GPU Accelerated Computation and Visualization of Hexagonal Cellular Automata

Stéphane Gobron[1], Hervé Bonafos[2], and Daniel Mestre[1]
[1] Institute of Mouvement Sciences, CNRS, Marseille, France
{stephane.gobron, daniel.mestre}@univmed.fr, [2] herverv@aol.com

Abstract

We propose a graphics processor unit (GPU)-accelerated method for real-time computing and rendering cellular automata (CA) that is applied to hexagonal grids. Based on our previous work [9] –which introduced first and second dimensional cases– this paper presents a model for hexagonal grid algorithms. Proposed method is novel and it encodes and transmits large CA key-codes to the graphics card and consequently, this technique allows to visualize the CA information flow in real-time to easily identify emerging behaviors even for large data sets. To show the efficiency of our model we first present a set of characteristic hexagonal behaviors, and then describe computational statistics for central processing unit (CPU) and GPU on a set of different hardware and operating system (OS) configurations. We show that our model is flexible and very efficient as it permits to compute CA close to a thousand times faster than classical CPU methods. Additionally, free access is provided to our downloadable software for hexagonal grid CA simulations.

Keywords—**Hexagonal cellular automaton, GPU-accelerated computation, Digital imaging, Real-time rendering, Emerging behavior**.

1 Introduction

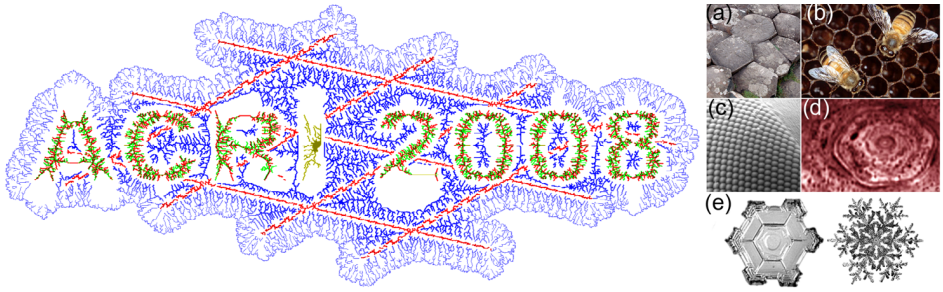

Fig. 1. [left] Example of hexagonal CA ($2D_b^hCA$) applied to the conference logo (101 steps); [right] Example of hexagonal structures found in nature: (a) stones found in the *Giant's Causeway*, Ireland; (b) honeycomb; (c) eye of a fly; (d) hexagon spotted over Saturns surface; (e) snowflakes.

This paper belongs to a series of papers concerning the computation and rendering real-time boolean multi-dimensional cellular automata (CA). Based on our previous approach proposing first dimension and second *von Neumann* dimension CA [9], the current work focuses on 2D hexagonal structures ($2D_b^hCA$).

As shown by the literature described in next Subsection 1, CA is a powerful tool that can be used in a wide variety of domains. Unfortunately, in CA, emerging phenomena are often impossible to predict by theoretical approaches [20]. To help researchers compute complex phenomena –such as species competition and evolution [3]– faster, or identifying emerging behaviors almost instantaneously, we propose real-time graphical visualization

models. A demonstration of the capabilities of our approach is available as a freeware provided on first author web site. In the following subsections we first present the literature focusing on hexagonal CA and then describe the structure of the paper.

Background This study belongs to the research fields of CA and computer science (CS) specialized in real-time digital imaging applied to visualization of multidimensional massive set of cells. Computer graphics literature dealing directly with CA has become more and more prevalent, especially since GPU programming became popular. There has been a growing interest in the field following the publishing of a fundamental textbook by Stephen Wolfram [20] in 2002, covering all practical aspects of CA. Assuming that every structure with interacting elements is a type of CA, references will actually be too numerous to mention. Moreover this paper is a direct extension of our introduction to multi-dimensional model [7] where the reader can find a non exhaustive list of references in fields relative to formal, classical, and applied CA is provided, as well as CG and GPU programming ([10]). Within the scope of this paper, we will focus on hexagonal CA (hCA) references.

As shown in Figure $1^{[right]}$, the hexagonal structure is very common in nature. Indeed, due to one of its properties regarding neighborhood cells (central symmetry), it offers a natural structure for simulation. This has inspired researchers to utilize hexagonal CA in a wide range of applications (hCA): reaction-diffusion system [1]; modeling forest fire [22,4]; simulation of debris flows [11]; physics-based simulation of material decomposition [12]; biology and repartition of forest studies [14]; gas basic flow simulation [18]; biological model of tuna school formation [17]. A number of research papers involving hCA were presented also in a previous Conference on Cellular Automata for Research and Industry ($ACRI2006$) such as: the study of CA inalterability of topology without memory in [2]; bacteria modeling [21]; fluid (greases) simulation [13]; robot path planning [19]. While there is a wide range of literature on applications of hexagonal CA, we did not find any references to general algorithms for GPU-based 2D hexagonal CA, which indicates that our approach is novel and original.

2 Previous work

In order to increase the performance of CA computations we use a GPU programming based approach. The following section summarizes concepts relative to graphics processor units (GPU) and cellular automaton (CA) which are detailed in our previous work [9] where we proposed a GPU solution for first and second direct neighbors Boolean CA respectively called $1D_bCA$ and $2D_b^{vN}CA$.

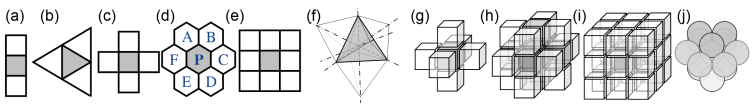

Fig. 2. Multidimensional CA geometric representations: (a) $1DCA$; (b) $2D^tCA$; (c) $2D^{vN}CA$; (d) $2D^hCA$; (e) $2D^MCA$; (f) $3D^tCA$; (g) $3D^{vN}CA$; (h) $3D^{M_1}CA$; (i) $3D^{M_2}CA$; (j) $3D^{h_1}CA$.

Graphical processor unit (GPU) Over the last few years, GPU cluster programming ([5]) has increased CPU computational capacity by a factor of 10 to 30 depending mainly on the graphics card series and the operating system. Our current approach uses a C++ code on CPU to communicate with OpenGL Shading Language (GLSL) programs on GPU. The graphics card uses three algorithms in its pipeline: the *vertex Shaders*, the

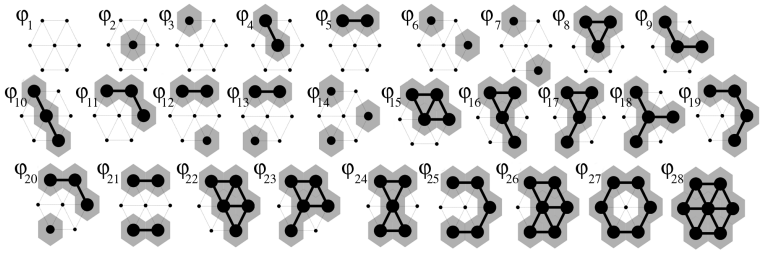

Fig. 3. All symmetrical configurations for Boolean $2D_b^hCA$.

Geometric Shaders, and the *Fragment* or *"Pixel" Shaders*. To use the graphics card as a computational device only the pixel Shaders is useful, however, we must compile and call the vertex Shaders. The main loop which allows cellular buffers to be considered as textures and copied back to matrices, is detailed in [9]. To include a summary here, the following main steps illustrate the aforementioned relationship between CPU and GPU:

1. associate a Vertex and Pixel Shaders pointers to source files;
2. compile Vertex and Pixel Shaders programs => return 2 pointers;
3. if compilation is fine...
4. ...get the Shaders-program pointer...
5. ...attach Vertex and Pixel pointers to the Shaders program;
6. ...link them all together.

Terminology, concept and definitions The terminology used in this paper, as in previous papers ([9]), include various notions of CA types, neighborhoods, rules, changes of state, and CA-key codes. Some cellular automata structures are presented in Figure 2 which illustrates most 1^{st}, 2^{nd}, and 3^{rd} dimensional CA. Considering the multitude of possible CA structure and types, we use the following rule to symbolize them all:

$$[\text{dimension}]D^{[\text{structure}]}_{[\text{state nb}]}CA \equiv dD^N_nCA \text{ with:} \tag{1}$$

- d as dimension, *e.g.* d=3 for space;
- n the number of states that a cell C can have, *e.g.* for Boolean $n \equiv$ 'b' $= 2$;
- N the structure, *i.e.*
  - for the square/cube structure with direct neighbors $N \equiv$ 'vN' $= 4$ in 2D, vN stands for **v**on **N**eumann;
  - for the square/cube structure with indirect neighbors $N \equiv$ 'M_i' $= 8$ in 2D, M_i stands for **M**oore with i being the growing number of possible indirect cases;
  - for the hexagonal grid $N \equiv$ 'h' $= 6$ in 2D;
  - for the triangle or tetragonal structure $N \equiv$ 't' $= 3$ in 2D and $= 4$ in 3D;

Therefore, in the dD^N_nCA domain, for each cell C with N number of neighbors, let's define c and ρ respectively for the number of configuration structure –see Figure 3 including all possible rotations *i.e.* the CA key-code φ_1, φ_2,... , φ_c)– and the number of possible CA, then:

$$\text{with } c = n^{(N+1)} \text{ and } \rho = n^c \text{ then } \rho = n^{n^{(N+1)}} \tag{2}$$

Let τ be the truth table made with the neighbor of C: $A, B, C...X$. Notice that the "pivot" cell is call "P" in Figure 2.

Now that basic concepts relative to Shaders and multidimensional CA have been defined, we move on in the following section to propose a model for computing any rule of $2D_b^hCA$ enabling the visualization of the corresponding data flow in real-time.

3 Current model: GPU-accelerated $2D_b^hCA$

Using equation 2 we can determine that $\rho_{(2D_b^hCA)}$ is equal to 2^{128} which is a large number of possible Boolean CA. A huge majority of these CA seem to behave in a very chaotic way such that it is not possible to identify geometrical characteristics of their respective behavior. That is why in this study we concentrate on symmetrical CA. We will see in the next section that the number of symmetrical CA is much smaller (2^{28}) than all possible Boolean CA, which narrows the search space. Nevertheless, trying to identify one by one the corresponding 268.435.456 behaviors is not the best approach.

Fragment Shaders model As previously stated the algorithm of the hexagonal CA is based on the relationship between a CPU program (using MSVC++) and Shaders codes. To make such a software, three main functions are required: CA computations, the hexagonal rendering, and the human machine interface (HMI, also called *user interface*) for the interactive choice of CA. As the first two functions are highly computationally expensive, both are implemented using Shaders. Furthermore, when CA's number of neighbors is superior to 6 key-code, it becomes too large to be transmitted. To solve this issue we encode the key-code as a 1D-texture that can be easily transmitted to the Shader program. Here is the pseudo-code of the fragment Shaders for the computation of the CA:

Based on the coordinates shown in Figure 2(d), we use the symbol " $\diamond$ " for the texel (texture pixel), Γ for the key-code texture, l the parity of the matrix line, and i for the intensity threshold to assume input images as Boolean matrices:

- $F_{xy} \leftarrow C_{xy} \leftarrow P_{xy} \leftarrow \diamond_{xy}$
- $(C,F)_x \leftarrow (C,F)_x \pm \delta_\diamond$
- if (l): $A_{xy} \leftarrow E_{xy} \leftarrow F_{xy}$ and $(A,E)_y \pm \delta_\diamond$ and $B_{xy} \leftarrow D_{xy} \leftarrow P_{xy}$ and $(B,D)_y \pm \delta_\diamond$
- else: $A_{xy} \leftarrow E_{xy} \leftarrow P_{xy}$ and $(A,E)_y \pm \delta_\diamond$ and $B_{xy} \leftarrow D_{xy} \leftarrow C_{xy}$ and $(B,D)_y \pm \delta_\diamond$
- $\forall$ cells X: set its state X_s to true if intensity of $\diamond_X > i$
- define state code $\gamma = P_s + 2A_s + 4B_s + 8C_s + 16D_s + 32E_s + 64F_s$
- $P_s^{t+1} \leftarrow \Gamma(\gamma)$

$(FEDCBAP)_d$	Corresponding $2D_b^hCA$ symmetrical shape –see Figure 3
0..31	1 2 3 4 3 4 5 8 3 4 6 9 5 8 11 15 3 4 7 10 6 9 12 16 5 8 13 17 11 15 19 22
32..63	3 4 6 9 7 10 13 17 6 9 14 18 12 16 20 23 5 8 12 16 13 17 21 24 11 15 20 23 19 22 25 26
64..95	3 4 5 8 6 9 11 15 7 10 12 16 13 17 19 22 6 9 13 17 14 18 20 23 12 16 21 24 20 23 25 26
96..127	5 8 11 15 12 16 19 22 13 17 20 23 21 24 25 26 11 15 19 22 20 23 25 26 19 22 25 26 25 26 27 28

Table 1. Corresponding symmetrical shape for the 128 bits $2D_b^hCA$ key-code.

Symmetrical rules As illustrated in [9], symmetrical CA can be very useful in the domain of real-time imaging, especially in computer graphics (CG) for automatic surface texturing or analysis of image flow. To understand the relationship between the key-code (rule) of a CA and its equivalent symmetrical code (if any), here is a practical example based on the emerging behavior in Figure $1^{[left]}$. This CA shows a growing of ramification spreading regularly but not symmetrically all over the surface. To do so we applied a specific, selected symmetrical key-code (*i.e.* 1 to 28 possible ϕ' set to $true$) equivalent to a CA-rule with a key-code of 32ϕ. In this particular case the relationship is described by equation 3.

$$\begin{aligned} \varphi'_{true} &= [2.4.6.7.9.10.12.13.15.16.17.18.20.23.24.26.28] \equiv \\ \tau(\varphi_{(1..32)} &= [A2E8EFC2EFBFCBE82BF3FEBB83FA2B8A]) \end{aligned} \tag{3}$$

This global behavior generating root-like patterns can be associated to non-photorealistic renderings (NPR). Notice that only cells that have changed state are shown, furthermore to improve printing effects surrounding cells are blurred.

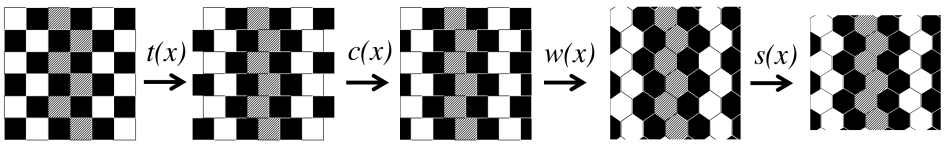

Fig. 4. Hexagonal rendering pipeline.

Hexagonal grid rendering

$$w(x) = \sum_{i=1}^{n} (\cos(\frac{k.x}{k^2})) \text{ with } k = 2i - 1 \tag{4}$$

Figure 4 presents the pipeline for the rendering Shaders enabling real-time and very accurate visualisation of the hexagonal grid. The four functions are $t(x)$ for translation, $c(x)$ for clipping, $w(x)$ for triangle-wave, and $s(x)$ for scaling. The first two allow the center of the cells to be consistent relative to the hexagonal grid. We used the *triangle wave* (*i.e.* Fourier series) shown in equation 4 enabling to simulate hexagons for $w(x)$ and the last function scales the (y)-axe by $\frac{\sqrt{3}}{2}$ so that hexagons are regular in all directions. An example of hexagonal grid rendering is illustrated in the pictures (e) and (f) of Figure 5.

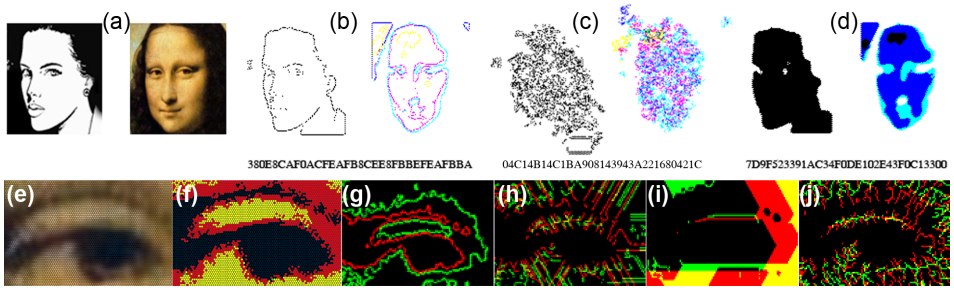

Fig. 5. From (a) to (d) a comparative test of five $2D_b^hCA$s on two input images with different levels of complexity (the right one being Da Vinci's "La Joconde"); from (e) to (j) Mona Lisa's left eye where triple CA (R,G,B) were applied, from left to right, a hexagonal discretization of the eye, the RGB Boolean equivalent, and four examples of hCA.

Graphical results and observations The last four pictures in Figure 5 (g) to (j) propose interesting hCA emerging behaviors respectively contour detection, artistic vectorization, fragment-Voronoi patterns, and –similarly to Figure 1– another CA of root-like pattern. Another example of hCA set is presented Figure 5 where three different hCA are simultaneously tested on two type of faces (with 25 steps): picture (b) representing a converging edge detection CA; picture (c) with symmetrical rule $[5.6..18.19.23.27]$ gives an excellent representation of the famous *game of life* CA –converging after 2324 steps. To study hCA behaviors more systematically, we focus on CA that grows from simple to complex seeds. Figure 6 presents a set of eight of such CA with ten different seeds –row (a)– from simple

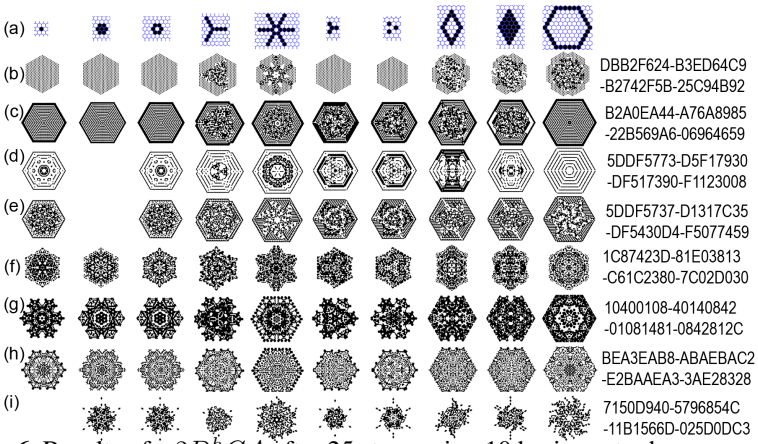

Fig. 6. Results of a $2D_b^hCA$ after 25 steps using 10 basic roots shows row (a).

dot (left) to a very small hexagon (right). In this palette of patterns, we can observe these non exhaustive list of properties: silhouette and texture symmetries, chaos, fractal, spiral formation, crystal-like patterns (*e.g.* snowflakes see also Figure **??** for comparison).

4 Performance

In this section, we present our contribution in acceleration of CA computations by comparing different types of algorithms (CPU or GPU based), machine configurations (with a number of different graphical cards), and texture sizes.

Six configurations All algorithms presented in this paper were developed in C++ on Microsoft Windows XP-pro using $MSVC$++, the graphics library OpenGL [16], and the OpenGL Shading Language ($GLSL$) [15]. To highlight the efficiency of the method presented in this paper, we demonstrate that our technique offers high real-time performance on common everyday computers which are available in reasonable prices (within the price range of regular home and office computers). Table 2 presents the six hardware configurations used to test our model.

Config	Comp.type	OS	Graphical Card	Memory	Processor	3DMark06 pts
1	PC (HP)	Vista	GeForce 8600 GS, 512MB	3072MB	2 proc. Q6600, 2.39 GHz	2650
2	PC	WinXP	GeForce 7600 GT, 256MB	1024MB	2 proc. Q6300, 1.86 GHz	3432
3	PC (Dell)	WinXP	Quadro FX 3500, 256MB	2048MB	2 proc. 6700, 2.66 GHz	4326
4	PC (Dell)	WinXP	Quadro FX 3500, 256MB	3072MB	4 proc. Xeon, 3.20 GHz	5198
5	NB ()	WinXP	GeForce 7900M GTX, 512MB	2048MB	2 proc. T7200, 2.00 GHz	4700
6	PC ()	WinXP	GeForce 8800 GTX, 768 MB	2048MB	2 proc. E6750, 2.90* GHz	12500

Table 2. General specifications of six configurations with their respective graphical test points based on 3DMark06 ranking system –(*): overclocked processor.

Texture limitations and influence As explained in previous sections, graphical textures are used as a computational buffer for the CA matrices as well as the CA key-codes. The size of these textures depend on the graphics card and pixel Shaders drivers. For GeForce 7000s and Quadro 3000s, buffer sizes are limited –for any dimension–to 4096 pixels, and can reach 8192 for high-end graphical cards (GC) such as the NVidia GeForce 8800 GTX. Therefore, when using GPU for fast buffer computation, one buffer cannot exceed

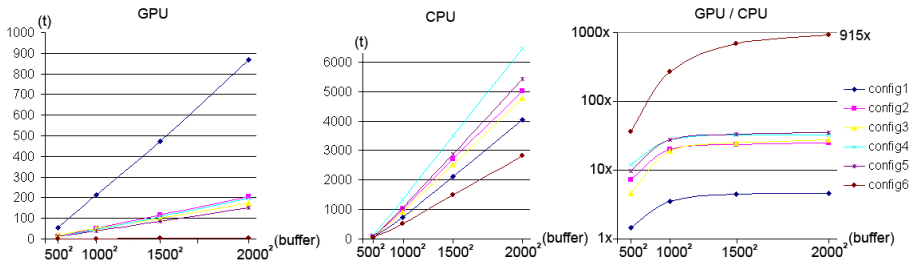

Fig. 7. Comparing performance between six different configurations: (a) and (b) time to compute ten thousand iterations on different buffer sizes, respectively for the GPU and the CPU; (c) comparison of the speed of GPU and CPU –using a logarithmic scaling.

a matrix bigger than 8192^2 –only available for high-end GC in 2D. In practice, the matrix cannot exceed the GC memory, *i.e.* in the best case 768Mb. Nevertheless there are ways to work this around to enable access to even large 3D matrices, some of these will be presented in our next publication on $3D_b^{vN}CA$–.

GPU and CPU results The left graph of Figure 7 presents the time (in seconds) that the six configurations spent for computing (using the GPU method) CA over different matrix sizes (from 500^2 to 2000^2) for ten thousand iterations. For the largest matrix, the slowest was configuration 1 with $869.2s$ and the fastest configuration 6 with $3.1s$. Notice that first results are linear and that the four intermediate machines are regrouped very close to each other at values from 152.3 to 204.2 seconds. The middle graph of Figure 7 proposes also the same task but computed by the CPU. Results are also linear but this time are ranged closer to each other from $2828s$ for configuration 6 (again the fastest) to $4042s$ the Xeon processor.

Comparison between GPUs Three categories of computers is clearly shown by right graph of Figure 7 representing the ratio between GPU and CPU computational power for this particular task. In one particular case (PC 1, GeForce 8000 series) , we suspect the operating system (MS Windows Vista) might be slowing down the graphics card, but to conclude this for certain, more tests are needed. We were also surprised by the excellent result of configuration 6, showing a computation ratio 915 faster than its own CPU which is the fastest of all other PCs. The rest of the configurations c_2, c_3, c_4, c_5 provided fairly good results which corresponds to the similar cases in literature, respectively: $24.6, 27.4, 32.7, 35.6$ times faster than their CPU. As demonstrated by the reported results, our proposed algorithm greatly increases the computational performance on all platforms.

5 Conclusions and Future Work

Following up on our recent publications [9], in this paper we have presented an extended method to simulate –i.e. compute and visualize– Boolean CA using a graphics card accelerated method for hexagonal grids. After introducing the programming technique using both CPU and GPU and summarizing concepts used in our previous work, we proposed a novel method to encode large CA key-codes allowing a generalized Boolean CA algorithm to be performed on a GPU –restricted to memory size limitations. We then have presented an original method to automatically sort symmetrical CA for any dimension based on their symmetrical structure. We have detailed how to encode generalized algorithms for $2D_hCA$ presenting the pseudo-code of the Shaders algorithms. Finally, to

show the capabilities of our model we have presented examples of characteristic patterns, common global behaviors, and computational statistics done on six different hardware and software configurations. We demonstrated that our model allows to compute up to 915 times faster than if CPU alone was utilized. We are convinced that this method can lead to an accelerated and generalized model for 3D-surface CA [8] which would be a breakthrough for automatic texturing and simulations. The future work in our agenda regarding this research is to report the findings on an extended model for three-dimensional Boolean CA with direct neighbors ($3D_{vN}CA$). This work also explores new issues such as memory management and real-time graphical rendering of 3D with GPU. Furthermore, our current research team works on a range of topics on virtual reality (VR) and we are developing a system to explore CA models interacting with virtual scenes.

Acknowledgment We would like to thank Dr. Arzu Coltekin and Ms. Thelma Coyle for their technical suggestions, kindness, and proofreading of the text.

References

1. Andrew Adamatzky, Andrew Wuensche, and Benjamin De Lacy Costello. Glider-based computing in reaction-diffusion hexagonal cellular automata. *ScienceDirect, Chaos Solutions and Fractals*, 27:287–295, 2006.
2. Ramón Alonso-Sanz and Margarita Martín. A structurally dynamic cellular automaton with memory in the hexagonal tessellation. In *7th International Conference on Cellular Automata for Research and Industry (ACRI06)*, pages 30–40, Perpignan, France, 2006. Springer.
3. Bastien Chopard and Daniel Lagrava. A cellular automata model for species competition and evolution. In *7th International Conference on Cellular Automata for Research and Industry (ACRI06)*, pages 277–286, Perpignan, France, 2006. Springer.
4. L. Hernández Encinas, S. Hoya White, A. Martín del Rey, and G. Rodríguez Sánchez. Modelling forest of fire spread using hexagonal cellular automata. *ScienceDirect, Applied Mathematical Modelling*, 31:1213–1227, 2007.
5. Zhe Fan, Feng Qiu, Arie Kaufman, and Suzanne Yoakum-Stover. GPU cluster for high performance computing. In *SC'04*. IEEE Computer Society, 2004.
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, $2^{n.d}$ edition, 1993. 1175 pages.
7. Stéphane Gobron, Francois Devillard, and Bernard Heit. Retina simulation using cellular automaton and GPU programming. *Machine Vision and Applications Journal*, 66:331–342, 2007.
8. Stéphane Gobron and Denis Finck. Generating surface textures based on cellular networks. In *In The Geometric Modeling and Imaging international conference (GMAI06)*, page 113120, Londres, UK, July 5-7 2006. IEEE Computer Society.
9. Stéphane Gobron and Daniel Mestre. Information visualization of multi-dimensional cellular automata using GPU programming. In *11th International Conference on Information Visualisation (iV07)*, pages 33–39, Zurich, Switzerland, July 2-6 2007. IEEE Computer Society.
10. Simon Harding and Wolfgang Banzhaf. Fast genetic programming and artificial developmental systems on GPUs. In *21st International Symposium on High Performance Computing Systems and Applications (HPCS'07)*, Saskatoon, SK, Canada, 2007. IEEE Computer Society.
11. G. Iovine, D. D'Ambrosio, and S. Di Gregorio. Applying genetic algorithms for calibrating a hexagonal cellular automata model for the simulation of debris flows characterised by strong inertia effects. *Geomorphology*, 66:287–303, 2005.
12. Y. L. Lan, D. Z. Li, and Y. Y. Li. Modeling austenite decomposition into ferrite at different cooling rate in low-carbon stell with cellular automaton model. *ScienceDirect, Acta Materalia*, 52:1721–1729, 2004.
13. Shunsuke Miyamoto, Hideyuki Sakai, Toshihiko Shiraishi, and Shin Morishita. A flow modeling of lubricating greases under shear deformation by cellular automata. In *7th International Conference on Cellular Automata for Research and Industry (ACRI06)*, pages 383–391, Perpignan, France, 2006. Springer.
14. C. Pagnutti, M. Anand, and M. Azzouz. Lattice geometry, gap formation and scale invariance in forests. *ScienceDirect, Theoretical Biology*, 236:79–87, 2005.
15. R.J. Rost. *OpenGL Shading Language*. Addison Wesley Professional, 2nd edition, 2006.
16. D. Shreiner, M. Woo, J. Neider, and T. Davis. *OpenGL Programming Guide: the official Guide to learning OpenGL v2.0*. AddisonWesley Professional, 1st edition, 2005.
17. Sabine Stöcker. Models for tuna school formation. *Mathematical Biosciences*, 156:167–190, 1999.
18. Jean-Christophe Toussaint, Jean-Marc Debierre, and Loïc Turban. Deposition of particles in a two-dimensional lattice gas flow. *Physical Review Letters*, 68(13), 1992.
19. Gabriel A. Wainer. Modeling robot path planning with cd++. In *7th International Conference on Cellular Automata for Research and Industry (ACRI06)*, pages 595–604, Perpignan, France, 2006. Springer.
20. Stephen Wolfram. *A new kind of science*. Wolfram Media Inc., 1st edition, 2002.
21. Yilin Wu, Nan Chen, Matthew Rissler, Yi Jiang, Dale Kaiser, and Mark Alber. Ca models of myxobacteria swarming. In *7th International Conference on Cellular Automata for Research and Industry (ACRI06)*, pages 192–203, Perpignan, France, 2006. Springer.
22. Zhang Yongzhong, Z-.D. Feng, Han Tao, Wu Liyu, Li Kegong, and Duan Xin. Simulating wildfire spreading processes in a spatially heterogeneous landscapes using an improved cellular automaton model. In *Geoscience and Remote Sensing Symposium (IGARSS'04). Proceedings. 2004 IEEE International*, pages 3371– 3374. IEEE International, 2004.