Machine Vision and Applications manuscript No.
(will be inserted by the editor)

Stéphane Gobron · François Devillard · Bernard Heit

Retina Simulation using Cellular Automata and GPU Programming

Received: date / Accepted: date

Abstract This article shows how the architectural modelization of biological retina allows real-time performances on cheap and widespread computing systems. Firstly we describe the biological retina in terms of its pipeline architecture detailing its layer behaviors and properties. Then we propose a corresponding pipelined model of artificial retina based on cellular automata. In this work, the main innovation is the computing method based on the programming of a personal computer graphical card using *OpenGL shading language*. The last section demonstrates efficiency of our model with numerical and graphical results. We particularly highlight that our direct implementation on the *Graphical Processor Unit* (*GPU*) provides a computation power about twenty times as fast as ordinary programming.

Keywords Computer vision · Real-time computing · Artificial retina · Graphical Processor Unit programming · OpenGL shading language · Cellular automata

S. Gobron
Movement and Perception laboratory, University of Marseille 2.
163, avenue de Luminy, 13288 Marseille cedex 9, France
Tel.: +33(0)491172255
E-mail: stephane.gobron@univmed.fr
http://www.laps.univ-mrs.fr/∼gobron

F. Devillard
UMR CNRS 7039 (CRAN - CNRS), Henri Poincaré University, Saint-Dié-des-Vosges Institute of Technology.
11, rue de l'Université, F-88100 Saint-Dié-des-Vosges, France
Tel.: +33(0)329536001
Fax: +33(0)329536011
E-mail: francois.devillard@iutsd.uhp-nancy.fr

B. Heit
UMR CNRS 7039 (CRAN - CNRS), Henri Poincaré University, ESSTIN.
2, rue Jean Lamour, F-54500 Vandœuvre-lès-Nancy, France
Tel.: +33(0)383685131
Fax: +33(0)383685001
E-mail: bernard.heit@esstin.uhp-nancy.fr

1 Introduction

Nowadays, computer vision needs more and more visual information to describe and analyze natural scenes efficiently. Visual clues such as shapes, movements, and colors bring complementary information about the observed scene and are useful for a reliable interpretation. The simultaneous perception of several properties provides a better understanding of the scene.
These methods applied to image processing often fail because of their excessive specialization and their lack of flexibility, whereas biological visual systems turn out to demonstrate outstanding achievements. The alternative and bioinspired methods have shown robust results but need costly computing systems to reach real-time processing. So the relevance of using graphical processor unit programming to avoid both constraints mentioned above and to provide flexibility and real-time computation and rendering is highlighted.

1.1 Background

Recently, we observed real-time systems dedicated to the retinian implementations essentially based on *Application Specific Integrated Circuit* (*ASIC*) components. Here are a few examples of the related literature:

- Analog components ([14], [19], and [21]);
- Or more rarely digital *Very Large Scale of Integration* components (*VLSI*) [3].

These works implement photodetectors and basic processings and so are classified as *artificial retinas*. Most of them are extremely specific while our work puts forward an original, real-time and flexible model using cheap and widespread programmable systems.

1.2 Overview

The paper is organized as follows. In section 2 we begin by describing the biological retina, emphasizing on its

pipelined structure and corresponding layer properties. The next two sections explain the heart of our model. First, section 3 describes the data flow technical architecture. Secondly, section 4 details the reasons for using GPU programming and how to compute each retina layer resorting to cellular automaton on GPU. Results in terms of computational rates and corresponding graphical plates are presented in section 5. And finally, section 6 concludes this paper and puts forward some of the related future work including parallel pipeline and three dimensional upgrades.

2 Methodology

Our algorithm describes the architectural modelization of the retinian pipeline processing. The resulting implementation needs well-suited computing systems. We show in this article that a standard personal computer solution that directly uses parallel architecture of its GPU produces spectacular performances especially in this kind of applications. Results (section 5) show that direct and optimized programming using mixed algorithm –*Object Oriented Language* and *OpenGL Shader Language* (OOL [24], and $GLSL$ [23])– reduces first the software development compared to very low-level programming and second the computing times comparatively to an ordinary method. Therefore, the proposed implementation obtains real time performances on cheap and widespread computing system.

To understand our methodology enabling to reach implementation level, we need to understand well enough how the biological retina works (keeping in mind the computer science (CS) context). The following subsections describe the global architecture of the biological retina, then the specific key role of *Outer and Inner Plexiform Layers* (respectively OPL and IPL [17]) and the link between data flows going through layers and their respective functionalities.

2.1 Biological retina

Mechanisms of the visual perception ([16], and [17]) –see figure 1(a)– start by a scene acquisition (symbolized by black arrows at the top of the figure) through photoreceptors and end by several data flows specialized in characteristics of that scene. Retina is composed of several layers that are organized together to filter visual information. Retinian filters behave similarly to low-pass frequencies spatiotemporal filters modelized by signal processing ([2], and [20]).

The figure 1 demonstrates the pipelined architecture of the retina: picture 1(a) being a vertical section through a human retina, we propose to grouped main functionalities of the retina around OPL and IPL layers presented in schema 1(b) and (c).

The photopic visual perception starts with the scene captured by photoreceptors that are cone and rod cells. Rod cells have photoreceptor properties specialized for low light intensity which is not required for current approach, that is why in this paper, only the cone cells are taken into account. Image luminance is sampled and transmitted by cone layer for the downstream retinian and cortical layers. From cone to ganglionar layers [5], a pipeline of cellular layers can be identified. Two functional layers, the OPL and IPL, can classify the pipeline into two functionality categories.

Through the biological retina, each layer is composed of a network of interconnected cells linked by more and less distant connections. Therefore, each cell can be modelized by a processor or a basic filter that has:

- Several inputs are established by connections with a variable number of neighboring cells;
- And an output that performs the cell activity comparable to a low-pass frequencies filter.

2.2 OPL and IPL areas of neuropil

The OPL layer where connections between rod and cones and vertically running bipolar cells and horizontally oriented horizontal cells occur [1]. OPL properties are generated by the synaptic triad which is composed of three kind of interconnected cells:

- The cone cells constitute a layer of the transduction and regularisation processing. The transduction converts luminance into electrochemical potentials aimed at the downstream layers. The regularisation consists in filtering input signals with a light low-pass frequencies filtering. The cone cells defined by their shapes (midget, diffuse...), their types of response (on, off) and their functions (color sensibilities: LMS respectively red, green , and blue colors). Moreover their behaviors depend locally on the luminance intensity and contrast [29];
- The horizontal cells constitute a layer of strong regularisation processing. The output response performs from the cone cells output to elaborate a spatial average of the image intensity [29];
- The bipolar cells make the difference between the horizontal (luminance average) and the cone outputs. So bipolar cells estimate the local contrasts of the image intensity to increase visual indices. As cone cells, the bipolar cells are various. We observe bipolar cells classified also by their shapes (midget, diffuse...), their types of response (bipolar on, off...) , and their functions (contrast estimation: red/green, blue/green+red...) [27].

The bipolar axons transmit the OPL outputs to the IPL area. The IPL processing is assumed by amacrine and ganglion cells:

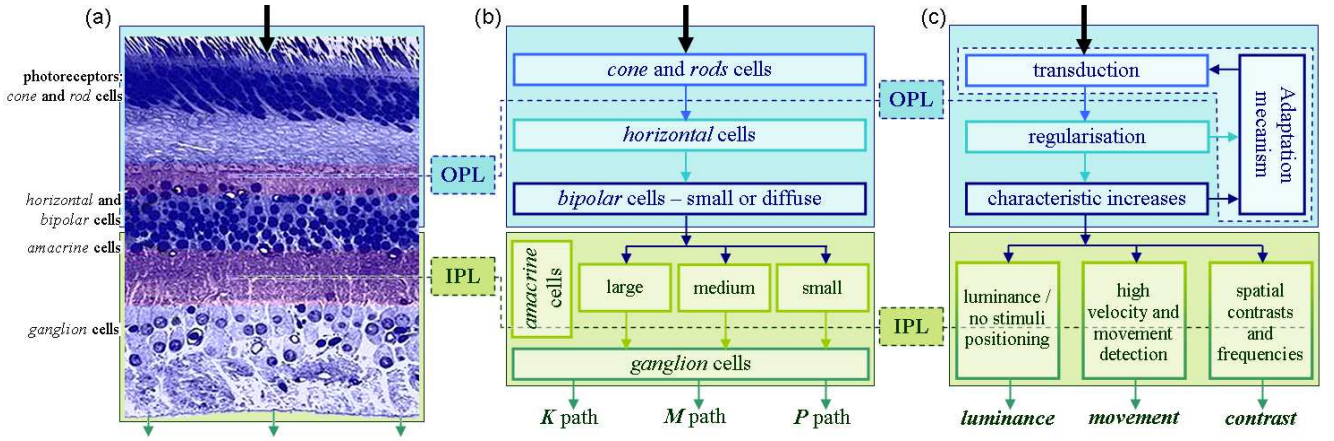


Fig. 1 Three ways of showing the human retina architecture, in terms of: (a) biological section through a real retina; (b) region names and general pipeline of processes; (c) pipeline functionalities.

- The amacrine cells have multiple functions. Its main function transforms a temporal variation into a peak signal typically similar to a high-pass frequencies filtering [18];
- The ganglion cells make the difference between the bipolar cells (contrast estimation) and the amacrine cells ouputs. The ganglion cells are classified as bipolar cells by their shapes (midget, diffuse...), their types of response (on, off, on+off) , and their functions (spatio temporal contrast estimation and luminance estimation) [27].

The retina pipeline because the various types of cells is composed by three pathways that sort out visual information (luminance, contrast, colors and movement). The known pathways in the inner retinian areas are called K, M , and P respectively the koniocellular, magnocellular , and parvocellular pathways (see figure 1).

2.3 The retinian data flows

The visual information flow is little by little divided in three known pathways with weak redundancies. The retinian output provides a split information and shared in three specialized pathways:

- Koniocellular path (K): presents numerous features which make it sensitive to luminance and chrominance [8];
- Magnocellular path (M): carries out a first processing which is essential in movement perception. Antagonistic receptive fields are used [4];
- Parvocellular path (P): achieves a pre-processing enabling to extract the visual clues characterizing the objects. The cells making up the type P distance are the majority by far. They are small-sized, tonic –of less sensitivity to temporal modulations), show and situated especially in central retina. Sensitive to high spatial frequencies (thin details or small objects) and sensitive to low time frequencies, they are the seat of visual acuteness and thus code the specific position of the stimulus and the color [4].

We have made in these last paragraphs a basic description of the whole retina. The biological retina is difficult to modelize with accuracy. The inner retinian mechanisms (feedback, adaptation...) are complex and sometimes not located thus all are not well studied (for instance K pathway). The proposed modelization uses a layer organization that copies the presented architecture figure 1. The whole processing approaches P well-known pathways of the retina. The P Pathway is considered in an achromatic version. We have designed a generalized layer (shared by each kind of cells) that is set up to give it the liked properties. This layer is repeated to build retinian architecture.

3 Approach

3.1 Layer modelling

Each layer is built by a network of cells. Each cell is an individually spatiotemporal low-pass filter. The figure 2 shows how our algorithm computes for a cell for an $1D$ spatial dimension network. The cell has an output s (cell activity) and takes for inputs s and these of their neighboring cells. The perform of s is obtained by an iterative computation. Four performing steps are necessary and illustrated by the figure 2. Variables x and k are respectively the location and instant using an unit sampling. The whole of our layers uses our cellular algorithm with an adapted settings function of their functionalities.

The resulting filter is composed by twos categories of pipelined filters. A *Finite Impulse Response* filter (FIR) and an *Infinite Impulse Response* filter (IIR) that smooth respectively spatially and temporally cells input.

The parameters that influence spatially the cell output are n and α. They simulate the lateral interaction

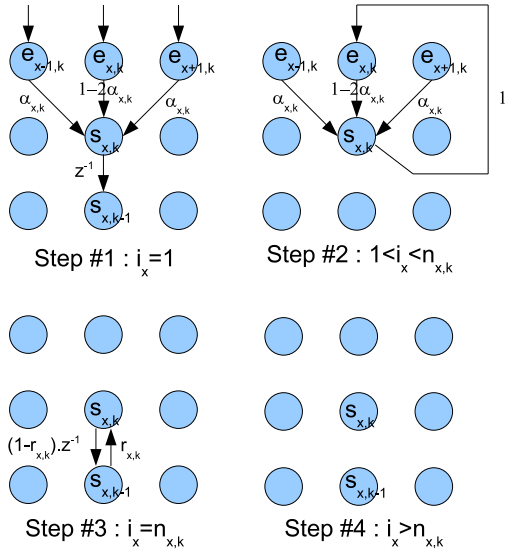


Fig. 2 The equation graphs that present the four steps of our algorithm implemented for an $1D$ layer. the variable x represents the cell position and k the time value with unit scale samplings. z^{-1} is the unit delay operator.

between neighboring cells. The FIR filter coefficient α (see figure 2) performs influence between two immediate neighboring cells. n is the number of FIR filter iterations that is applied for each cell. Thus the values α and n adjust the dendritic spreading out for each cell ($2n+1$ by $2n+1$). The coefficient α is adjusted to obtain a pseudo-gaussian low-pass frequencies response (best approximation for $\alpha = 0.25$) [20]. The parameter that influences temporally the cell output is r. It allows to create a persistence effect on the cell output (produced by the cell membrane capacity).

The usual cell layer has input connexions with one downstream layer. It has two exceptions for the transduction layer (luminance is the input of the cone layer) and the differentiation layers that substract outputs of two downstream layers (bipolar and ganglion layers). For each instant k cone layers sample another luminance image I –hence forming a video sequence– and for each cone cell is associated to one cone (1 pixel = 1 cone). Furthermore, to simplify complexity, cone cells are considered as achromatic photoreceptors.

3.2 Cellular automaton modelling

The implementation of our dynamic model is based on simple rules of the operation of the cell and evolution of its neighborhood. Thus we have naturally chosen to model each cell layer by a cell network associated with a cellular automaton [28]. The cellular automaton constitutes a network of cells having common geometrical properties to which a change of state (finished list of rules) is applied according to a common synchronization. The cellular automatons enable us to approach simply from an architectural and behavioral point of view, the local properties of the retinian layers. The simplicity of the cells and connections makes it possible to consider an implementation adapted for processors GPU strong parallelism but with a reduced instruction set. Each cell is driven by rules of operation induced by its state. The change of state of the cell is conditioned by the iteration number n.

Each cell is a state machine. It is composed by states associated with behavioral rules. The 4 steps constituting an iterative cycle of processing that are made by the cellular automaton. The individual cell behavior of the automaton is driven by the 4 following steps:

- Step #1: Input sampling;
- Step #2: FIR filter performing (locally various numbers of iterations);
- Step #3: IIR filter performing (one iteration);
- Step #4: Automaton synchronization.

Step 1 (one iteration) consists in sampling for all the cells of the layer upstream ouput(or of the signal of brightness for the cones layer). The cell remains in step 2 (duration of n iterations) as long as spatial filtering localized is not succeeded. The cell passes then at step 3 (duration of one iteration). The cell remains at step 4 (variable duration) as long as all the cells individually did not reach it.
For a row of cells that processing is presented by the figure 2. The computing steps for the cell at location x are given by a counter value i_x (initialized by -1 at starting). The following table 1 gives us their actions and activation conditions for all processing steps:

The $2D$ release of the network described in figure 2 complicates a little the whole processing. The network grid has a square shape and we have chosen to use only four connexions for each cell. Thus each cell is connected with four cells of its immediate neighborhood; The four connections used are respectively towards horizontal and vertical directions centered on the processed cell.

4 GPU retina simulation

4.1 Retina pipeline software design

Similarly to [10] –which proposes a CPU-based OPL simulation– our representation of the retina pipeline is modelled using a succession of two-dimensional matrices often associated with a cellular automaton (CA) for computation.

For readers not used with CA, Gérard Vichniac [26] gives the following definition that fit issues of current retina model: *Cellular automata are dynamical system where space, time , and variables are discrete. Cellular automata are capable of non-numerical simulation of physics, chemistry, biology , and others and they are*

Step	Action	Condition
#1	$e'_{x,k} \leftarrow e_{x,k}$; $i_x \leftarrow i_x + 1$;	$i_x < 0$
#2	$s'_{x,k} \leftarrow (1 - 2 \cdot \alpha_{x,k}) \cdot e'_{x,k} + \alpha_{x,k} \cdot e'_{x+1,k} + \alpha_{x,k} \cdot e'_{x-1,k}$; $i_x \leftarrow i_x + 1$;	$0 \leq i_x < n_{x,k}$
#3	$s'_{x,k} \leftarrow (1 - \cdot r_{x,k}) \cdot s'_{x,k} + r_{x,k} \cdot s'_{x,k-1}$; $i_x \leftarrow i_x + 1$;	$i_x == n_{x,k}$
#4	$s_{x,k} \leftarrow s'_{x,k}$; $i_x \leftarrow -1$;	$i_x > n_{x,k}$

Table 1 The table of rules of the CA that drives an $1D$ retinian layer. e' and s' are the sampled internal values of respectively e and s in each CA cell.

useful for faithful parallel processing of lattice models. In general, they constitute exactly computable models for complex phenomena and large-scale correlations that result from very simple short range interactions. The technical design of our model also covers other specific domain using CA. Here is a list of references presenting CA mixed with computer graphics and/or image processing:

- [6],and [22] respectfully for notions of grammar and computer engineering;
- [7],[12],and [13] for CA and graphics based simulations;
- [9] for *neural networks* (sim. to CA) and image processing –especially segmentation;
- [11], [25],and [15] for CA and GPU programming.

The figure 3 –where OPL section is presented in blues and IPL section in greens as usual in this paper– presents the pipeline five major steps:

From an input I –that can be an AVI video of a *webcam*– the information is first transmitted to the cone layer. Resulting computation C is then transmitted both in the horizontal and bipolar layers where results of the horizontal cells H are simultaneously subtracted. In a similar way, bipolar cell's output B is transmitted to both amacrine and ganglion layers. Finally, ganglion outputs G are the result of the subtraction of amacrine outputs A and B.

As shown in figure 3 each layer has a specific behavior defined by one or two functions. These functions can be simulated with cellular automata with thresholds enabling real parameters [1]. Cone cells have a light blur effect on the image that can be simulated with a cellular rule equivalent to step 2 of table 1. Similarly to cone cells, horizontal cells produce a blur effect but an intensive one. This effect is associated with a persistence property that is presented with the cellular rule table 1, step 3. Bipolar cells provide output values $(H - C)$. Amacrine cells use a persistence function which is already defined by the horizontal cells representation. And finally, ganglion cells behave similarly to the bipolar layer using $(A - B)$ instead.

[1] Classical cellular automata use non-real parameters and therefore thresholds are not necessary

4.2 Real-time computation using GPU

Due to the increasing needs of the video game industry - especially in terms of fast computation of large textures- nowadays graphical cards have strong computational potential. This potential can be accessed using graphical processor units that can be directly accessed using assembly language or hardware specific languages. Fortunately, they can also be programmed with C-like languages called *shader programming language* such as $GLSL$ (OpenGL shader language proposed by OpenGL ARG) or CG (developed by nVIDIA) -see [25] and [12]. Notice that a graphical card is composed of a set of GPU that have the property to parallelize these shaders programs and therefore is much more powerful than a single CPU of the same generation.

As said in the previous section, we propose to use this powerful tool not for rendering purposes but instead as a computational unit specialized in the retina pipeline data transform. To do so we first associate every cellular layer matrices with two dimensional graphical textures. Second, based on $GLSL$ we developed a general cellular automaton program that can be parametrized for each specific retina functionality –see table 1.

The communication between the CPU program (that control the retina pipeline) and the GPU program (that control local cellular behavior) is presented figure 5.

The first main task consists of making our $C++$ software compiles the CA *shader* program –see appendix A. The second task is to transform continuous input data flow into a $2D$ graphical texture buffer and send this texture as input of the GPU pipeline. As shown in figure 5 we have tested in this paper two types of input animations: an AVI file or a webcam buffer. From that point, the CPU program leads the GPU via specific parameterization of the CA-*shader* code: for each specific layer behavior, CA rules are reset. All GPU results are projected onto virtual windows, then screen-copied and send back into different pipeline texture buffers. Finally, output texture is then shown onto the screen.

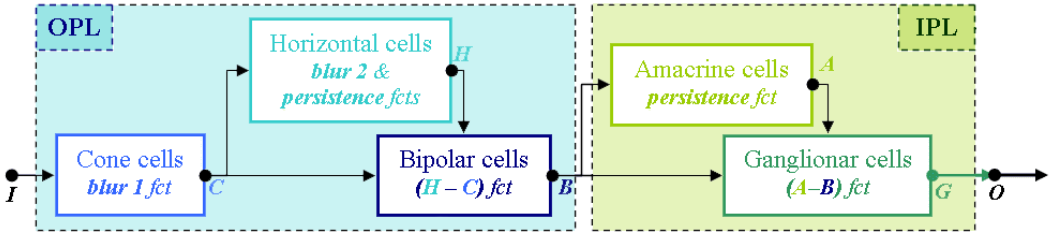


Fig. 3 Simplified retina pipeline viewed in terms of cellular layers.

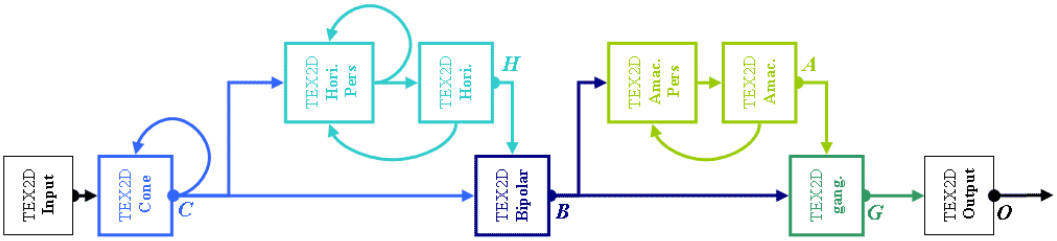


Fig. 4 Simplified retina pipeline viewed in terms of textures.

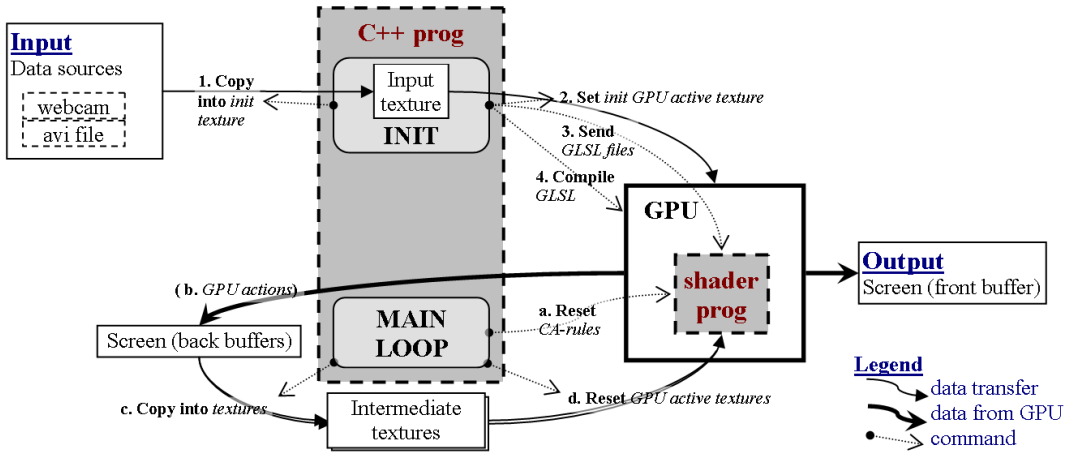


Fig. 5 Parameterization of different CA on the GPU.

5 Results

In this section, after presenting the hardware environment used for testing, we first propose why and how to estimate capacities of our model (subsection 5.2) and second we illustrate this retina simulation with resulting images emphasizing characteristics that would have been almost impossible to find is not in real-time –see subsection 5.3.

5.1 Environment

Data from each cell in the CA is directly transmitted via OpenGL^{TM} 1.4 shading language using ARB extensions [24] for the best compatibility. Notice that ARB extensions are not trivial as they lack of documentation and standardization, that is why we advice using the brand new (2006) OpenGL^{TM} 2.0 [23] as a reference guide. Results presented in this paper were generated using an AMD Athlon^{TM} 64X2 Dual Core 4400+ CPU –2.21 GHz– with 2 Go of RAM and a NVidia^{TM} GeForce 7800 graphical card. For the implementation we used $C++$ and $GLSL$ languages on MS-Visual Studio^{TM} 2005 and MS-Windows^{TM} 2000 as *Operating System* (OS). Concerning the webcam, we used a $CREATIVE^{TM}$ model No:PD1030. Webcam testings have shown that for a resolution of 640 × 480 (the only one we used in this paper), the webcam produced at most 15 fps.

Notice that most computations were dedicated to the graphical card and that in term of pure memory cost the developed program outmost reaches 35 Mo of RAM. Therefore it is does not required having a dual core processor unit with 2 Go to obtain results similar to the ones presented in the following subsections. The minimum

Blur factors		*GPU* approach										*CPU* model
		direct input file					*webcam inputs*					*webcam inputs*
n_c	n_h	C	H	B	A	G	C	H	B	A	G	B-outout
1	1	889	531	440	390	351	40	30	28	27	27	3
2	4	626	264	240	225	211	33	24	22	22	21	3
3	7	483	176	165	158	151	29	20	19	18	18	2
4	10	393	132	126	121	117	27	19	18	17	17	< 2
5	13	331	105	101	98	96	25	18	17	17	17	< 2
6	16	286	88	85	83	81	23	17	16	16	16	< 2
7	19	252	75	73	72	70	22	17	16	16	16	< 2
10	28	186	53	52	52	51	21	16	16	16	16	$<< 1$
20	58	98	27	26	26	26	18	15	15	15	15	$<< 1$

Table 2 Output image rates in frames per second (fps). n_c and n_h are respectively blur loop indexes of the cone and horizontal layers. α parameter is fixed at 0.125 for each cell to approach a $2D$ pseudo-gaussian low-pass filtering. The r parameter does not matter for these tests.

hardware materials are an adequate motherboard bus environment for high speed transfer between CPU and GPU and of course, a graphical card such as NVidia^{TM} GeForce 6800 or more.

In the following paragraphs, we deduce individual computation time for layers. We perform the whole retinian algorithm with several conditions of parameter settings to deduce the time duration for a standard layer because we was not able to evaluate accurately short times elapsed ($< 1ms$) on MS-Windows OS.

5.2 *GPU* solution testings

Many parameters interacts with our model, therefore testing capacities of such algorithm is not a trivial task. Nevertheless here are the main objectives of our testing methodology:

- To define possible input resolution and estimate corresponding image rates;
- To compare similar algorithm using a CPU and a GPU approach;
- And to find execution cost of each retina layers.

As explained above, we mainly used our webcam resolution (640×480) for experiments. Our technique implies the use of graphical cards which in current case restrict resolution to a maximum of 4096×4096 – notice that this resolution remains much higher than most professional HS video cameras. Concerning influence between resolution and computational expenses, due to graphical card GPU architecture, increasing input resolution would decrease fps rates in a linear manner. For instance using a resolution of 1280×960 as input would be four times slower than results presented in this paper.

Figures 6 (a) and (b) graphically illustrate the two right sections of table 2. From that figure we can observe that on the left diagram horizontal cell computation is far to be the most expensive time and that the right diagram shows that

For a set of n couple – see parameters of the layer modelling described by the section 3.1– the table 2 proposes resulting *Frames Per Second* (fps) at C, H, B, A , and G outputs (see figure 4). As horizontal cell are characterized by a blur factor much higher than cone cells, we have selected n couple such that n_h increase three times faster than n_c. The use of such a regular factor is justified by Bonnet *et al.* [4], p.13: sizes of receptor fields (simulated in this paper by n couple) increase linearly with the retina eccentricity.

First, in the left side, the table presents values assuming direct files –such as an AVI sequence or even a single image– is used as input. Even for relatively high (n_c, n_h) values, results are always over the real-time threshold rate –*i.e* around 25 fps. These high frame frequencies demonstrate that this model of artificial retina enables to realize more advanced and powerful models (see perspectives in the section 6).

Resulting time expenses using as input a webcam are shown on the right side (without the last column). The tremendous drop in performance between previous results and the one in this section of the table is due to the specific webcam used: it has low efficiency capture rates (less than 15 fps in the worst conditions) and frequent interruption requests which virtually freeze the hardware performances. Nevertheless, even with such poor quality material frame rates never dropped below 15 fps. Notice that due to webcam interruptions the resulting animation appears often jerky.

To compare our model with CPU-based works [10], the last column of table 2 presents resulting fps for a similar CPU retina simulation. To make it possible, only OPL layers computational results (at bipolar –*i.e.* B– outputs) are compared; further layers using CPU have much too slow rates. Based on experimental values – comparing GPU column B and the CPU's column– we can conclude that the approach presented in this paper is at least 18 times faster than the CPU solution. This coefficient of 18 can also be found using input GPU fps rate compared to CPU rates, respectively 1537 divided by 85 fps and 82 divided by 4 –see table 3.

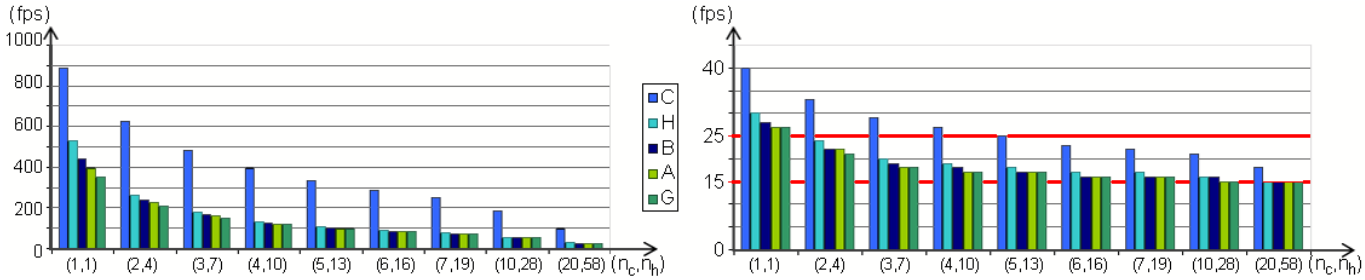


Fig. 6 Graphical interpretations of table 2: (a) outputs using a direct input file and *GPU*; (b) outputs using a webcam input and *GPU*.

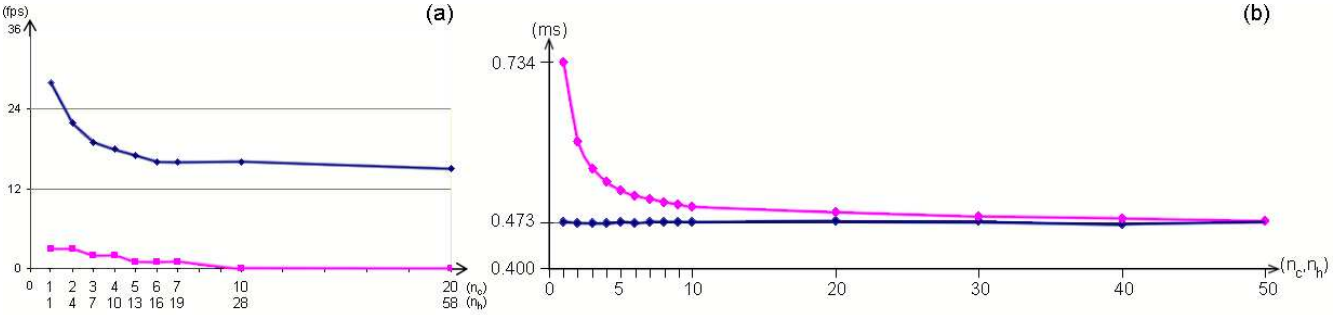


Fig. 7 Comparing results: (a) *CPU* and *GPU* approach using a webcam input, graphical interpretations of table 2 –red for *CPU* and blue for *GPU*; (b) cone vs. horizontal cells shader computational expenses.

GPU **approach**	*Direct input file*	*Webcam inputs*
	1537	82
CPU **model**	*Direct input file*	*Webcam inputs*
	85	4

Table 3 Input image rates in *fps*. For webcam tests, high values of *fps* are obtained by over sampling of the video input buffer.

As hardware used for experiment in this paper is different to the one used by [10], to make an exact comparison between this *GPU* and the *CPU* approaches, we used same conditions and compare identical outputs. Table 3 presents *GPU* and *CPU* based model input video rates. Moreover, figure 7(a) compares the *GPU* and *CPU* bipolar outputs –see table 2. Figure 7(b) presents the ratio of computational cost –in *ms*– divided by n. Resulting values are proposed in the left table and corresponding graph is shown on the right respectively in pink/violet for cone and in dark blue for horizontal n values. Two experimental interesting values can be deduced from that diagram:

– First, as both function aim at an identical constant value for n reaching high values, the time cost for a single blur functionality computation using *GPU* is around 0.47 *ms*;
– Second, cone appears to be more costly for low n value and then reaches the tangent 0.47 *ms* value. The difference between cone and horizontal functionalities is only on the *GPU* resetting. Hence, as cone cells make the first *GPU* call, resetting *GPU* cost should be about (0.734-0.473 *ms*) which is about 0.26 *ms*.

5.3 Graphics results

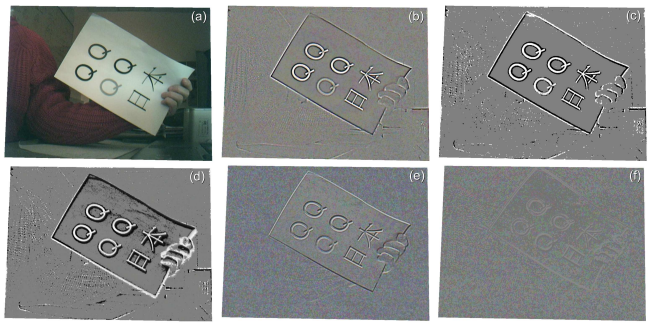


Fig. 8 Main step of the real-time retina pipeline: (a) input webcam animation; (b) bipolar output; (c) saturated (*i.e.* [0.0, 0.5, 1.0]) bipolar output; (d) amacrine output; (e) and (f) ganglion outputs respectively with and without movements.

Figure 8 proposes the main resulting steps of the retina pipeline using webcam inputs. As main viewed object we used a sheet of paper with a large and strongly contrasted font: the letter Q is an interesting sample as it

shows a natural loop crossed by a segment; for more complex character visual simulation we use the term japanese kanjis *ni-hon* (actually meaning *Japan*).

This plate shows in particular that front an average/low contrasted input sequence –figure 8(a)– our model produce expected behaviors for each main retina layer –in real-time. Figure 8(b) illustrates the bipolar information enhancing: localisation, magnitude and length of the light. These crucial properties can be use for edge detection. We tried to focus on that contour determination, using thresholds Δ_b on cell's brightness c_b in figure 8(c) respectively: if $c_b < (0.5 - \Delta_b) \Rightarrow c_b \leftarrow 0.0$ else if $c_b > (0.5 + \Delta_b) \Rightarrow c_b \leftarrow 1.0$ else $c_b \leftarrow 0.5$.

The figure 8(d) presents the amacrine output cellular layer which is similar to delayed bipolar outputs.

The last two pictures presenting the ganglion outputs give precious informations. The contrast between figure 8(e) and (f) is obvious: (e) presents immediate contour detection of object even with very a slight movement, giving a depth visual aspect. On the contrary, (f) demonstrates that if object stop moving –and as the webcam cannot move by itself as human eyes can– all the scene fades into greyish color intensities and only pure grey contour of the sheet of paper remains. These effects can be tested by anyone looking straight to a contrasted object.

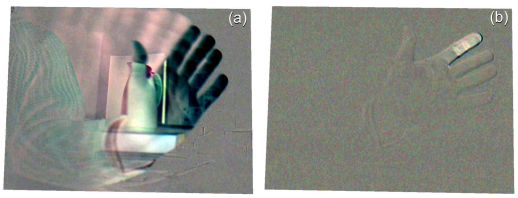


Fig. 9 Ganglion multi functionalities: (a) fast movement; (b) sligh motion.

Figure 9 proposes a comparison on a ganglion layer between a fast and abrupt hand movement 9(a) and about no movement but a slight one: the top of the forefinger –see 9(b). From these two pictures five properties can be observe: faster the movement is, brighter the contrast seems –see (a) and (b); behind contour object also appears contrasted –see (a); similarly to comics tips, directions of moving objects are also obvious –see (a); all the scene seems to disappear as movement decrease –see (b); at the contrary, even resting object appears when a movement occurs –see (a).

Authors would like to point out that characteristics and properties demonstrated using the model we presented have been identified because real-time computation was possible. Hence, studying a *GPU* based architecture was not only a way to have frame rate improvements but rather an open door to strongly improve research in the field of artificial retina.

6 Conclusion and future work

Focusing on the OPL and IPL layers, we presented an efficient and original model in order to simulate retina cellular layer behaviors. We showed that real-time simulation was possible using GPU programming based on OpenGL Shader Language. After describing the biological retina, we first identified how its layer behaviors could be interpreted in terms of a pipeline. We then proposed a computer science design for this pipeline, detailing our model of cellular automaton GPU approach and its relationship with the CPU program. To prove the model efficiency, we specified the resulting time expenses and we compared this new GPU model with a previous CPU-based approach. In particular, we demonstrated that the current GPU-based approach is at least twenty times as fast as a classical CPU model.

Many interesting aspects are to be further improved. We are currently extending our model to a simulation specialized in contour real-time restoration and vectorisation. A complete analysis of the influence of parameters –about twenty in the whole pipeline– defined in this model should reveal interesting properties. Moreover, another key area for further work is the influence of low brightness visualization using rod cells. Photoreceptors (*i.e.* cone and rod cells) spatial positioning and densities in the eye sphere have not been taken into account in this paper. Such a three-dimensional approach would lead to the design of a model able to focus automatically on image specific areas, thus reducing the analysis cost tremendously. Furthermore, a non-trivial study would be to add the RGB data flow as the biological retina does, hence identifying separate colorimetric intensities. To conclude, we are convinced that those new fields of research would imply the study of multiple pipelined architectures and would lead to fast, low-cost , and efficient artificial retina.

A Algorithm

To help understanding software architecture, here follows the general algorithm of our retina model.

1. Initialisation
   (a) init environment
   (b) init classical OpenGL
   (c) init OpenGL shading language
      – ARB calls
      – auto-compile *shaders* programs
   (d) input source (for instance Webcam or Avi file)
   (e) init texture
2. Main loop
   (a) Run retina pipeline (see figure 4)
      – open virtual projection buffer
      – cone LCB[2]:
      inputTex $\mapsto$ coneTex, call *shader*[blur-CA(coneTex)] and scan virtual buffer n_c times and *StopPipeline* if needed
      – horizontal LCB:
      coneTex $\mapsto$ persHoriTex, call *shader*[blur-CA(persHoriTex)] and scan virtual buffer n_h times, call *shader*[

[2] LCB stands for *layer cellular behavior*.

persistence(pershoriTex, horiTex)] and *StopPipeline* if needed
- bipolar LCB:
  call *shader*[subTex2D(coneTex−horiTex)], scan virtual buffer ↦ bipoTex and *StopPipeline* if needed
- amacrine LCB:
  bipoTex ↦ persAmacTex, call *shader*[persistence(persAmacTex, amacTex)], scan virtual buffer ↦ amacTex and *StopPipeline* if needed
- ganglion LCB:
  call *shader*[subTex2D(bipoTex−amacTex)], scan virtual buffer ↦ gangTex
- *StopPipeline*:
  close virtual projection buffer

(b) User event managment (mouse, keyboard interuptions...)
(c) Information managment (current view, parameters, fps...)
(d) scene rendering

3. Release dynamic allocations

Acknowledgements We would like to thanks Mr Hervé Bonafos for his technical assistance and advices concerning *OpenGL Shader Language*. Special thanks are also due to Mrs Aline Caspary for the proofreading of the draft and daily services.

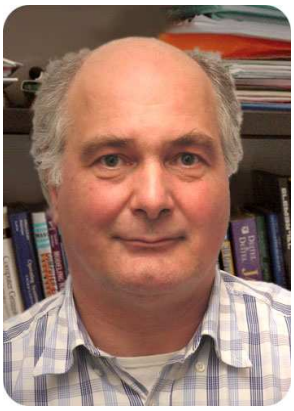
References

1. Ahnelt, P., Kolb, H.: Horizontal cells and cone photoreceptors in human retina. Comp. Neurol. **343**, 406–427 (1994)
2. Beaudot, W.: Le traitement neuronal de l'information dans la rétine des vertébrés - un creuset d'idées pour la vision artificielle. Ph.D. thesis, Institut National Polytechnique de Grenoble (1994). 228 pages
3. Bernard, T., Guyen, P., Devos, F., Zavidovique, B.: A programmable VLSI retina for rough vision. Machine vision and Applications **7**(1), 4–11 (1993)
4. Bonnet: Traité de psychologie cognitive 1. Dunod, in french (1989)
5. Buren, J.V.: The retinal ganglion cell layer, springfield, illinois edn. Charles C. Thomas (1963)
6. Chaudhuri, P., Chowdhury, R., Nandi, S., Chattopaghyay, S.: Additive Cellular Automata Theory and Applications. IEEE Computer Society Press and John Wiley & Sons (1993)
7. Conway, J.: Game of life. Scientific American 223 pp. 120–123 (1970)
8. Dacey, D.: Circuitry for color coding in the primate retina. In: Proc. Natl. Acad. Sci. USA, vol. 93, pp. 582–588 (1996)
9. Deshmukh, K., Shinde, G.: An adaptive color image segmentation. Electronic Letters on Computer Vision and Image Analysis **5**, 12–23 (2005)
10. Devillard, F., Gobron, S., Grandidier, F., Heit, B.: Implémentation par automates cellulaires d'une modélisation architecturale de rétine biologique. In: READ'05. Institut National des Télécommunications, Evry, France (2005)
11. Druon, S., Crosnier, A., Brigandat, L.: Efficient cellular automaton for 2d / 3d free-form modeling. Journal of WSCG **11**(1, ISSN 1213-6972) (2003)
12. Gobron, S., Chiba, N.: 3D surface cellular automata and their applications. The Journal of Visualization and Computer Animation **10**, 143–158 (1999)
13. Gobron, S., Chiba, N.: Crack pattern simulation based on 3d surface cellular automata. The Visual Computer **17**(5), 287–309 (2001)
14. H. Kobayashi T. Matsumoto, T.Y., Kanaka, K.: Light-adaptive architectures for regularization vision chips. In: Proc. Neural Networks, vol. 8:1, pp. 87–101 (1995)
15. Harris, M.: Implementation of a cml boiling simulation using graphics hardware. In: CS. Dept, UNCCH, Tech. Report, vol. 02-016 (2003)
16. Kolb, H.: The neural organization of the human retina. Principles and Practices of Clinical Electrophysiology of Vision pp. 25–52 (1991)
17. Kolb, H.: How the retina works. America scientist (2003)
18. MacNeil, M., Masland, R.: Extreme diversity among amacrine cells: Implications for function. Neuron **20**(5), 971–982 (1998)
19. Mahowald, M.: Analog vlsi chip for stereocorrespondance. In: Proc. IEEE International Circuits and Systems, vol. 6, pp. 347–350 (1994)
20. Marr, D.: Vision. W. H. Freeman and Company (1982)
21. Mead, C.: Analog VLSI and neural systems. Addison-Wesley (1989)
22. Rosenfeld, A.: Picture Languages. Academic Press, New York (1979)
23. Rost, R.: OpenGL Shading Language, 2^{nd} Edition. Addison Wesley Professional (2006)
24. Shreiner, D., Woo, M., Neider, J., Davis, T.: OpenGL Programming Guide v1.4. Addison Wesley Professional (2003)
25. Tran, J., Jordan, D., Luebke, D.: New challenges for cellular automata simulation on the GPU. www.cs.virginia.edu (2003)
26. Vichniac, G.: Simulating physics with cellular automata. In: Physica, vol. 10D, pp. 96–116 (1984)
27. Wässle, H., Boycott, B.B.: Functional architexture of the mammalian retina. Physiological reviews **71:2**, 447–480 (1991)
28. Wolfram, S.: A new kind of science, 1st edn. Wolfram Media Inc. (2002)
29. X.L. Yang, K.T., Dowling, J.: Modulation of cone horizontal cell activity in the teleost fish retina. i. effects of prolonged darkness and background illumination on light responsiveness. The journal of neuroscience **8:7**, 2259–2268 (1988)

S. Gobron Specialized in software engineering and computer graphics, Stephane Gobron embarked on an international studies plan at the age of nineteen, starting with a four-year US B.Sc. degree and then he did a two-year M.Sc. degree in France. Finally, he spent five years in Japan teaching and studying for a Ph.D. degree specialized in cellular automata at Iwate University. After graduating, his research activities have focused on numerical modelization of environmental phenomena. These studies led to a large range of applied domains in numerical computation and visual modelling such as ceramics and glaze fracture propagations, metallic patina and corrosion simulations, three-dimensional material deformations, and virtual surgery. Recently, he has been teaching mainly computer science, computer engineering, physics, and computer graphics at Henri Poincare University in France at the undergraduate level. His research interests mainly include development relative to weathering, cellular automaton, dynamic cellular networks, GPU programming, and layer based retina simulation. Finally, he has a strong interest in industrial development and

international relations, especially between European Union, United States of America and Japan.

F. Devillard François Devillard was born in Luxeuil-les-Bains, France, in October 27, 1963. He received the B.Sc. in 1984 and M.Sc. degrees in 1986 from the university of Grenoble, France. He received the Ph. D. degree in 1993 in "Signal, Image and Speech" from the Institut National Polytechnique de Grenoble. Since 1994, He is Assistant Professor at the Centre de Recherche en Automatique de Nancy - Henri Poincare University in France. His first research interests were image and signal processing whereas most cases studies achieved to an implementation allowing real-time performances. Currently they are oriented toward architectural modelization of biological retina.



B. Heit Bernard HEIT was born in 1954 in France. He obtained his Ph.D. in electrical engineering in 1984 and was from then Assistant Professor. Ten years later he was promoted Associate Professor, and since 1997, he is now Full Professor at the Henri Poincare University, Nancy, France.
His research interests include monocular passive vision, image processing and their real-time computer implementation. Lately his works focus on computer implementation of retinal layer architectural model based on mechanisms observed in the biological visual system.