BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2026
http://excel.fit.vutbr.cz

3D Neural Cellular Automata: An Interactive Framework for Volumetric Morphogenesis

Michal Repcik*

Abstract

Existing 3D Neural Cellular Automata implementations are mostly batch scripts with no interactive training tooling and no practical workflow for modelling targets and exporting ready-to-use coloured voxel datasets (Fig. 1). This work presents a modular framework consisting of a Blender add-on for mesh voxelization and live viewport rendering (Fig. 2), a TCP server exposing any training backend over a local or tunnelled connection, and a standalone Python core package usable in Jupyter notebooks without Blender; one training step is summarised in Fig. 3. The framework is validated through morphogenesis experiments on symmetric and asymmetric targets, and through multi-target metamorphogenesis, where a single model grows distinct voxel shapes conditioned on task channels (Figs. 4–5). Implementing a new NCA experiment requires subclassing one abstract class; the server, protocol, logging, and visualisation pipeline require no changes.

*xrepcim00@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Neural Cellular Automata (NCA) [1] replace the hand-crafted rules of classical cellular automata with a small shared neural network trained by gradient descent. Each voxel perceives its local neighbourhood through fixed convolutional filters and applies a learned residual update. Complex global structure – growth, regeneration, pattern formation – emerges from repeated local interaction without any centralised coordination.

Extending NCA to three-dimensional voxel grids is conceptually straightforward. Prior work confirms that volumetric growth is feasible [2][1][2], but most implementations remain standalone training scripts. They lack live visualisation, interactive hyperparameter steering, and a mesh-to-voxel dataset workflow.

This work presents a framework that addresses these limitations. It consists of three decoupled layers: a Blender add-on, a TCP training server, and a standalone core package. The central design goal is that implementing any NCA experiment requires writing one class; everything else is provided by the framework.

Main contributions are: (1) a mesh-to-voxel dataset workflow integrated into Blender, including export of ready-to-use voxel targets (Fig. 1); (2) live 3D training visualisation and interactive control through the Blender UI (Fig. 2) with a decoupled backend boundary; and (3) model-level extensions – positional channels and task conditioning – that address asymmetric targets and enable multi-target metamorphogenesis (Figs. 4–5).

[1] Framework source code (E. Rossi 2021 base): https://github.com/e-rossi/3d-nca.

[2] Extension repo (A. Jha 2024): https://github.com/Aadityaza/3d-Growing-neural-cellular-automata.

2. System Architecture

Blender add-on. The add-on performs mesh voxelization and dataset export (Fig. 1) and viewport rendering of live training snapshots (Fig. 2). A selected mesh is sampled onto a 3D grid using a raycast inside-outside test. Per-voxel colour is extracted either from the Principled BSDF base colour (fast) or via UV texture lookup (general). The voxelized target can be exported as `.npz` or transmitted directly to the training server.

During training, state broadcasts received from the server are rendered as merged-cube meshes in the Blender viewport. The source mesh, voxelized target, and current NCA state appear side by side in three spatial slots for direct visual comparison during training.

The sidebar control panel exposes connection settings, session control, and a schedule editor for one-shot parameter updates (Fig. 2).

Server and protocol. The server exposes training over TCP and runs the selected backend in a background thread while streaming state snapshots at a controlled cadence. For remote GPU training on Colab or Kaggle, an ngrok TCP tunnel bridges the connection with no firewall configuration required.

Core package. The core package provides a clean PyTorch NCA implementation with configuration dataclasses and self-describing checkpoints. It is independently usable from Python scripts or Jupyter notebooks via the high-level `NCAModel` API.

3. NCA Model

State representation. Each voxel holds a real-valued vector of C channels partitioned into hidden, positional, task, and visible groups. Hidden channels carry unsupervised internal memory. Visible channels (RGBA) define the output compared against the target. Positional channels encode normalised absolute coordinates $(x, y, z) \in [0, 1]^3$, re-injected as read-only constants after each step to provide global spatial reference that purely local symmetric perception cannot supply. Task channels enable one model to learn multiple target behaviours via one-hot conditioning.

Perception and update. Fixed $3 \times 3 \times 3$ depthwise convolutions produce a perception vector at every cell. The baseline configuration uses three filter groups: identity, 6-neighbour sum, and Laplacian. The directional configuration uses five groups: identity, 6-neighbour sum, and signed gradients along x, y, and z. Directional filters resolve reflection ambiguity for asymmetric targets, where the isotropic baseline converges to mirrored solutions.

Combined with positional channels (x, y, z) as read-only constants, the update rule receives an explicit global frame of reference while remaining strictly local and translation-invariant in its learned parameters.

The update MLP maps the perception vector to a bounded state increment, $\Delta\mathbf{s} = 0.1 \cdot \tanh(\mathrm{MLP}(\mathrm{perceived}))$, applied residually: $\mathbf{s}_{t+1} = \mathbf{s}_t + \Delta\mathbf{s}$. An alive mask computed by $3 \times 3 \times 3$ max-pooling of the alpha channel and thresholding gates which cells receive updates. Dead cells adjacent to alive cells are included in the mask and can grow; isolated dead regions cannot activate.

Training. Pool-based training maintains a pool of persistent states across developmental stages. Each iteration runs the automaton for a randomly sampled number of steps under a step-range curriculum, reseeds the highest-loss sample to maintain diversity, and optimises a weighted sum of alpha loss, masked colour loss, and an overflow penalty for activity outside the target volume. In the default configuration, the pool holds 32 states and the step range expands from 8–16 steps early in training to 32–64 steps later to encourage stability over longer horizons.

4. Experiments

All experiments use a 32^3 grid, 16 hidden channels, and 4 visible (RGBA) channels, seeded from a single alive centre cell.

Morphogenesis. A symmetric target converged from total loss 2.36 to 4.5×10^{-3} over 3000 epochs. An asymmetric target (chess pawn (Fig. 1) with the symmetric baseline configuration repeatedly converged to mirrored intermediate solutions. Enabling positional channels and directional perception resolved this; the corrected run reached a total loss of 2.75×10^{-4} at epoch 4000. This failure and its resolution directly motivated positional channels as a framework-level feature.

Metamorphogenesis. Metamorphogenesis denotes a multi-target regime in which one model both generates form and transitions between forms through task-channel conditioning (Figs. 4–5). A related approach to conditional morphogenesis in NCA was published concurrently [3]; the implementation here demonstrates that the same conditioning idea is achievable with minimal code changes to the framework, a custom `NCARunner` subclass with no modifications to the server, protocol, or visualisation pipeline, confirming the practical extensibility of the design.

Training uses a phased curriculum: the model first learns a single target, then expands to additional targets at phase boundaries (Fig. 5). Pool states can be relabelled across phases to preserve continuity of the transition, producing characteristic loss spikes followed by re-convergence (Fig. 4).

5. Conclusions

This work presents a modular 3D NCA framework connecting mesh-based dataset preparation, remote GPU training, and live Blender viewport visualisation in a single interactive workflow. Experiments validate symmetric/asymmetric morphogenesis and task-conditioned metamorphogenesis, and show positional channels as an effective remedy for asymmetric reflection ambiguity.

Acknowledgements

I would like to thank my supervisor for guidance throughout this work.

References

[1] Alexander Mordvintsev, Ettore Randazzo, Eyvind Niklasson, and Michael Levin. Growing neural cellular automata. *Distill*, 2020.

[2] Shyam Sudhakaran, Djordje Grbic, Siyan Li, Adam Katona, Elias Najarro, Claire Glanois, and Sebastian Risi. Growing 3d artefacts and functional machines with neural cellular automata. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 2021.

[3] Ali Sakour. Conditional morphogenesis: Emergent generation of structural digits via neural cellular automata. *arXiv preprint arXiv:2512.08360*, 2025.