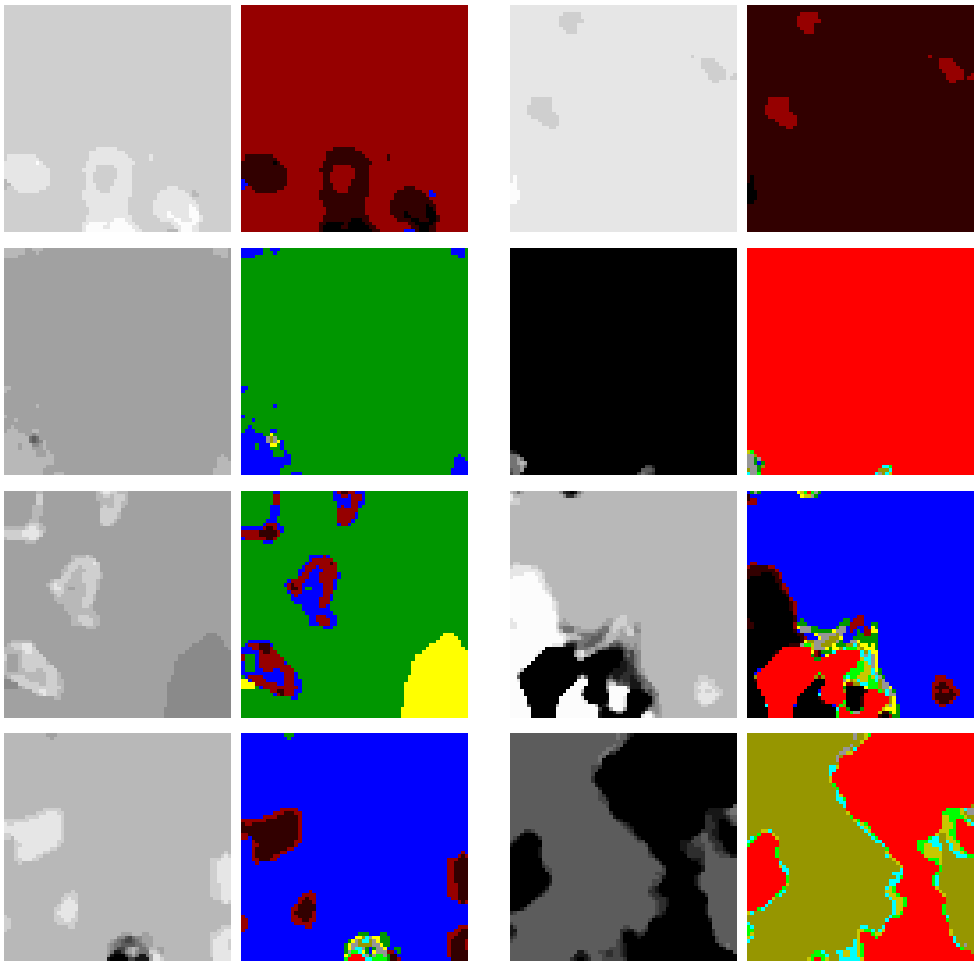
Figure 6: Some examples of generated image pairs (gray scale label map on the left side, RGB elevation map on the right side of each single pair).

further depicts a colorized version of the final thresholded label maps. It generally shows plausible *splatmaps* but also some problems. Especially at the borders of two segments one may note transitions of a third label that does not fit into the context. In addition, noisy regions with many labels can appear. This is probably due to the fact that the preprocessing step did not smooth these regions enough.

For now, we did not find a fitting simple metric for evaluating the plausibility of the label data. For example, we want to avoid labels where glaciers are next to deserts and other landscapes that rarely exist in reality. In order to achieve this, one could for example define rules derived from knowledge about climate zones and other terrestrial processes and then calculate whether the segments per label map do or do not fulfill the defined constraints. This would be similar to a set of constraints that is also used for classical terrain generation with biomes.

For the elevation data however, we could show that meaningful height maps are encoded in a sampling of generated outputs by rendering those maps as 3D-meshes. Figure 8 shows rendered results of two generated elevation maps. It shows, that the generated results indeed encode a structure with notable valleys and mountains that depict a plausible terrain. The rendering of labels in a meaningful way (reasonable textures or meshes per label) is not implemented yet, since the renderer as application is not part of this project. Therefore, default textures are used for the visualization in Figure 8 to provide a more intuitive visualization of the terrain.

One may note some spikes in the rendering of the original predicted elevation maps which is due to noise caused by the neural network. Since the renderer provides the functionality of applying erosion algorithms to height maps, a minor application of the thermal erosion algorithm as shown by [7] was used to smooth the elevation maps, making them look more realistic. Similar results may be achieved with a minor application of Gaussian filtering.



🟥	agriculture	🟩	grass
🟨	scrub	🟦	glacier
🟫	wood	⬜	rock
🟨	sand	🟩	park
🟦	water	🟥	hillshade
🟫	landcover	⬛	background

Figure 7: The threshold version and the colorized threshold version of the label maps as depicted in Figure 6 in the same order. Note that sometimes unfitting labels appear at the border between two segments. The right bottom label map is dominated by agriculture and wood for example. However the label 'glacier' appears at some borders, probably an artifact due to the generator failing to predict clear transitions. Noisy predictions can also occur as one can see when looking at the left bottom label map. Here, the pre-processing step of filtering and segmentation could not prevent label noise in the lower middle of the picture.

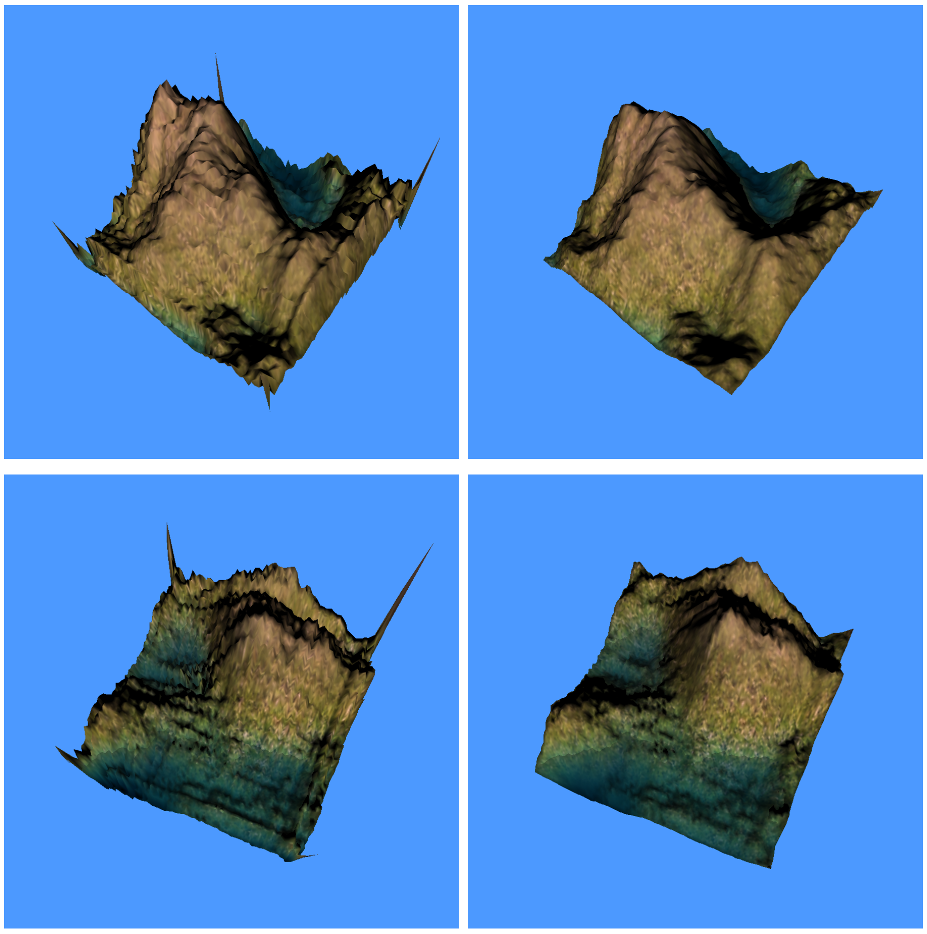


Figure 8: Rendering of two predicted elevation maps. On the left side, the rendering of two original height fields as generated by the network are depicted. The right side shows a smoothed result (application of the thermal erosion algorithm).

References

[1] Christopher Beckham and Christopher Pal. A step towards procedural terrain generation with gans. *arXiv preprint arXiv:1707.03383*, 2017.

[2] Kazimierz Choros and Jacek Topolski. Parameterized and dynamic generation of an infinite virtual terrain with various biomes using extended voronoi diagram. *J. UCS*, 22(6):836–855, 2016.

[3] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777, 2017.

[4] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[5] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[6] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 1(10):e3, 2016.

[7] Jacob Olsen. Realtime procedural terrain generation. 2004.

[8] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[9] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.