Information Visualization of Multi-dimensional Cellular Automata using GPU Programming

Stéphane Gobron, Daniel Mestre
Movement and Perception Laboratory, Federative Research Institute Marey
UMR 6152, CNRS, University of Aix-Marseille II, Luminy, F-13009 Marseille, France.
{stephane.gobron, daniel.mestre}@univmed.fr

Abstract

We propose a method for generating all possible rules of multi-dimension Boolean cellular automata (CA). Based on an original encoding method and the programming of graphical processor units (GPU), this method allows us to visualize the CA information flow in real-time so that emerging behaviors can be easily identified. Algorithms of first and von Neumann neighborhood second degrees are detailed with their respective Fragment Shaders programs. As symmetrical CA rules are especially useful in many research fields, we propose an encoding technique to automatically derive their codes; we then apply this technique to identify the 4096 possible cases for surface CA. To show the efficiency of our model a set of converging global behaviors are listed and described. In the last part of the paper we present methods for developing Moore neighborhood in two and in three dimensions. Finally we discus issues concerning computation and the visualization of non-Boolean and higher dimension CA.

Keywords—**Cellular automaton, information visualization, emerging behavior, real-time imaging, GPU programming, computer graphics**.

1 Introduction

This paper focuses on the cellular automaton (CA) multi-dimensional representation and behavior identification in the visualization and information domains.

In any natural phenomena, the complex behavior and diversity that can be observed at a macroscopic level are essentially due to the fact that there are numerous particles in continual interaction. Furthermore, when working at one level, a description in terms of the preceding level can unify phenomena that were previously considered as individual cases. In fact, CA embodies such multi-level observations as it consists, at low levels of a set of change of states –called ”rule”–and at high levels we can observe an emerging behavior. CA is a powerful tool which only refers to states, rules, and time: CA rule is a set of change of states for all possible states of every cell in its neighborhood. Hence, in order to simulate multi-behavior surface effects, CA appears to be a relevant prospect field of research. Unfortunately, CA emerging phenomena are very often impossible to predict by theoretical approaches [23]. To help researchers identify these emerging behaviors, we propose real-time graphical visualization tools which allow massive computational CA change of states for fast renderings. Let's first present the literature and the structure of the paper.

1.1 Background

This study belongs to the fields of CA and computer graphics (CG) applied to multidimensional information visualization (IV).

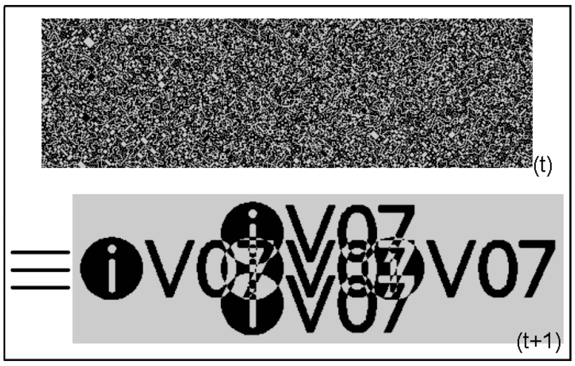


Figure 1: Information visualization of a 2D-CA: an example of equivalent rule (symmetrical rule #1370) at step t and $t+1$; see section 4 for details.

Literature dealing directly with computer graphical CA has become more and more prevalent, especially since GPU programming became popular [3, 21] and following the publishing of a fundamental book by Stephen Wolfram [23] which covers all practical aspects of CA. If we assume that every structure with interacting elements is a type of CA, references will actually be too numerous to mention. That is why we refer the reader to [8] where a non exhaustive list of well-known references in fields relative to formal, classical, and applied CA is provided, as well as CG and GPU programming. Nevertheless, with respect to CA, and GPU here is a selection of very recent publications on various research directly related to IV.

- Theoretical CA neighbors and global behavior [12];
- CA and IV: Social behavior [22]; Species competition and evolution [2]; Biology: tumor growth [18], bacteria swarming [24], Epidemiology prediction [16]; Image compression [14].
- GPU and IV: Surgery [11]; Graphical shadow research [25]; realistic local illumination and multilayered thin film interference [4]; fast fourier transform [5].

We did not find however any paper referring to information visualization of CA based on GPU as is our approach in this paper.

1.2 Overview

This paper is organized as follows. For non-specialists of OpenGL Shading Language (GLSL) [15, 13] (URL: [20]), we begin in section 2 by presenting GPU programming, CA general concepts, and the software and hardware contexts. Section 3 presents classical first dimensional Boolean CA including twenty of its most famous rules. Section 4 describes how to extrapolate to direct neighborhood in 2D, emphasizing a technique to identify symmetrical rules. A set of non-symmetrical and symmetrical graphical results are then presented. Key ideas concerning higher dimensions and non Boolean CA are detailed in sections 5. Finally, section 6 concludes the paper and outlines some related future work.

2 Introduction to GPU and CA

In order to increase CA computations we use a GPU programming based approach. The following section introduces concepts relative to graphical processor units (GPU) and cellular automaton (CA) which will be used in this paper.

2.1 Graphical processor unit

Over the last few years, GPU programming has increased CPU computational capacity by a factor of about twenty [3, 9, 19, 7]. The formal Shaders algorithms used in this paper, is based on a previous work on artificial retina simulation [7]. As the current paper focuses on the information visualization of CA, only main ideas underlying Shaders software development are presented.

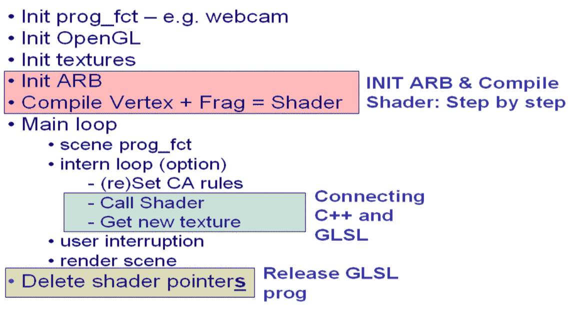
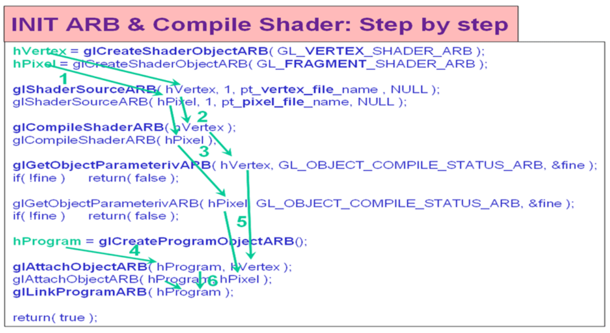


Figure 2: Basic CG-engine algorithm using GLSL.

Figure 2 presents a basic OpenGL graphical engine including the three key sections which allow the C++ program (CPU) to communicate with the Shaders program (GPU)–in our case OpenGL Shading Language (GLSL). A solution to the non-obvious ARB initialization calls (shown in light pink figure 2) is proposed in figure 3 and consists of the following six steps:

1. associate a Vertex and Pixel Shaders pointers to source files;
2. compile Vertex and Pixel Shaders programs;
3. if compilation is fine...
4. ...and from a unique Shaders program pointer...
5. ...then attach Vertex and Pixel pointers to the Shaders program;
6. finally, link them all together.

The main loop (see the light blue zone in figure 2) which allows cellular buffers to be consider as textures and copied back to matrices, is detailed in [7]. Notice that once C++ and Shaders programs are linked, data transmission can be performed. However, to maintain the high performance of the algorithm, the communication and the number of Shaders-code lines have to be kept to a minimum. Developing a single Shaders program –for each type of CA—allowing any possible solutions to be visualized,


INIT ARB & Compile Shader: Step by step

```
hVertex = glCreateShaderObjectARB( GL_VERTEX_SHADER_ARB );
hPixel = glCreateShaderObjectARB( GL_FRAGMENT_SHADER_ARB );

glShaderSourceARB( hVertex, 1, pt_vertex_file_name , NULL );
glShaderSourceARB( hPixel, 1, pt_pixel_file_name, NULL );

glCompileShaderARB( hVertex );
glCompileShaderARB( hPixel );

glGetObjectParameterivARB( hVertex, GL_OBJECT_COMPILE_STATUS_ARB, &fine );
if( !fine )     return( false );

glGetObjectParameterivARB( hPixel, GL_OBJECT_COMPILE_STATUS_ARB, &fine );
if( !fine )     return( false );

hProgram = glCreateProgramObjectARB();

glAttachObjectARB( hProgram, hVertex );
glAttachObjectARB( hProgram, hPixel );
glLinkProgramARB( hProgram );

return( true );
```

Figure 3: Details of the linking between CPU and GPU.

was one of the challenges solved in this paper (see rule and CA-key code definitions in subsection 2.3). Finally, (see the light green zone in figure 2) Shaders pointers can be released by *detaching* and then *deleting* the Shaders program using respectively the *glDetachObjectARB()* and *glDeleteObjectARB()* ARB functions.

2.2 Hardware, software, and rendering technique

All algorithms presented in this paper were developed in C++ on Microsoft Windows XP-pro using $MSVC$++, the graphical library OpenGL [15], and the OpenGL Shading Language ($GLSL$) [13]. CA and corresponding graphical results were computed and rendered using 3.10^5 cells using a Xeon^{TM} CPU (3.19 GHz) with 3 Gb of RAM (note that for this application 512 Mb is largely sufficient) and a NVidia Quadro 3500 graphical card. Indeed, to highlight the efficiency of the method presented in this paper, our technique offers real-time performances on cheap and easily available computing systems.

2.3 CA vocabulary, concept and definition

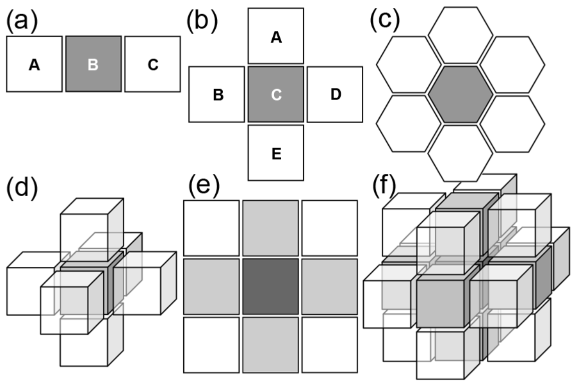


Figure 4: Multidimensional CA geometric representations: (a) 1DCA; (b, c, and e) 2DCA; (d and f) 3DCA.

Here is a short introduction to some important CA concepts such as notions of CA type, neighborhood, rule, change of state, and CA-key code.

Cellular automaton types are presented in figure 4 which illustrates most 1^{st}, 2^{nd}, and 3^{rd} dimensional CA. A CA-type depends on its dimensional domain and its number of neighbors. For instance, we can see that for the 2^{nd} dimension, three types of CA neighborhoods exist:

- figure 4(b) the classical grid direct neighborhood –also called *van Neumann* neighborhood;
- figure 4(c) the hexagonal grid neighborhood;
- figure 4(e) the classical grid indirect neighborhood –also called *Moore* neighborhood.

As previously said, a CA rule is a set of change of states where all possible states of every cell in the neighborhood is defined. For instance, for a unique set of Boolean values of φ_1, φ_2,... , φ_8 the truth table 1 represents one rule of the Boolean 1D-CA (also note $1D_bCA$). A CA-key code is the encoding of a rule, $e.g.$ one of the fractal emergent behavior in figure 5 has a CA-key code of 146. The CA-key code –instantiating a CA-rule—is a fundamental notion used in this paper. In fact, to be able to produce a general method and as data transmission between CPU and GPU can only be done by a limited number of parameters (which varies depending on the generation of the graphical card), we need to use the CA-key and find a way to decode it in the Shaders program. The encoding of this key is proportional to the number of possible rules (ρ), which unfortunately increase exponentially (see sections 4 and 5). Hence, considering that the state of a cell C remains in a discreet domain of n possible cases, (for instance, n would be 2 for Boolean, 256 for *unsigned char* integer), the number of neighbors of C is N. Then, the maximum value of the key is shown by equation 1 where C represents the number of possible combinations of the neighborhood states.

$$\rho = n^C \text{ with } C = n^{(N+1)} \Rightarrow \rho = n^{n^{(N+1)}} \qquad (1)$$

For more details concerning general and theoretical concepts related to CA, we strongly advice Stephen Wolfram's pedagogic book [23]. Now that basic concepts and the hardware context have been defined, we move on in the following section to propose a model for computing any rule of 1D-CA enabling the visualization of the corresponding data flow.

3 One dimensional cellular automata

Unidimensional CA defined as a set of c cells C_x are represented as a 2D-matrix –often as a textured triangular picture when using a single root. Every line is related to each change of state which occur over time. Therefore, for each row y, the Boolean states of C –which we symbolize by $\beta(C_x)$—at a time $t+1$ has to satisfy:

$$\beta(C_x)@(t+1) = \beta(C_{(t+1)[x,y+1]}) \qquad (2)$$

A less formal way to interpret equation 2 is to say that a row y represents the y^{th} change of state at time t which means that y and t are equivalent in this dimension –see figure 5. On this basis, the next sub-section 3.1 formally presents the Pixel Shaders structure of the one dimensional Boolean CA.

β_{C_A}	β_{C_B}	β_{C_C}	$\beta_{C_{x,y}}$
0	0	0	φ_1
0	0	1	φ_2
0	1	0	φ_3
...	...	...	φ_i
1	1	1	φ_{2^3}

Table 1: 1D-Boolean CA truth table.

3.1 GPU-accelerated $1D_bCA$ Fragment Shaders model

Considering a root image $P(\Delta_x, \Delta_y)$ where Δ_x and Δ_y are respectively column and row numbers, P can only be interpreted in the GPU as a texture $T(0.0 \ldots\ 1.0,\ 0.0 \ldots\ 1.0)$. Then for any Φ $1D_b$CA rule, we need to find the Boolean change of state corresponding to the next row. In particular, for any cell $C_{[x,y+1]}$ at time t, we have:

$$C_{(t+1)[x,(y+1)]} = \Phi(C_{(t)[x-1,y]}, C_{(t)[x,y]}, C_{(t)[x+1,y]}) \qquad (3)$$

Using equation 1, the current possible rule ρ is $2^{2^3} = 256$. Among these 256 possible rules, a rule Φ can also be interpreted as a set of 8 Boolean values (φ):

$$\forall \Phi_i \in (0..255)_i \mid \exists \Phi_b \in ([0..0]..[\varphi_1..\varphi_8]..[1..1])_b \qquad (4)$$

As texture has to be used with length and height ranging from 0.0 to 1.0, we must define δ_x and δ_y as the texture thickness of one line and one row, so that:

$$\delta_x = \frac{1}{\Delta_x} \text{ and } \delta_y = \frac{1}{\Delta_y} \qquad (5)$$

Then we can find the texture coordinates $\tau()$ of the three cells C_A, C_B, C_C (cell neighborhood, respectively the upper left, central, and right cells) necessary for the change of state using a vectorial translation:

$$\forall C_{x,y} : \begin{cases} \tau(C_A) = \tau(C_{x,y}) + (\overrightarrow{-\delta_x, -\delta_y}) \\ \tau(C_B) = \tau(C_{x,y}) + (\overrightarrow{0.0, -\delta_y}) \\ \tau(C_C) = \tau(C_{x,y}) + (\overrightarrow{\delta_x, -\delta_y}) \end{cases} \qquad (6)$$

From that point, we use the Boolean state function $\beta()$ defined on a cell C_x as:

$$\beta(C_x) = (\Sigma(C_x.R, C_x.G, C_x.B) < 1.5)_b \qquad (7)$$

In the domain of computer graphics (CG) [6], the most common representation of colorimetry is the $RGB\alpha$ four components. In $1D$, first the alpha parameter is not being used, and second the 3 other parameters represent a waist of memory, $i.e.$ three real numbers to encode a Boolean... We will see in section 5 that $RGB\alpha$-textures can be used in a much more efficient way. Thus to find the final new state of a cell C, we only have to use the truth table 1 with equation 7.

Finally, we can deduce equation 8 and therefore, the CA-key code of $1D_bCA$ can be transmitted to the Shaders program in one byte T_b.

$$\beta_{C_{x,y}} = \varphi_{(\beta_{C_A}\beta_{C_B}\beta_{C_C})_b} \qquad (8)$$

3.2 1D results and observations

Our $1D_bCA$ model has a constant computational rate of about 876 millions change of states per second.

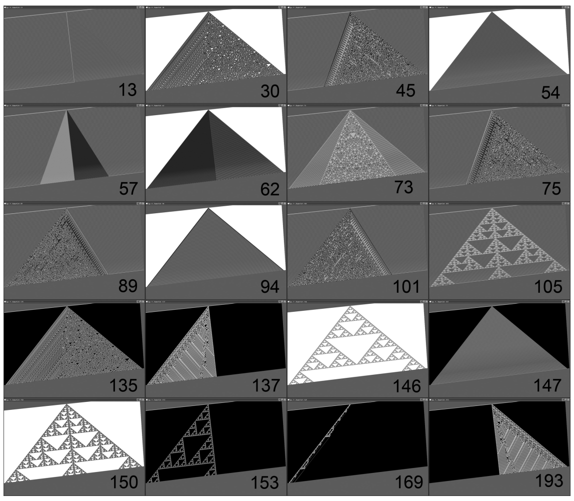


Figure 5: Twenty of the most popular 1D Boolean CA –also called $1D_bCA$; here the x-axis represents the automate cells and y-axis the time.

Figure 5 presents various examples of unidimensional Boolean CA with their respective rule number using a texture of 640 by 480. The four classes of feedback behavior (see [23]) that CA can exhibit are shown, $i.e.$: convergence to a fixed, homogeneous state ($e.g.$ rule 54); a chaotic aperiodic structure ($e.g.$ rule 101); a simple separated periodic pattern ($e.g.$ rule 105); and a recursive periodic but acyclic behavior ($e.g.$ rule 193). Notice that these results appear slightly inclined as they were rendered in 3D. In the following section 4 we present a solution to visualize second degree direct neighborhood data flow, and eventually identify emerging behaviors with their respective CA-key codes.

4 Two dimensional cellular automata

Following from the previous section on $1D_bCA$ our next goal is to design a general Shaders algorithm to visualize any rule of Boolean CA in two-dimension. The structure of this section is similar to the previous with subsection 4.1 presenting a formal algorithm of $2D_bCA$, subsection 4.2 a set of six examples. In the domain of real-time imaging, symmetrical CA can be very useful (CG surface texture [17, 10], image analysis [7]) that is why in the last two subsections 4.3 and 4.4 we propose a specific technique to automatically derive those symmetrical rules. Three rules $\rho_{2D_{vN}}$, ρ_{2D_H}, and ρ_{2D_M} can be associated to the van Neumann, hexagonal, and Moore neighborhoods –see figure 4(b), (c), and (e). From equation 1 we can deduce that:

- $\rho_{2DvN} = 2^{2^5} = 4294967296 \simeq 4.3$ billion, and a key of 4 bytes;
- $\rho_{2DH} = 2^{2^7} \simeq 18$ billion billion, and a key of 16 bytes;
- $\rho_{2DM} = 2^{2^9} \simeq 1.34\ 10^{154}$ rules, and a key of 64 bytes.

β_{C_A}	β_{C_B}	β_{C_C}	β_{C_D}	β_{C_E}	$\beta_{C_{x,y}}$
0	0	0	0	0	φ_1
0	0	0	0	1	φ_2
0	0	0	1	0	φ_3
...	...	...	...	...	φ_i
1	1	1	1	1	φ_{25}

Table 2: $2D_bCA$ van Neumann truth table.

β_{C_A}	β_{C_B}	$\beta_{C_{x,y}}$
0	0	T_{b_1}: $\varphi_{1..8}$
0	1	T_{b_2}: $\varphi_{9..16}$
1	0	T_{b_3}: $\varphi_{17..24}$
1	1	T_{b_4}: $\varphi_{25..32}$

Table 3: Another way to encode $2D_bCA$ van Neumann using four bytes.

4.1 GPU-accelerated $2D_bCA$ Fragment Shaders model

As hexagonal grids are incompatible with our current CG texture method, and Moore neighborhood requires too much data to be transmitted to the Shader program, we focus on the study of direct neighborhood grids (see section 5 for higher degree CA). Analogous to Φ_{1D} shown in equation 4, the term $\Phi_{2D_{vN}}$ has to be encoded by a set of Boolean, only that time with 64 bits: $[\varphi_0\varphi_1..\varphi_{64}]$. CG texture δ_x and δ_y as well as Boolean state function $\beta()$ remain the same. Of course, in $2D$, x and y are required for information visualization. Emerging behaviors can directly be identified by graphical texture forms, shapes, or silhouettes. Behaviors involving time then have to be evaluated through animation sequences. A, B, D, E being the four neighbors of the central cell C, corresponding texture coordinates $\tau()$ (see equation 6 and figure 4(b)) are now defined by:

$$\tau_{t+1}\begin{bmatrix} & C_A & \\ C_B & C_C & C_D \\ & C_E & \end{bmatrix} = \tau_{(t)}(C_c) + \begin{bmatrix} & \overrightarrow{(0,-\delta_y)} & \\ \overrightarrow{(-\delta_x,0)} & \overrightarrow{0} & \overrightarrow{(\delta_x,0)} \\ & \overrightarrow{(0,\delta_y)} & \end{bmatrix} \quad (9)$$

From the truth table 2, we can notice that the transmitted CA-key code $[T_{b_1}, T_{b_2}, T_{b_3}, T_{b_4}]$ corresponds to each of the four combinations of φ_A and φ_B. Hence, we can reuse a single byte decoder similar to the one developed for $1D - CA$ –see table 3. This method saves around 75% of the Shaders lines and therefore the algorithm is three to four times faster.

4.2 General case results

Figure 6 proposes six examples of van Neumann $2D_bCA$ initialized with a single bit set to "1" placed in the center of the texture. However based on very simple changes of states, these CA illustrate how complex the emerging behaviors can become: (a) generates a silhouette of a quarter circle; (b) presents a complex fractal shape based on a recursive pattern with empty triangles in the upper left corner and non-ordered cellular sets on

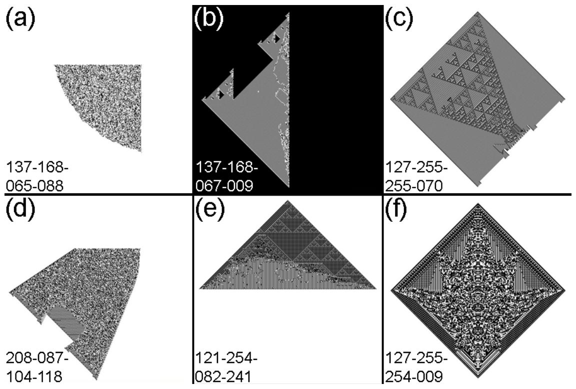


Figure 6: Example of $2D_bCA$ asymmetrical rules generated from a single root.

the right; figure 6 (c) shows acyclic fractal patterns texture and regular fractal in silhouette; (d) belongs to a family of polygonal shape generation; (e) demonstrates that order and chaos can co-exist in a triangle silhouette; (f) belongs to a family of leaf shape generation.

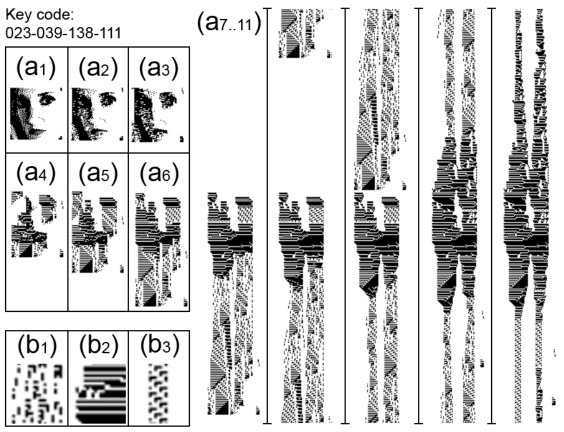


Figure 7: An example of $2D_bCA$ asymmetrical rules generated from a complex root: fluid/sedimentation-like simulation

Figure 7 illustrates a example of fluid and sedimentation like simulation. Based on an initial face texture pattern shown figure 7(a_1), the first steps (a_2) to (a_6) show melting and stalactite-like sedimentation. Then from (a_7) to (a_{11}) three information behaviors can be visualized: half-dynamic (see figure 7(b_1)), static (stalactite and stalagmite-like –see (b_2)),and pure cyclic (pure fluid-like–see (b_3)).

4.3 Identifying symmetrical rules

As previously explained, although symmetrical rules could be particularly interesting in many research fields, identifying the CA-key codes can be difficult. In nature, symmetrical pattern can easily be found, for instance snowflakes as shown figure 8. Thus we have developed a method to identify central symmetry. We identify the 12 non-redundant change of states ($S_1..S_{12}$)

Figure 8: Examples of real snowflakes presenting symmetrical patterns.

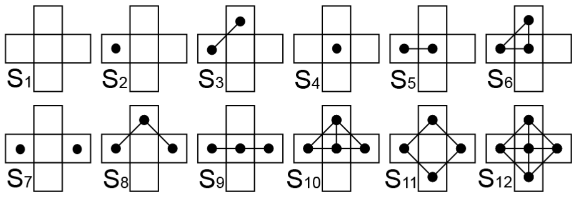


Figure 9: All possible cases of central symmetry in direct $2D_bCA$.

from truth table 2; a graphical interpretation of S_i is presented figure 9.Then each bit of $[\varphi_1..\varphi_{32}]$ correspond to the sequence presented table 4. Reciprocally, we have now to derive for any $[S_1..S_12]$ symmetrical key, its corresponding four byte key (T_a, T_b, T_c, T_d). As every byte is encoded on sequence of eight bits (from higher to lower powers of 2, $e.g.\ T_b \equiv [b_8b_7b_6b_5b_4b_3b_2b_1]_b$), we can derive the table 5. Finally, based on table 5, the general four bytes key code for any symmetrical rule can be deduced as shown in table 6.

4.4 General case results

Our direct-neighbor $2D_bCA$ model has a constant computational rate of about 189 million change of states per second. As demonstrated in the above sub-section, only 4096 symmetrical rules exist. These can be divided into three fields of application: image analysis, surface convection detection, and pattern memory[1]. Based on triple roots texture, figure 10(a) illustrates how powerful a CA tool can be in finding contours. The first contour CA is proposed in figure 10(b) converging after 5 steps to (c) –symmetrical key #2254. The second contour CA is presented in (d) –and (e) being its Boolean complement for comparing with (b)—directly converging via a bit inversion –symmetrical key #0069. Notice that the first filter proposes the inside silhouette, and that the second one an outside silhouette. Figures 10(f) to (j) propose a more complex contour transformation such that only horizontal, vertical, and diagonal patterns remain –symmetrical key #0446.

Another complex behavior can be observed in figure 11: surface convection detection –symmetrical key #1328. In this complex CA, we can observe that after filling the entire texture of

[1]These figures being composed of very fine blank and white dots, behaviors should be observed at very high resolution using electronic version or on our web site [1].

$(T_{b1}:)$	$[S_1$	S_2	S_2	S_3	S_4	S_5	S_5	S_6
$(T_{b2}:)$	S_2	S_3	S_7	S_8	S_5	S_6	S_9	S_{10}
$(T_{b3}:)$	S_2	S_7	S_3	S_8	S_5	S_9	S_6	S_{10}
$(T_{b4}:)$	S_3	S_8	S_8	S_{11}	S_6	S_{10}	S_{10}	$S_{12}]$

Table 4: Analyzing symmetrical properties of truth table 3 with patterns proposed figure 9.

S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}
a_8	a_7	a_5	a_4	a_3	a_1	b_6	b_5	b_2	b_1	d_5	d_1
	a_6	b_7		a_2	b_3	c_7	c_5	c_3	c_1		
	b_8	c_6		b_4	c_2		d_7		d_3		
	c_8	d_8		c_4	d_4		d_6		d_2		

Table 5: Associating symmetrical keys to bits of the byte-based CA-key code.

	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7
T_a	S_6	S_5	S_5	S_4	S_3	S_2	S_2	S_1
T_b	S_{10}	S_9	S_6	S_5	S_8	S_7	S_3	S_2
T_c	S_{10}	S_6	S_9	S_5	S_8	S_3	S_7	S_2
T_d	S_{12}	S_{10}	S_{10}	S_6	S_{11}	S_8	S_8	S_3

Table 6: Final four bytes CA-key code T reconstructed from symmetrical a S key of 12 bits .

random-like cellular states, only few unstable strings-like region remain, always converging to minimize their respective length. As the CG-texture used here is not limited in dimension (*i.e.* $max(x, y) + 1 = 0$ and $min(x, y) - 1 = max(x, y)$), depending of the root, the pattern will converge to x or y perfectly straight lines –as shown figure 11(l). However quite useless in two-dimension, we believe that this CA could be extremely useful in three-dimension. The front page figure 1 and figure 12 present a CA that can memorize and repeat patterns with a cycle of 16 steps. This behavior is quite spectacular as shapes seem to disappear and become random noise. Nevertheless, the enlarged region of screen shot 12(f) shows the cells are not ordered randomly but rather in a maze-like pattern. An animation of this phenomena as well as other CG-pictures can be seen at the URL referred in footnote 1.

5 Higher Dimensions and non-Boolean CA

We have mainly explored first and direct neighborhood second dimension CA and found many examples of interesting behavior. In this section we present issues and propose some solutions to extend our models to higher dimensions and non-Boolean CA.

5.1 Hexagonal grid and third dimension

Even if their respective graphical geometric structures are different, hexagonal and 3^{rd} dimension direct neighborhood CA (see figure 4(c and d) generate an equivalent number of key rule (*i.e.*: $\rho_{2DH_{bCA}} = \rho_{3DvN_{bCA}} = 2^{2^7}$) which is around 18 billion billion. Following from what has been shown in this paper, to identify interesting behaviors, we have to decrease the number of key rules by considering symmetrical rules of those CA. For hexagonal grid based CA, we believe that a more natural cellular structure will lead to promising studies. Concerning direct neighborhood 3D-CA, nowadays maximum graphical card memory is typically $768Mb$ and therefore as 3D-textures can directly be instantiated a 900 cells side cube can be generated –which can produce sufficient renderings. For 2D Moore neighborhood and 3D CA, even Boolean cells produce a real challenge as a the number of key rule code is: $3DM_bCA$: 2 pow(2 pow(19)) = 2 pow(524288), round e^{9215} which is hard to conceive. Therefore for as long as this issue remains unresolved, there is no point in designing a GPU-CA model for such dimension or higher.

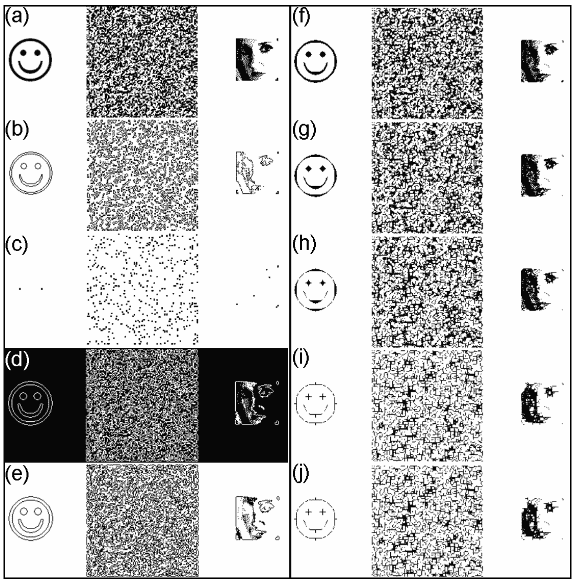


Figure 10: Example of symmetrical 2D-CA contour behavior: (a) initial figure; (b) first step of 2DCA-contour rule (sim-key #0069) converging at step four in (c); (d) 2DCA-contour rule (sim-key #2254) converging at first step with a color inversion –(e) being its boolean complement; (f) to (j) 2DCA-contour (sim-key #0446) where only the four direction lines remain.

5.2 Non-Boolean CA

Indeed, non-Boolean CA is not really an issue as cellular values can be interpreted as thresholds. Of course, key-rule code numbers will increase exponentially but we can imagine encoding the four texture $RGB\alpha$ parameters in a more efficient way to simulate cellular thresholds.

6 Conclusion and future work

We have presented an original method to visualize CA of different dimensions to identify their emerging behavior. After explaining the use of CA and the need for such a tool, we summarize the technique of GPU programming and detail how to encode 1D and 2D von Neumann CA. In particular, we have presented an original way of sorting symmetrical 2D CA. To show the capabilities of our model, we have presented a non restricted set of convenient rules simulating fractals, drop and string like sets, and contour identification. We then discussed ways and issues to implement non Boolean and higher order CA. Based on the ideas developed in this paper, we are hoping to identify other emerging behaviors and we are also currently developing a general method for real-time information visualization of 3D-CA. In the field of computer graphics (CG) this research can help model an accelerated and generalized 3D-surface CA [8] –*i.e.* where any symmetrical or asymmetrical CA could be directly encoded without developing specific code. Furthermore, virtual reality (VR) being

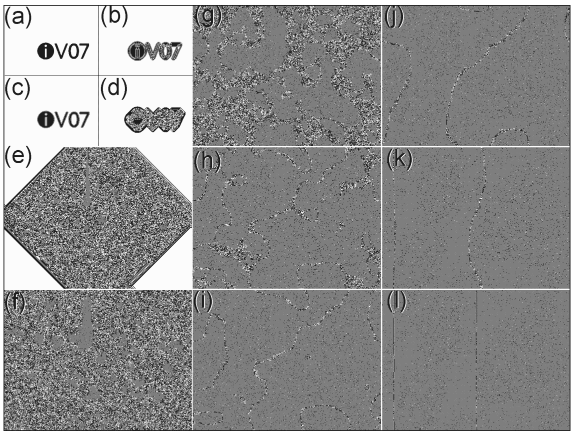


Figure 11: 2D-CA Strings-like behavior (sym-key #1328): (a) to (d) initial steps; (e) merging of stable regions; (f) to (g) stable region becomes dominant as "strings" appears; (h) to (l) all strings decreases as remains only unlimited string regions.

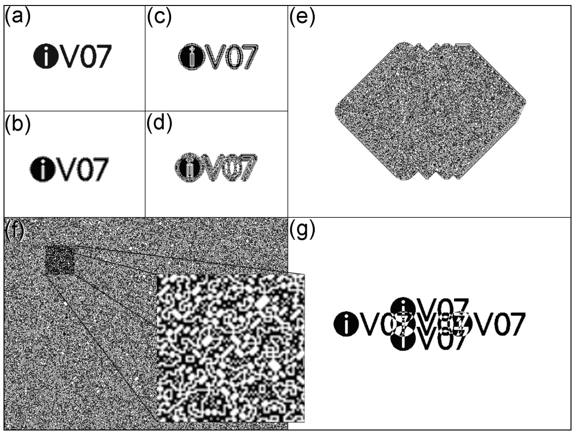


Figure 12: 2D-CA (sym-key #1370) for the following steps: (a) initial figure; (b,c,d) first, second, and fifth steps; (e) step 255 mature growth (d); (f,g) respectively step 511 and 512 –also shown figure 1.

one of the major topics of our research team, we are currently designing CA models interacting with scenes and end-users as well. For instance, sceneries based on real-time weathering textures or interactive intuitive CA-based menu for helping parametrization.

Acknowledgment

We would like to thanks Mr Hervé Bonafos, Mr Jean-Marie Pergandi, and Mr Cédric Goulon, for their technical support, daily services, and being part of our great CS team! Special thanks are also due to Mrs Thelma Coyle for her kindness and the proof-reading of the text.

References

[1] CNRS-Univ.Aix-Marseille II IFR Marey Additionnal pictures, S. Gobron. http://www.laps.univ-mrs.fr/~gobron/, 2007.

[2] B. Chopard and D. Lagrava. A cellular automata model for species competition and evolution. In *The International Conference on Cellular Automata for Research and Industry (ACRI06), Springer*, pages 94–103, Perpignan, France, september 20-23 2006.

[3] S. Druon, A. Crosnier, and L. Brigandat. Efficient cellular automaton for 2d / 3d free-form modeling. *Journal of WSCG*, 11(1, ISSN 1213-6972), 2003.

[4] R. Durikovic and R. Kimura. Gpu rendering of the thin film on paints with full spectrum. In *Tenth International Conference on Information Visualisation (IV'06), IEEE Computer Society*, pages 751–756, London, UK, 5-7 July 2006.

[5] O. Fialka and M. Cadik. Fft and convolution performance in image filtering on gpu. In *Tenth International Conference on Information Visualisation (IV'06), $IEEE$ Computer Society*, pages 609–614, London, UK, 5-7 July 2006.

[6] J.D. Foley, A.F. van Dam, K. Stephen, J.F. Hughes, and R. Phillips. *Introduction to Computer Graphics, principles and practice*. Addison-Wesley, 2^{nd} edition, 1993. 1175 pages.

[7] S. Gobron, F. Devillard, and B. Heit. Retina simulation using cellular automaton and GPU programming. *Machine Vision and Applications Journal, Springer Berlin-Heidelberg*, pages 1432–1769, 2007.

[8] S. Gobron and D. Finck. Generating surface textures based on cellular networks. In *The Geometric Modeling and Imaging international conference (GMAI06), IEEE Computer Society*, pages 113–120, Londres, UK, july 5-7 2006.

[9] M. Harris. Implementation of a cml boiling simulation using graphics hardware. In *CS. Dept, UNCCH, Tech. Report*, volume 02-016, 2003.

[10] S. Lefebvre, S. Hornus, , and F. Neyret. All-purpose texture sprites. In *INRIA Research Report 5209*, INRIA, Grenoble, France, 2004.

[11] Jesper Mosegaard, Peder Herborg, and T.S. Sorensen. A gpu accelerated spring mass system for surgical simulation. *Studies in Health Technology and Informatics*, (978-1-58603-498-6):342–348, 2005.

[12] H. Nishio. How does the neighborhood affect the global behavior of cellular automata? In *The International Conference on Cellular Automata for Research and Industry (ACRI06), Springer*, pages 94–103, Perpignan, France, september 20-23 2006.

[13] R.J. Rost. *OpenGL Shading Language*. Addison Wesley Professional, 2^{nd} edition, 2006. 800 pages.

[14] C. Shaw, S. Das, and B.K. Sikdar. Cellular automata based encoding technique for wavelet transformed data targeting still image compression. In *The International Conference on Cellular Automata for Research and Industry (ACRI06), Springer*, pages 94–103, Perpignan, France, september 20-23 2006.

[15] D. Shreiner, M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide: the official Guide to learning OpenGL v2.0*. Addison Wesley Professional, 1st edition, 2005.

[16] R. Slimi and S.El Yacoubi. Spreadable probabilistic cellular automata. In *The International Conference on Cellular Automata for Research and Industry (ACRI06), Springer*, pages 94–103, Perpignan, France, september 20-23 2006.

[17] Cyril Soler, Marie-Paule Cani, and Alexis Angelidis. Hierarchical pattern mapping. In *ACM, SIGGRAPH'02 Conf. Proc.*, July 2002.

[18] P. Topa. Towards a two-scale cellular automata model of tumour-induced angiogenesis. In *The International Conference on Cellular Automata for Research and Industry (ACRI06), Springer*, pages 94–103, Perpignan, France, september 20-23 2006.

[19] J. Tran, D. Jordan, and D. Luebke. New challenges for cellular automata simulation on the gpu. *www.cs.virginia.edu*, 2003.

[20] Shaders tutorials. http://www.3dshaders.com/joomla/, 2007.

[21] L. Wang, X. Wang, X Tong, S. Lin, S. Hu, B. Guo, , and H-Y. Shum. View-dependent displacement mapping. In *SIGGRAPH'03 Conf. Proc.*, volume 22, pages 334–339, 2003.

[22] J. Was, B. Gudowski, and P.J. Matuszyk. Social distances model of pedestrian dynamics. In *The International Conference on Cellular Automata for Research and Industry (ACRI06), Springer*, pages 94–103, Perpignan, France, september 20-23 2006.

[23] S. Wolfram. *A new kind of science*. Wolfram Media Inc., 1^{st} edition, 2002. 1197 pages.

[24] Y. Wu, N. Chen, M. Rissler, Y. Jiang, D. Kaiser, and M. Alber. Ca models of myxobacteria swarming. In *The International Conference on Cellular Automata for Research and Industry (ACRI06), Springer*, pages 94–103, Perpignan, France, september 20-23 2006.

[25] Y. Xiao and J. Yicheng. Gpu based real-time shadow research in large ship-handling simulator. In *Tenth International Conference on Information Visualisation (IV'06), IEEE Computer Society*, pages 585–590, London, UK, 5-7 July 2006.