Pattern generation with neural cellular automata

Matic Pristavnik Vrešnjak, Žiga Lesar, Ciril Bohak

University of Ljubljana, Faculty of Computer and Information Science
E-mail: mp6760@student.uni-lj.si, {ziga.lesar, ciril.bohak}@fri.uni-lj.si

Abstract

Neural cellular automata (NCA) have been shown to perform well on a variety of tasks that require them to form self-organized structures. Over the years, they have been applied to various tasks such as 3D mesh texture wrapping, procedural generation of game levels and buildings, and more. In this work, we extend neural cellular automata to generate height maps that could potentially be used for 3D terrain generation.

1 Introduction

NCA are a class of models that combine local rule-based computation with the learning capabilities of neural networks (NNs). They have been successfully applied to a variety of tasks, including texture synthesis, dynamic image generation, mesh-based modeling, and procedural content creation. A key advantage of NCAs is their ability to form complex, self-organizing patterns through simple local interactions, making them especially suitable for tasks involving spatial coherence and robustness.

In this work, we explore the use of NCA for generating height maps that resemble those produced by traditional erosion algorithms, commonly used in terrain generation. Our approach builds on existing NCA architectures by introducing noise-based initialization, global positional encoding, and multi-scale filter kernels to enhance pattern complexity and inter-cell communication. We also compare our models to a baseline erosion method in terms of visual quality, generalization, and computational performance, and discuss their potential use for seamless chunk-based map generation.

The source code is publicly available on GitHub,[1] along with a live demo.[2]

2 Related Work

Mordvintsev et al. [4] introduced neural cellular automata, simulating living cells that communicate via chemical signals using 2D convolutions and Sobel filters. They apply a 1×1 convolution without ReLU, use masking to isolate living cells, and employ pooling strategies to improve stability and regeneration over extended time steps. Image orientation can be controlled by rotating the Sobel filters, and they demonstrate the model's ability to repair damaged regions. In a follow-up [3], the authors applied NCA to texture generation, arguing that natural textures emerge from particle interactions similar to partial differential equationss (PDEs), which can be learned by simple networks. Their approach is fast to train, highly parallelizable, and robust to noise, although it sometimes produces repetitive or nonsensical patterns.

Pajouheshgar et al. [7] extended NCA to dynamic images, improving inter-cell communication with multi-scale perceptrons and incorporating optical flow to model motion. Their system allows post-generation editing of automata behavior, including speed and direction, though it struggles with unrealistic motion paths such as curvilinear trajectories for linear patterns. In a later work [5], the authors adapted NCA to 3D meshes by replacing 2D convolutions with graph NNs and using spherical harmonics to guide the automaton, achieving higher-quality textures and supporting multimodal inputs like text, motion fields, and images. Their system can blend two textures on the same object and interpolate between them. They also found that using uniform noise as the seed improves generalization and resolution scalability, unlike non-uniform noise which leads to overfitting [6].

Finally, Earle et al. [2] applied NCA to 2D level generation, showing that despite only local perception, automata can propagate information globally, outperform GANs in quality, and explore level space more effectively, though at the cost of diversity.

3 Our Approach

3.1 Basic model

First, we implemented the model by Mordvintsev et al. [3]. We apply Sobel filters and a Laplace filter. After applying all the filters, their results are concatenated channel-wise and fed into the NN. The architecture of the NN consists of a dense layer followed by a ReLU activation function, and then another dense layer without any activation function. The result of the NN is then added to the previous output. The architecture of the NN is presented in [3, Figure 1].

To maintain stability during training, we also used the pooling method described in the original paper. This involves occasionally using an older image to update the model instead of always using the most recent one. It is worth noting that this basic method behaves similarly

[1] https://github.com/MaticVP/NCA-heightmaps
[2] https://funtimes.streamlit.app/

to the Euler integrator, while more advanced approaches could use Runge-Kutta methods. However, in our experiments, this mostly resulted in slower training times without any visible benefits or more diverse outputs.

3.2 Adding noise

Initializing the model with random noise can lead to better results [6]. The noise model of the NCA remains largely the same as the basic model, with the exception of the initial image. In this case, a certain amount of noise is added to the original image. According to the original paper, the required amount of noise varies from image to image. Therefore, we experimented with various types and levels of noise and primarily relied on visual inspection to determine whether the results were satisfactory. The amount of noise that we used for the final results was 0.20.

To test the above-described model, we used two types of noise: *Perlin noise* [8] – commonly used for procedural generation and Fractal Brownian Motion (FBM) [1] – a type of fractal noise where multiple octaves of noise are summed at different frequencies and amplitudes.

3.3 Adding global position awareness

This model was originally used to reconstruct animations [7], but we reworked it to function with static images. It introduces several new adjustments to the architecture. The first change is the use of multiple Sobel and Laplace filters of different sizes. This should allow the automata to learn information at different scales by enabling communication with more distant cells. The results of these filters are concatenated channel-wise.

Additionally, the authors used positional encoding to help the automata gain a sense of "awareness" of their location on the grid. The results of the positional encoding are then concatenated along with the outputs of the Sobel and Laplace filters and passed to the NN. The rest of the network remains the same as in the basic model. The architecture of the model is presented in [7, Figure 3].

3.4 Model training

3.4.1 Loss function

The loss function used to train the aforementioned models is the VGG16 [9] texture loss, which emphasizes the preservation of texture and style by comparing feature maps extracted from a pre-trained VGG16 network. For supervision, the ground truth image is selected based on the specific sample drawn from the pool of existing images. This approach helps guide the model toward producing realistic outputs and maintains consistency during training. As a result, it leads to more stable generations and effectively prevents the outputs from mutating or growing uncontrollably over time.

To further improve the controllability of the model's outputs and limit excessive growth, an additional term called *overflow loss* is introduced:

$$\mathcal{L}_{\text{overflow}} = \sum |x - \text{clamp}(x, -1, 1)|\,, \tag{1}$$

where x denotes the model's output, and $\text{clamp}(x, -1, 1)$ restricts the value of x to the interval $[-1, 1]$. This loss penalizes any values that exceed the valid output range. The final loss function is then defined as:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{texture}} + \mathcal{L}_{\text{overflow}}. \tag{2}$$

Combining both losses encourages the model to generate visually coherent textures while keeping the output within a reasonable range.

3.4.2 Training loop

The model was trained using a variable number of steps per epoch. This strategy is intended to help the model generalize more effectively and produce more diverse outputs. In total, the training process spanned approximately 2,000 to 4,000 epochs. During each epoch, the model performed between 64 and 96 training steps, depending on the specific setup. This variation in training steps introduces additional randomness and diversity, which can be beneficial for texture synthesis and generalization.

3.4.3 Chunk map training

In an attempt to create continuous regions between different patches or chunks, we trained a specific model tailored for this purpose. The idea was inspired by the original paper [3], where the authors demonstrated that if a certain region of the image was corrupted during training, the model was able to repair it by generating a new pattern that blended seamlessly with the overall texture.

Following a similar approach, we modified the training regime so that the model would occasionally receive cropped or partially masked images as input. The goal was to encourage the model to learn how to naturally fill in these missing regions. In theory, this should allow us to extract a chunk from each texture and merge them into a larger, coherent whole. However, in practice, this method does not always yield desirable results and can still lead to noticeable artifacts or discontinuities.

3.5 Erosion

Since it would not make much sense to simply mimic an already fast noise sampling algorithm, we instead chose to first generate noise textures and then apply an erosion algorithm. The reason for selecting erosion is that, to the best of our knowledge, CPU implementations of erosion are inherently slow, and there are no widely available CPU-based algorithms capable of running efficiently in real time. Additionally, erosion methods often suffer from the so-called "parameter hell": tuning the algorithm's parameters can be a tedious and time-consuming process. Achieving desirable results typically requires multiple iterations, each involving rerunning the already slow algorithm just to evaluate the impact of parameter changes.

The erosion algorithm we used is relatively straightforward and primitive, especially when compared to more complex CPU and GPU implementations. It operates by performing multiple iterations over a procedurally generated noise-based terrain. In each iteration, we simulate rainfall, calculate water flow direction using terrain gradients, and estimate sediment transport based on slope, water velocity, and capacity. Sediment is either deposited

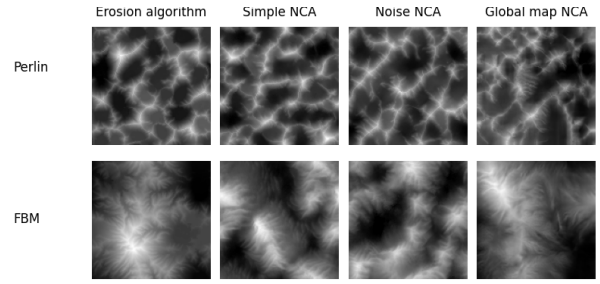

Figure 1: Examples of 2D height map textures obtained with each architecture.

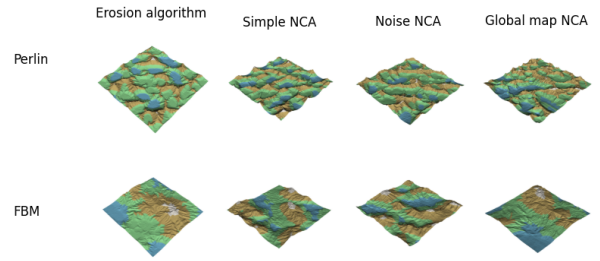

Figure 2: Examples of 3D meshes obtained with each architecture.

or eroded depending on local conditions, and both water and sediment are displaced in the downhill direction. Terrain is then smoothed using a simple slippage function. Over time, this iterative process sculpts the terrain into more natural forms, though the method remains basic in terms of physical accuracy and performance optimization.

4 Results

4.1 Single image results

In Figure 1, we can observe that the NCA models are able to generate outputs resembling the results achieved with the erosion algorithm. For Perlin noise, both the Simple NCA and the Noise NCA models produce results similar to those of the erosion algorithm. In the case of the Global Map NCA, there are slight deviations not present in the erosion algorithm. Most notably, some mountains exhibit centipede-like patterns along their edges, which do not appear in the other two architectures. It is also worth mentioning that both the Simple NCA and Noise NCA generate the most varied results among the three architectures, whereas the Global Map NCA tends to produce maps that are more or less identical to the one shown in this figure.

In the second row of Figure 1, when using FBM noise, the results differ slightly from those of the erosion algorithm. The erosion algorithm produces well-defined fractal-like structures, whereas all the NCA architectures yield centipede-like patterns. Other observations remain consistent with the Perlin noise case.

For a better interpretation of the results, we can examine the height maps rendered as 3D meshes, as shown in Figure 2. In the case of Perlin noise, we observe that the erosion algorithm is able to produce significantly more landmass compared to the NCA-based methods. Both

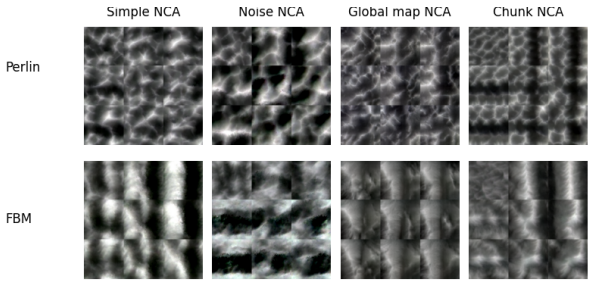

Figure 3: Examples of 2D height maps obtained with each architecture.

the Simple NCA and Noise NCA generate meshes with relatively small land areas that are predominantly mountainous. The only NCA variant that produces results with a reasonable amount of land is the Global Map NCA. This is likely due to improved communication between cells, allowing them to cluster more effectively. In contrast, the cells in the other two models often fail to communicate, resulting in sparse or fragmented structures.

Similar trends can be observed when using FBM noise. Once again, the Simple NCA and Noise NCA produce less land compared to the erosion algorithm. The centipede-like patterns become even more apparent in the 3D meshes. An interesting observation is that the Global Map NCA yields results that are very similar to the erosion baseline, showing variation in height but maintaining an overall consistent shape. Notably, this shape tends to remain stable regardless of how many times the model is rerun. The same observation applies to the Global Map NCA when using Perlin noise.

4.2 Chunk map results

In order to generate a map composed of chunks, it was necessary to train a model capable of connecting separate regions. From Figure 3, we can observe that NCAs using Perlin noise are better suited for chunk-based generation. Even models that were not explicitly trained for chunk generation perform better at interpreting spatial hints compared to the FBM-based models, which largely fail to connect regions effectively.

While models trained specifically for chunk generation tend to perform better than those that are not, they still occasionally fail to connect regions. Notably, even when the FBM model is trained with the objective of connecting regions, it often performs worse than some of the other FBM models not trained for this purpose. This suggests that the structure of FBM noise may inherently hinder the model's ability to bridge disconnected regions.

4.3 Image resolution

In Figure 4, we can observe that as the resolution increases, the texture begins to lose much of its detail. This occurs because the cells are unable to communicate effectively over large distances, resulting in patterns that gradually deviate from those produced by the erosion algorithm. The exception to this trend is the Global Map NCA model, which manages to maintain a somewhat similar pattern to the erosion-based output. This robustness is due to the

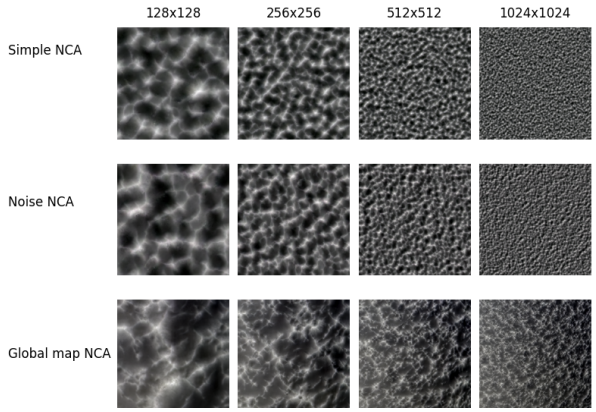

Figure 4: Examples of 2D Perlin height maps at different resolutions for each model.

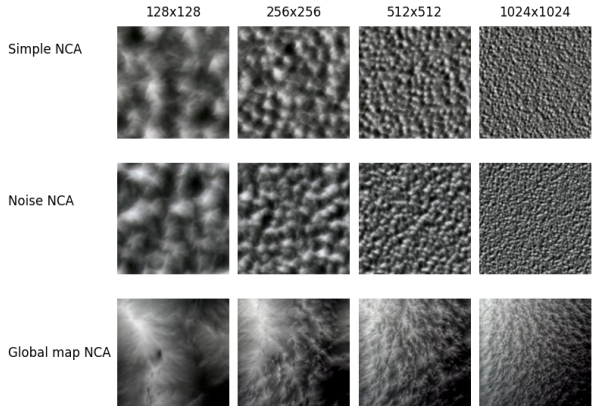

Figure 5: Examples of 2D FBM height maps at different resolutions for each model.

use of global positional encoding and the application of multiple Sobel filters at varying scales.

In Figure 4, we can observe that the results are largely similar to those produced with Perlin noise. However, a surprising detail is that the Global Map NCA begins to deviate and break apart as the resolution increases. This is unexpected, especially considering that, at least in the case of FBM, this model typically produces nearly identical results regardless of how many times it is rerun.

4.4 Speed comparisons

In Table 1, we observe that using NCA to approximate the erosion algorithm results in significantly better performance in terms of execution speed. However, it should be noted that the erosion algorithm could be made to run at nearly the same speed if we reduce the number of iterations or tune the parameters for performance. Nonetheless, such modifications would lead to a noticeable drop in output quality—often worse than what the NCA models produce.

We can also observe that among the three NCA architectures, the Global Map NCA is the slowest. This is expected, considering that this architecture applies multiple Sobel and Laplace filters and computes positional encoding before passing everything through the neural network. These additional computations naturally lead to increased inference time.

Table 1: Speed comparison at different resolutions (in seconds).

Method	128×128	256×256	512×512	1024×1024
Simple NCA	0.38	1.53	6.09	45.84
Noise NCA	0.422	1.56	6.63	41.50
Global map NCA	1.58	5.47	21.16	101.24
Erosion algorithm	2.40	16.34	65.01	515.92

5 Conclusion

In this paper, we demonstrated that neural cellular automata can be used for faster generation of erosion height maps. We also showed that the model tends to lose accuracy as the texture size increases, primarily because larger automata make communication between cells more difficult. Additionally, we showed that applying positional encoding can mitigate this problem, although not significantly. We also briefly described how our approach could be used for chunk-based map generation, but more work is required to make it truly practical and capable of running in real time.

Acknowledgment

This research was conducted as part of the basic research project *Cell visualization of unified microscopic data and procedurally generated sub-cellular structures* [project number J2-50221], funded by the Slovenian Research and Innovation Agency (Javna agencija za znanstvenoraziskovalno in inovacijsko dejavnost RS) from the state budget.

References

[1] Ton Dieker. Simulation of fractional brownian motion, 2004. Master's Thesis, University of Twente.

[2] Sam Earle, Justin Snider, Matthew C. Fontaine, Stefanos Nikolaidis, and Julian Togelius. Illuminating diverse neural cellular automata for level generation. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '22, page 68–76, New York, NY, USA, 2022. Association for Computing Machinery.

[3] Alexander Mordvintsev, Eyvind Niklasson, and Ettore Randazzo. Texture generation with neural cellular automata. *arXiv preprint arXiv:2105.07299*, 2021.

[4] Alexander Mordvintsev, Ettore Randazzo, Eyvind Niklasson, and Michael Levin. Growing neural cellular automata. *Distill*, 5(2):e23, 2020.

[5] Ehsan Pajouheshgar, Yitao Xu, Alexander Mordvintsev, Eyvind Niklasson, Tong Zhang, and Sabine Süsstrunk. Mesh neural cellular automata. *ACM Transactions on Graphics (TOG)*, 43(4):1–16, 2024.

[6] Ehsan Pajouheshgar, Yitao Xu, and Sabine Süsstrunk. Noisenca: Noisy seed improves spatio-temporal continuity of neural cellular automata. In *ALIFE 2024: Proceedings of the 2024 Artificial Life Conference*. MIT Press, 2024.

[7] Ehsan Pajouheshgar, Yitao Xu, Tong Zhang, and Sabine Süsstrunk. Dynca: Real-time dynamic texture synthesis using neural cellular automata. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 20742–20751, 2023.

[8] Ken Perlin. An image synthesizer. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, page 287–296, New York, NY, USA, 1985. Association for Computing Machinery.

[9] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.