Artipixoids!
The Concept (draft)

by Andrey Zamaraev
a5kin@github.com

April 19, 2017

2D body, 2D brain,
2D pleasure, 2D pain.
— Flatlander credo

Artipixoids! is half generative art, half artificial life project with a focus on genetic cellular automata obeying energy conservation principle. Started in the end of 2014, it has undergone several major modifications, and is still in development. But now is the right time to share basic concepts underlying all previous modifications, so others could use it to build their own wonderful worlds.

The document is split up in two parts. In *The Manifesto* we will outline the main areas of the project development, and the main goals per each area. In *The Core* section, we will give the formal definitions to the features we would like to see in the core of our future framework.

Contents

1 The Manifesto

1.1 Artistic

1. We are going to generate the beautiful objects of abstract digital arts, even at the cost of computational resources.
2. The objects of arts are: static images, videos, sounds, 3D models, etc.
3. We will consider the object as Artipixoid if and only if:
   (a) the object is generated by CA fulfilling all the definitions from section 2 (The Core);
   (b) the object's generative algorithm is not using anything but CA states as input;
   (c) the object's generative algorithm is not based on the objects of arts, other than Artipixoids (eg. deep art based on Van Gogh paintings or photos doesn't count, but deep art based on other Artipixoids is OK).
4. Artipixoids that has no artistic value are bad and a subject to genocide.
5. Yet, if at least one person found them beautiful, we let them live.

1.2 Scientific

1. We are going to make experiments to test the hypothesis that a life can self-organize in an artificial (digital) environment from a random initial state.
2. ***The properties of life***, as we see it, are following.
   (a) *Integrity:* every lifeform should be distinct from its local environment, and should preserve its individuality during the lifespan.
   (b) *Sustainability:* a lifeform should be able to self-repair in case of damage, and to maintain the physical manifestation (body) for a long enough period of time.
   (c) *Reproductivity:* a lifeform should be able to replicate, creating new individuals with small enough differences to consider them inexact copy.
   (d) *Novelty:* new lifeforms should arise out of the old ones, and they should act more effectively than the old ones.

(e) *Diversity:* there should be enough different lifeforms to put them in groups called "species".

(f) *Complementarity:* different species should learn to live in symbiosis and help each other to survive and improve, forming stable ecosystems.

(g) *Adaptability:* species should learn to evolve in a way they are constantly adapting to aggressive environmental changes.

3. ***The properties of artificial environment***, as we see it, are following.

(a) *Discreteness:* the space and time of our environment should be discrete. So, space should consist of indivisible quanta, and their updates should occur in separate timesteps synchronously. In addition, any parameter of spatial quanta should take integer values only. This requirements will allow us to run the simulation of our environment on modern digital computational systems.

(b) *Determinacy:* a global state of the environment at the timestep t should lead to the one and only one global state at the timestep $t + 1$. This will give us insurance of the exactly same outcome, every time we run a particular experiment with the same initial state.

(c) *Locality:* each quantum should have the same number and the same topology of its neighbors, and quantum's state should be updated using nothing more than the states of its neighbors and its own state. This will allow us to exploit the massively parallel computations, thus greatly speeding up the simulation.

(d) *Isolation:* the environment as a whole should behave like an isolated thermodynamic system. So, each quantum should have an energy level, and sum of all quanta energy levels should be a constant value. The energy conservation will make us sure, that none of our experiments will lead neither to the extinction nor to the explosion of the "matter".

(e) *Isotropy:* all possible reflections of quantum's neighborhood at the timestep t should lead to the same new quantum's state at timestep $t + 1$. This will allow energy to spread in all possible directions uniformly.

(f) *Polymorphism:* each quantum should have a microprogram, influencing its local update rules to some degree. A microprogram in return could also be changed according to update rules. This will allow us to add genetic informational component to the quantum level. We believe, this component is the key for the emergence of self-organized matter effects at the higher (macro) levels.

(g) *Stochasticity:* optionally, some pseudorandom component would influence local updates too. For that, each quantum should have its own PRNG with the long enough period. This will allow energy to spread in chaotic clouds, rather than in direct lines.

(h) *Mutability:* optionally, a quantum's microprogram would randomly mutate over time. This will result in a wider genetic search for the better microprograms, and also may prevent the whole environment from being conquested by a single genome.

(i) *Magnetism:* optionally, quanta would have the tendency to "attract" each other with some sort of force (gravitational, electromagnetic, or any other). We believe, magnetism is another key aspect behind the phenomenon of life as we know it.

4. ***Random initial state***, could be obtained using following known patterns.

   (a) *'Primordial Soup':* each and every quantum initially has an equally small amount of energy, other parameters are random. This is a good test for the ability of energy to self-organize in clusters from the completely uniform distribution. The bigger clusters are, the better. Fractal clusters are even better.

   (b) *'Big Bang':* a small area of space is initialized with a high amount of energy and random parameters per each quantum. Outside the area, quanta has either zero or minimum possible amount of energy. This is a good test for the ability of energy to spread in empty space. We could also detect the fastest types of energy using this pattern.

   (c) *'Super Cluster':* same as Big Bang, but several areas with a bit lower energies are randomly initialized. The positions of areas are random across the space. This is a good test for gravitational effects, and also for collisions of different energy beams, produced in micro-bangs.

5. To obtain the better results, we reserve the rights to intervene the natural process of evolution. This will ruin the idea of self-emergence, but still may prove the possibility of artificial life in our environment.

   The presumable methods of intervention are following.

   (a) *Genetic selection:* during the simulation, we are collecting the most interesting and promising areas of space, then combine them all in a single initial state and repeat the process recursively while novel phenomena of interest are showing up.

(b) *Directed mutation (a.k.a. 'Hand of Fate')*: during the simulation, we are occasionally pointing our finger to a small area of interest, and all quanta in this area are randomly changing parameters, excluding energy level. Then, evolution continues.

(c) *Pre-designed components:* we are designing patterns of matter manually quantum by quantum, then using them either as a parts of more complex designs or to impact on 'raw' matter in some way.

1.3 Divine

1. We, as young goddesses and gods, are willing to create worlds with one click of our fingers. Actually, it doesn't have to be a click, a tap of one finger is even better.

2. After the tap, we are willing to say the Word, which contains the essence of the future world.

3. Then, the world begins to bloom, and it blooms the same every time we say the same Word.

4. Some of us will just meditate on the beauty of their worlds, wandering around and changing the views.

5. Others will actively play demiurges using scientific wizardry from previous section, or even more powerful stuff.

6. And some will get bored and destroy the creation with a single tap of the finger. Thus, letting its inhabitants live in some parallel universe, free of divine will and happy.

1.4 Social

1. We are going to build the community over those who interested in any of previously mentioned ideas.

2. For that, we need a framework, implementing basic CA engine described below, along with a set of modular tools to easily build engine modifications.

3. Then, we will be able to exchange new modifications in a form of compact modules.

4. We will also be able to exchange new modular tools and extend our framework with them for future re-using.

5. Finally, for each modification, we will be able to make a collection of the experiment files with initial state and hyperparameters included, and run them in uniform interactive environment on massively parallel hardware like GPU clusters.

1.5 Alien

1. We are going to monitor any signs of life in the artificial environment described in Section 1.2.

2. We will study the potential candidates for the role of living creatures.

3. We are swearing to refrain from any act resulting in irreversible harm to any particular creature we will consider as sentient.

4. We will try to make a contact with those creatures we will find intelligent.

5. We are responsible for any loss of data containing sentient or intelligent creatures.

2 The Core

There is a special subclass of cellular automata that allows us to implement all the properties of environment described in section 1.2. We call it *Buffered State Cellular Automata* (BSCA). In this section, we will give the formal definition of all BSCA features, to be integrated in the core of our future framework.

2.1 Grid and neighborhood topologies

As in any CA, grid in BSCA is D-dimensional lattice of cells, each having N neighbors. $(D, N) \in [1..\infty)$.

The cells are stored in a sequence of length M:

$$C = \langle c_0, ..., c_{M-1} \rangle, \quad M \in [1..\infty), \tag{1}$$

where elements are nested sequences with uniform structure called *cell state*. See the exact definition of the cell state in Eq. 17 below. We may refer to C as to a *board state* later.

The cartesian coordinates of the cell with index i could be obtained as

$$[x_0, ..., x_{D-1}] = \chi(i), \quad i \in [0..M), \tag{2}$$

where χ is *lattice topology function*.

Let also define a sequence of all cells' coordinates as

$$X = \langle \chi(0), ..., \chi(M-1) \rangle. \tag{3}$$

We may refer to X as to just a *board* later.

The cartesian coordinates of cell's j-th neighbor could be obtained as

$$[x_0, ..., x_{D-1}] = \nu(X_i, j), \quad i \in [0..M), \quad j \in [1..N], \tag{4}$$

where ν is *neighborhood topology function*.

Thus, we will assume the whole grid topology is homogeneous if and only if the following equation holds for each value of j:

$$X_i - \nu(X_i, j) = d_j, \quad \forall i \in [0..M), j \in [1..N], \tag{5}$$

where d_j is a constant vector for each j. So, the vector difference between the positions of cell and its j-th neighbor should be a constant for each cell in a grid.

2.2 Border Effects

A *border cell* is a cell, having at least one of its neighbors off the board X. If A_i is a set of all i-th cell's neighbors, that are inside X:

$$A_i = \{C_i \mid \nu(X_i, j) \in X, j \in [1..N]\}, \quad i \in [0..M), \tag{6}$$

then the following inequality holds for all border cells:

$$|A_i| \neq N, \quad i \in [0..M). \tag{7}$$

In order to correctly process all cells, including border ones, we have to define a conditional function η for neighbors obtainment. Let call it a *neighbor function*:

$$\eta(i, j) = \begin{cases} C_{\chi^{-1}(\nu(X_i,j))} & \text{if } \nu(X_i, j) \in X, \\ \beta(\nu(X_i, j)) & \text{if } \nu(X_i, j) \notin X, \end{cases} \tag{8}$$

where β is a *border function* returning a state for a hypothetical cells outside X, and χ^{-1} is a reverse lattice topology function, satisfying the following equation:

$$\chi^{-1}(\chi(i)) = i. \tag{9}$$

Border function β could take a variety of forms. It could just be a constant predefined state (static borders). Or wrap borders into higher dimensional manifold topology, like torus, Möebius strip or Klein bottle. It could even yield a random state each time.

In case of wrapping, the border function will take a form of

$$\beta(x) = C_{\beta^*(x)}, \quad \beta^*(x) \in [0..M), \tag{10}$$

where β^* is a *wrap function*, mapping off-board coordinates to other cells on the board, and on-board coordinates strictly to the cells with those coordinates:

$$\beta^*(x) = \begin{cases} \chi^{-1}(x) & \text{if } x \in X, \\ i \in [0..M) & \text{if } x \notin X. \end{cases} \tag{11}$$

We will assume that the whole grid neighborhood is homogeneous if and only if the border function takes a form as in Eq. 10, and the following condition holds:

$$C_i \in \{\eta(i', j) \mid j \in [1..N]\}, \quad i' \in \{\beta^*(\nu(X_i, j)) \mid j \in [1..N]\}, \quad i \in [0..M), \tag{12}$$

so, each cell should be present in a set of neighbors for all of its neighbors.

Be warned, any border function that breaks the neighborhood homogeneity, may also break an energy conservation, unless the buffered interactions with off-board cells are explicitly restricted in update rules. See Section 2.7 for more details.

2.3 Single Cell Design

Let recursively define a *parameter* as either an integer or a sequence of parameters. Then, the infinite set P of all possible parameters is

$$\begin{aligned} P_0 &= \mathbb{Z}, \\ P_{n+1} &= \bigcup_{i=1}^{\infty} P_n^i, \end{aligned} \tag{13}$$

thus we can say that p is a *parameter* if and only if $p \in P$.

Let also define a recursive function φ, that will replace all integers with zeros in a parameter:

$$\varphi(p) = \begin{cases} 0 & \text{if } p \in \mathbb{Z}, \\ \langle \varphi(p_0), ..., \varphi(p_{|p|-1}) \rangle & \text{if } p \notin \mathbb{Z}. \end{cases} \tag{14}$$

We will call φ *parameter topology function*, and will assume two parameters p_1 and p_2 have the same topology if and only if the following equation holds:

$$\varphi(p_1) = \varphi(p_2). \tag{15}$$

The *generic state* then is a parameter with the constant topology within a particular BSCA. Let define S_s as a set of all possible generic states:

$$S_s = \{s \mid s \in P, \varphi(s) = p_c\}, \tag{16}$$

where p_c is a constant topology value.

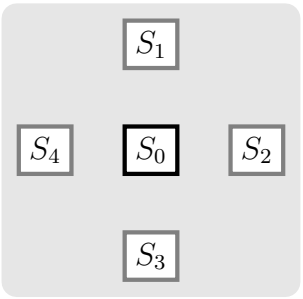

Figure 1: Single cell design in BSCA.

The *cell state* c (see Eq. 1) is a sequence of $N+1$ generic states:

$$c = \langle S_0, ..., S_N \rangle, \quad S_i \in S_s \forall i \in [0..N], \tag{17}$$

where S_0 is the *main state*, and S_1, ..., S_N are the *buffered states*, one per each neighbor. At the Fig. 1, you can see a cell design for 2-dimensional BSCA with Von Neumann neighborhood. Buffered neighbor states is the crucial aspect of BSCA, it is the basic feature allowing us to implement lossless models with energy exchanges and genome crossbreeding across all neighbor cells. See more on this topic in the sections below.

Let also define two more sets for future use:

$$S_c = S_s^{N+1}, \tag{18}$$
$$S_b = S_c^M, \tag{19}$$

where S_c is a set of all possible cell states, and S_b is a set of all possible board states.

2.4 Time and Updates

The *universe* is a board state evolving in discrete timesteps. Let define the universe U as a potentially infinite recurrent sequence:

$$\begin{aligned} U_0 &= C_s, \\ U_{t+1} &= \omega(U_t), \end{aligned} \tag{20}$$

where C_s is an *initial board state* (also called *seed*), t is a timestep number and $\omega : S_b \to S_b$ is an *update function*.

In BSCA, the evolution at a single timestep is going in two phases: *emit phase* and *absorb phase*, so update function will take a form of

$$\omega(u) = \alpha(\varepsilon(u)), \tag{21}$$

where α is *absorb function* and ε is *emit function*. Let describe how both of them works in the next sections.

2.5 Emit Phase

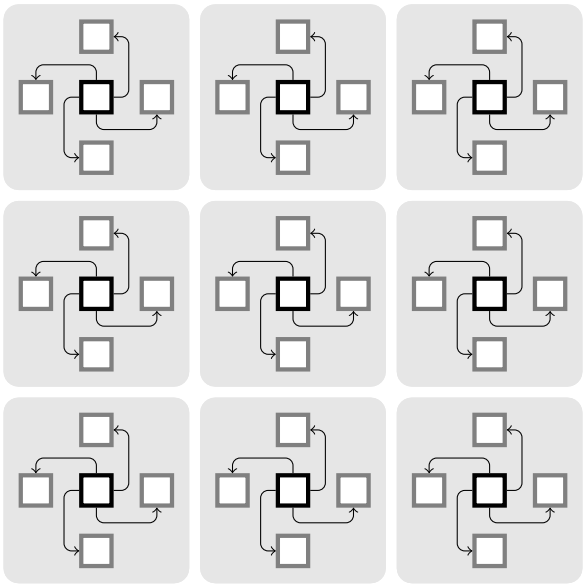

Figure 2: Emit Phase in BSCA.

The process of emit is like sending the messages from a cell to its neighbors. Each cell performs it as follows.

1. The cell is taking a sequence of all its neighbors' main states, as well as its own main state:

$$\eta_e(C, i) = \langle \gamma(C_i, 0), \gamma(\eta(i, 1), 0), ..., \gamma(\eta(i, N), 0) \rangle, \tag{22}$$

   where i is the cell's index and γ is a function for obtaining an item from the sequence:

$$\gamma(s, i) = s_i. \tag{23}$$

2. The cell is modifying its buffers and main state as a result of *decision function* $\sigma : S_c \to S_c$:

$$C_i := \sigma(\eta_e(C, i)), \tag{24}$$

   where i is the cell's index and C is a board state at the current timestep t: $C = U_t$. The decision function σ could be as complex as you need: from naive copy of main state to sophisticated microprogram execution.

3. At the end of emit phase, each buffer should contain a generic state intended for a specific neighbor.

The whole emit function $\varepsilon : S_b \to S_b$ thus takes a form of

$$\varepsilon(C) = \langle \sigma(\eta_e(C, 0)), ..., \sigma(\eta_e(C, M-1)) \rangle, \tag{25}$$

Fig. 2 shows the emit phase for 2-dimensional BSCA with Von Neumann neighborhood. The arrows are indicating the updates. So, at this phase, the main states are read-only, and the buffers are write-only within every given cell, which makes the process "thread safe".

2.6 Absorb Phase

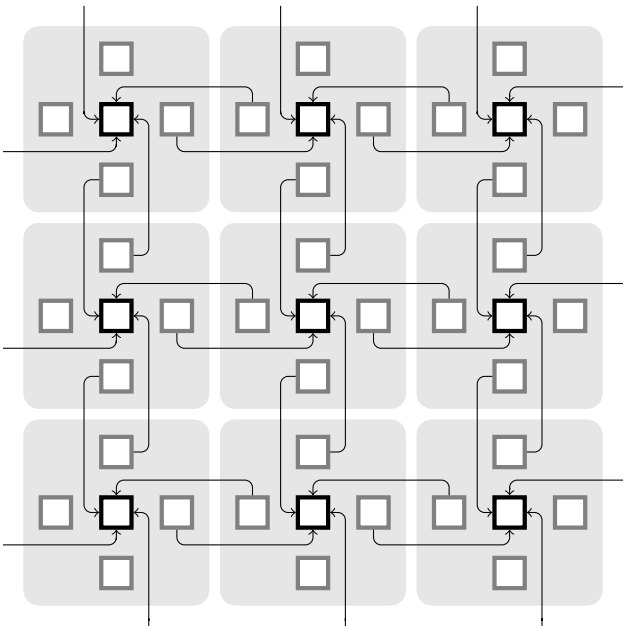

Figure 3: Absorb Phase in BSCA.

At the absorb phase, each cell is receiving buffered states as "messages" from its neighbors, then evolving. It goes as follows.

1. From each neighbor, the cell is taking a buffered state intended for itself, and making a sequence from these states, along with its own main state:

$$\eta_a(C, i) = \langle \gamma(C_i, 0), \gamma(\eta(i, 1), \iota^{-1}(i)), ..., \gamma(\eta(i, N), \iota^{-1}(i)) \rangle, \tag{26}$$

where i is the cell's index, γ is a function from Eq. 23, and $\iota^{-1} : \mathbb{Z} \to \mathbb{Z}$ is the function for reverse neighbor index obtaining, for which the following equation holds:

$$\eta(i, j) = \eta(\chi^{-1}(\nu(X_i, j)), \iota^{-1}(j)). \tag{27}$$

2. The cell is modifying its own main state as a result of *blend function* $\psi : S_c \to S_c$:

$$C_i := \psi(\eta_a(C, i)), \tag{28}$$

where i is the cell's index and C is a board state at the current timestep t: $C = U_t$. The blend function ψ is generally performing a combination of incoming buffer states. It could be a sum or mean value or some genome crossover etc. Although, no one can keep you from inventing more exotic blend functions.

3. At the end of absorb phase, each cell's main state is evolved and a subject to the observation, you can record it, use in visualization and so on.

The whole absorb function $\alpha : S_b \to S_b$ thus takes a form of

$$\alpha(C) = \langle \psi(\eta_a(C, 0)), ..., \psi(\eta_a(C, M - 1)) \rangle, \tag{29}$$

Fig. 3 shows the absorb phase for 2-dimensional BSCA with Von Neumann neighborhood. The arrows are indicating the updates. So, at this phase, the buffered states are read-only, and the main states are write-only within every given cell, which in combination with emit phase, gives us the ability to safely run the whole process on massively parallel architectures.

2.7 Energy conservation

Using the design from previous sections, it is very easy to define a model that will behave like an isolated thermodynamic system (see Item 3d from Section 1.2). If $E : S_s \to \mathbb{Z}$ is a function for extracting the energy component from the generic state, then we can say the system will obey the energy conservation principle if the following equations holds:

$$\sum_{i=0}^{N} E(c_i) = \sum_{i=0}^{N} E(\gamma(\sigma(c), i)), \quad \forall c \in S_c, \tag{30}$$

$$\sum_{i=0}^{N} E(c_i) = \sum_{i=0}^{N} E(\gamma(\psi(c), i)), \quad \forall c \in S_c. \tag{31}$$

Thus, the summed energy should stay the same for the argument and the result of either decision or blend function.

The broken neighborhood homogeneity (see Eq. 12) could easily be a threat for energy conservation. For example, in a case of static borders the energy may leak over the edge of the board. To prevent this, we must restrict buffered interactions with the off-board cells: at the emit phase, all the buffers pointing over the edge should be set to a state of zero energy, and Eq. 30 should still be held.

Please also note, energy conservation is the mandatory criteria for "genuine" Artipixoids (see Item 3a from Section 1.1).

2.8 Genetic Microprograms

As we stated in The Manifesto, informational component is the key aspect to the phenomenon of life as we know it. Without information, the energy would just be a dull clouds of dust wandering randomly in space and time. But, as we all can see, the matter is organizing in the complex self-replicating patterns, and this is happening at each level of existence: from galaxies to atoms. We believe, this principle of self-organization is scaling fractally from the very bottom level of indivisible Planck scale quanta, each behaving like a machine executing a genetic microprogram.

Since the cell is the smallest unit in our artificial environment, it's a good candidate for microprogram execution. For that, it must have several pre-defined parameters:

- an integer parameter p_p to hold an execution pointer;
- an integer parameter p_m to hold a set of microinstructions (program memory), L_m bits in total, L_c bits per instruction, $L_m, L_c \in [1..\infty)$;
- a sequence of integer parameters to hold some values (registers): $p_r = \langle p_{r_0}, p_{r_1}, ... \rangle$

Then, at the emit phase the cell acts as follows:

1. Incrementing execution pointer, and returning it to the beginning as needed:

$$p_p = (p_p + 1) \bmod (L_m / L_c). \tag{32}$$

2. Obtaining a microinstruction at the position of execution pointer:

$$p_i = (p_m \text{ lsh } (p_p * L_c)) \text{ and } (2^{L_c} - 1). \tag{33}$$

3. Modifying its own state as shown in Eq. 24, but the decision function is gotten from the pre-defined sequence of functions, one function per each microinstruction:

$$\sigma \Leftrightarrow \sigma_{p_i}. \tag{34}$$

   Thus, the decision function is depending on the current microinstruction.

We may consider a microprogram's memory as a genome. Then, at the absorb phase, the cell can change its microprogram with some genetic crossover algorithm using all "incoming" genomes from neighbor's buffers. The exact crossover algorithm should be implemented as a part of blend function ψ (see Eq. 28)

The microinstructions could take many forms. Here, we'll give some insights to the ideas we are actively using in our microprograms.

Value Saturation. The energy values of neighbors could be saturated to a single bit. A possible *saturation function* $\theta : S_s \to \{0, 1\}$ may take a form of:

$$\theta(s_m, s_n) = \begin{cases} 0 & \text{if } E(s_m) < E(s_n), \\ 1 & \text{if } E(s_m) \geq E(s_n), \end{cases} \tag{35}$$

where s_m is a cell's main state, s_n is a neighbor cell's state, and E is a function from Eq. 30. Then, we can use those bit values in the functions described below in this list. You can use other variations of saturation functions as well, like a fixed threshold, hysteresis, etc.

Gated View. Sometimes, you need to reduce a number of neighbors to operate on. For that, let introduce a special parameter $p_g \in [1..N]$ called *gate*. The gate is an integer pointing to a neighbor's index. Thus, you can use this value i.e. for spreading an energy to a gate's direction only, or to get 3 saturated neighbors' values around the gate for using in elementary rule.

The static gate, of course, would be quite boring. To make it dynamic, let define another function $\xi : [1..N] \to [1..N]$, which is called *spin function*.

Then, after each step, the gate would change its value as a result of this function:

$$p_g := \xi(p_g). \tag{36}$$

The spin function may run the gate in loops of pre-defined sequences, or set it accordingly to other parameters' values or even change it randomly.

Elementary Rules. One of the easier microprograms are those which using elementary CA rules as instructions. So, a microprogram is a set of rules, executing sequentially. Each rule directly mapping the neighborhood state to the state of the cell at the next timestep. The cells' states are binary, so here we may use saturated energy values as an input. The outcome is also a binary state: cell is either alive or dead. If we are using a model with energy conservation, we may interpret "dying" as a command to spread cell's energy (either whole or part) to its neighbors. The "birth" may be simulated by setting a special parameter, telling the dying neighbors at the next timestep to send more energy to the direction of the "borning" cell.

In case when we are using a rule with lower dimension (e.g. 1D rule in 2D automaton), we may use a gate to get only those neighbors that are closer to it. Then, apply corresponding rule, and, in the case of "dying", spread energy only in the direction of the gate.

Since rules may take a lot of memory, we may also increment the execution pointer by a value less than a microinstruction (rule) width. This may produce interesting effects of "shifting" rules.

Turing Machines. Another possible superset of microprograms is based on Turing machines. Here, neighbors may be interpreted as a tape, the gate as a head, the genetic microprogram as a table of states, and the execution pointer as a Turing machine state register. Then, at the emit phase, we are **a)** getting a machine state by the execution pointer **b)** getting a saturated value from the neighbor at the direction of the gate **c)** sending additional energy in the direction of the gate if the instruction telling us to write a 1 to the tape and **d)** modifying the gate and the execution pointer according to the instruction.

The gate is modifying as follows:

$$p_g := \begin{cases} \xi(p_g) & \text{if head is shifting left,} \\ \xi^{-1}(p_g) & \text{if head is shifting right,} \end{cases} \tag{37}$$

where ξ^{-1} is *reverse spin function.*

XOR Networks. This is another approach we found interesting for microprograms implementation. XOR Network is a graph with boolean values in the vertices. At the next timestep, each vertex taking a new value, XORing all the values of vertices it is sharing an edge with. The idea of using it in our automata is very simple. We are just looping through the vertices, getting their boolean values and using them as the instruction to "born", "live" or "die", just like for Elementary Rules.

The isolated XOR network is always changing its state in loops, usually it has several independent loops with "garden of Eden" states. But, if you interconnect a lot of XOR networks together and let them influence each other, the result would be quite unpredictable. Let just share some possible scenarios.

1. The network topology is fixed for all cells, but neighbors could change a value of the vertex at the execution pointer.
2. The network topology is encoded in genome which is different for each cell, and a subject to interbreed with the neighbors.
3. The cells are emitting buffered values, that should be XORed with the neighbor's current vertex in addition to the network inner logic.
4. The Elementary rule is encoded in vertices values and changing with the state of network. Then, at each timestep the resulting rule is applying to the neighbors as described in Elementary Rules paragraph.
5. The vertices are implementing different boolean operators, which are encoded into cell's genome. It takes us away from the XOR concept, but the results may still be interesting.

XOR networks has a lot in common with the loop quantum gravity concept. Therefore, we must pay more attention to them if we'd like to implement models with the gravitational effects.

2.9 Parameter Classes

To make it easier to implement parameters for a new CA, we may inherit them from some pre-defined basic classes. In this section, we will categorize parameters into possible basic classes.

Scalar value. The most basic parameter, it is just a common integer.

Conserved value. The scalar with additional checks for conservation. Every time we are changing this parameter in the buffer state, the system automatically changes the main state with the opposite sign, and vice versa.

Angular value. The scalar "rotating" in K grades. Generally, the system makes sure that the value is in $[0, K)$ range, dividing it modulo K. But there are also special cases of angular mean and angular difference to be implemented additionally.

Weighted value. At the absorb phase, this parameter's value is auto-scaled proportionally to some other parameter's value.

Decaying value. At each timestep, this value is automatically divided by some factor.

Random value. The parameter is generating new value each timestep, using some PRNG algorithm. We are found KISS algorithm currently fits best for our purposes.

Genetic value. At the absorb phase, this value is calculated as crossover of all incoming genomes. It may also mutate randomly as needed.

2.10 Optimization Patterns

Our framework will generate a code for GPU, and there are several patterns that could help us to produce more effective code.

1. Parameters should be "lazy". We must read them from memory, only if we need them in kernel code further. They also could be completely discarded from the memory in the case if we're never actually using them.

2. Buffered states should also be "lazy", in the same manner as parameters: no need to read unnecessary buffers each time.

3. Parameters should be packed in memory in a most compact way, there should be no unused bits in a cell's representation in memory. This will reduce both the memory access time and the memory space per each cell.

4. A special case, when we are using "gate" and interacting only with the neighbors in the direction of the gate, would be reduced to the less number of buffers, down to a single buffer.

5. Loops should be manually unrolled when it is possible.

6. Sorting networks should be used as the source for sorting algorithms.

This list is most likely not complete, we will extend it in future.

Acknowledgments

I would like to give special thanks to Alexey Shchepin, for the optimization ideas, testing, constructive criticism, advices in LaTeX, and corrections to this document.