Machine Vision and Applications (2007) 18:331–342
DOI 10.1007/s00138-006-0065-8

ORIGINAL PAPER

Retina simulation using cellular automata and GPU programming

Stéphane Gobron · François Devillard · Bernard Heit

Received: 14 June 2006 / Accepted: 19 October 2006 / Published online: 24 January 2007
© Springer-Verlag 2007

Abstract This article shows how the architectural modelization of biological retina allows real-time performances, on standard widespread computing systems. First, we describe the biological retina with regard to its pipeline architecture, detailing its layer behaviours 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 the efficiency of our model through numerical and graphical results. We lay particular emphasis on the fact that our direct implementation of the *Graphical Processor Unit* (*GPU*) provides computation power about 20 times as fast as conventional programming.

S. Gobron
Movement and Perception laboratory,
University of Marseille 2, 163, avenue de Luminy,
13288 Marseille cedex 9, France
e-mail: stephane.gobron@univmed.fr
URL: 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,
88100 Saint-Dié-des-Vosges, France
e-mail: francois.devillard@iutsd.uhp-nancy.fr

B. Heit
UMR CNRS 7039 (CRAN—CNRS),
Henri Poincaré University, ESSTIN,
2, rue Jean Lamour, 54500 Vandœuvre-lès-Nancy, France
e-mail: bernard.heit@esstin.uhp-nancy.fr

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

1 Introduction

Today, computer vision requires increasing visual information to describe and analyse natural scenes efficiently. Visual clues such as shapes, movements, and colours bring additional information about the observed scene which are necessary 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 demonstrate outstanding achievements. The alternative bioinspired methods have produced good results but require costly computing systems to reach real-time processing. So using graphical processor unit programming to avoid both constraints mentioned above, to provide flexibility, real-time computation and rendering, is highly relevant here.

1.1 Background

We observed recently 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 [4,9,19];
- more rarely digital *Very Large Scale of Integration* (*VLSI*) components [27].

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 Sect. 2 we first describe the biological retina, emphasizing its pipelined structure and corresponding layer properties. The next two sections develop the heart of our model. Section 3 describes the data flow technical architecture; Sect. 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 Sect. 5. Finally, Sect. 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 requires well-adapted computing systems. In this article, we show that a standard personal computer solution directly using parallel GPU architecture produces spectacular performances, especially in this kind of application. Results (Sect. 5) show that direct and optimized programming using mixed algorithm—*Object-Oriented Language* (OOL [7]) *OpenGL Shader Language* (GLSL [22])—first reduces software development, compared to very low-level programming, and second, the computing times compared to a conventional method. Therefore, the proposed implementation obtains real-time performances on standard widespread computing systems.

To understand the method of implementation, we need first to understand well 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 [11]) and finally the link between data flows going through layers in view of their respective functionalities.

2.1 Biological retina

The mechanisms of visual perception [10,11]:—see Fig. 1a—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. The retina is composed of several layers, organized to filter visual information. Retinian filters behave similarly to low-pass frequency spatiotemporal filters modelized by signal processing [5,28].

Figure 1 shows the pipelined architecture of the retina: Fig. 1a being a vertical section through a human retina, we suggest grouping the main functionalities of the retina around OPL and IPL layers presented in Fig. 1b and c.

The photopic visual perception starts with the scene captured by photoreceptors (cone and rod cells). Rod cells hold photoreceptor properties specialized for low light-intensity which are not required for the current approach, which is why only the cone cells are taken into account. Image luminance is sampled, then transmitted by cone layer towards the downstream retinian and cortical layers. A pipeline of cellular layers can be identified going from cone to ganglionar layers [14]. 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 or less distant connections. Therefore, each cell can be modelized by a processor or a basic filter which contains

- several inputs established by connections with a variable number of neighbouring cells;
- an output which performs the cell activity, comparable to a low-pass frequency filter.

2.2 OPL and IPL areas of neuropil

The OPL layer is where connections between rod and cones, vertically running bipolar cells, and horizontally oriented horizontal cells occur [20]. OPL properties are generated by the synaptic triad which is composed of three kinds of interconnected cells:

- The cone cells constitute a layer of the transduction and regularization processing. Transduction converts luminance into electrochemical potentials aimed at the downstream layers. The regularization consists in filtering input signals with a light low-pass frequency filtering. The cone cells are defined by their shapes (midget, diffuse,...), their types of response (on, off) and their functions (colour sensibilities: LMS, respectively red, green, and blue colours). In addition, their behaviours depend locally on the luminance intensity and contrast [29].

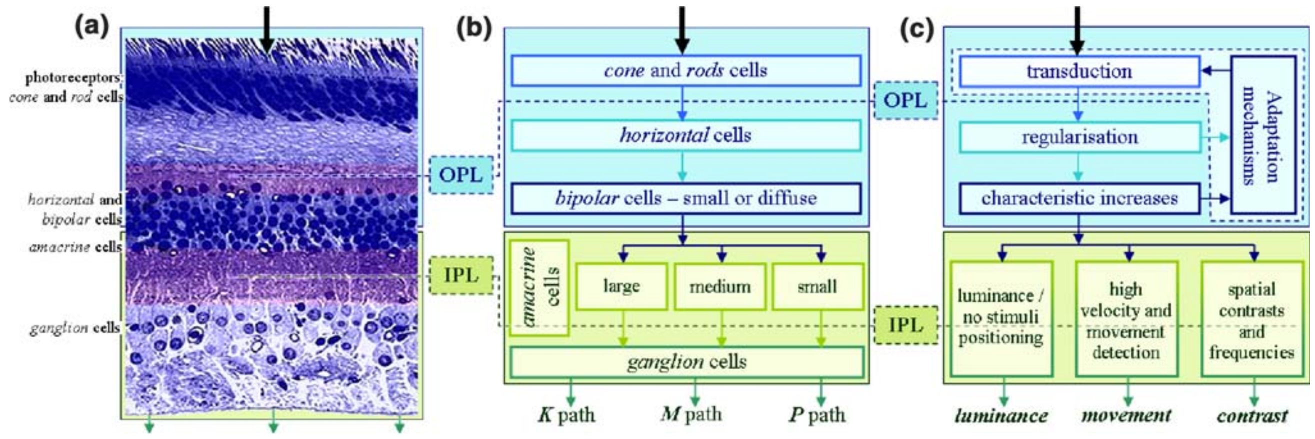

Fig. 1 Three ways of showing the human retina architecture, in terms of **a** biological section through a real retina, **b** region names and general pipeline of processes, **c** pipeline functionalities

- The horizontal cells constitute a layer of strong regularization 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. Like cone cells, bipolar cells vary considerably according to shapes (midget, diffuse, …), types of response (bipolar on, off,…) and functions (contrast estimation: red/green, blue/green + red,…) [12].

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

- Amacrine cells have multiple functions. Their main functions are to transform a temporal variation into a peak signal, typically similar to a high-pass frequency filtering [17].
- Ganglion cells make the difference between the bipolar cells (contrast evaluation) and the amacrine cells ouputs. The ganglion cells are classified as bipolar cells according to their shapes (midget, diffuse, …), their types of response (on, off, on + off) , and their functions (spatiotemporal contrast estimation and luminance estimation) [12].

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

2.3 The retinian data flows

Gradually, the visual information flow is divided into three known pathways with weak redundancies. The retinian output provides split information, organized along three specialized pathways:

- The koniocellular path (K), presents numerous features which make it sensitive to luminance and chrominance [6].
- The magnocellular path (M), carries out an initial processing step which is essential to movement perception. Antagonistic receptive fields are used [3].
- The parvocellular path (P), achieves a preprocessing step enabling it to extract the visual clues characterizing the objects. The cells making up the type P distance are by far the most numerous. They are small-sized, tonic (of less sensitivity to temporal modulations), and situated mainly in central retina. Sensitive to high spatial frequencies (thin details or small objects) and to low time frequencies, they are the seat of visual acuteness and thus code the specific position of the stimulus and the colour [3].

The previous description of the whole retina showed how difficult it is to modelize this matter accurately. The inner retinian mechanisms (feedback, adaptation, …) are complex and sometimes not even located, therefore all have not been studied in depth (for instance K pathway). The proposed modelization uses a layer organization which copies the architecture shown in Fig. 1. The whole processing approaches P pathways of the retina. The P Pathway is considered in an achromatic version. We have designed a generalized layer (shared by each kind of cell) designed to give it the main properties. This layer is repeated to build retinian architecture.

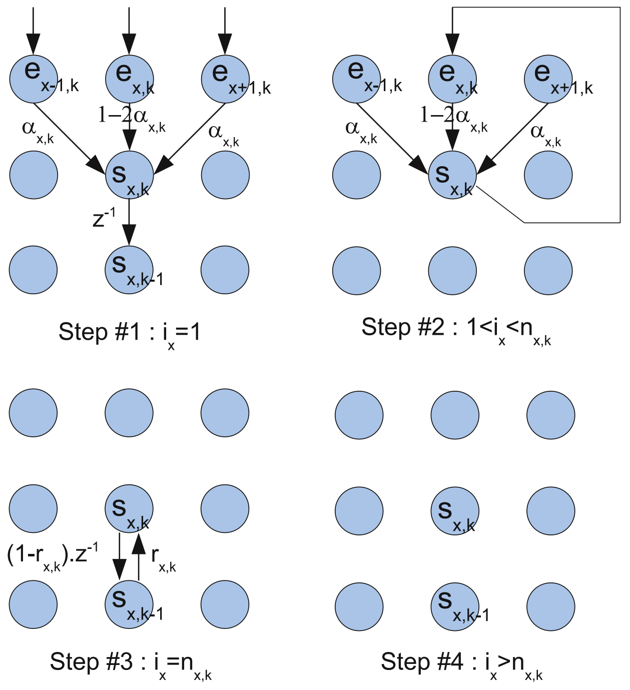

Fig. 2 Equation graphs presenting the four steps of our algorithm implemented for a $1D$ layer. The variable x represents the cell position and k the time value with unit scale samplings. z^{-1} is the unit delay operator

3 Approach

3.1 Layer modelling

Each layer is built by a network of cells. Each cell is an individually spatiotemporal low-pass filter. Figure 2 shows how our algorithm computes for a cell considered in a 1D spatial dimension network. The cell has an output s (cell activity) and takes e signal and its close neighbourhood as inputs. The performance of s is obtained by an iterative computation. Four performing steps are necessary and illustrated by Fig. 2. Variables x and k are, respectively, the location and instant using a unit sampling. All layers use our cellular algorithm with an adapted setting depending on their functionalities.

The resulting filter is composed of two categories of pipelined filters. A *Finite Impulse Response* (FIR) filter and an *Infinite Impulse Response* (IIR) filter which smooth cell inputs, respectively, spatially and temporally.

The parameters that spatially influence the cell output are n and α. They simulate the lateral interaction between neighbouring cells. The FIR filter coefficient α (see Fig. 2) performs influence between two immediate neighbouring cells. n is the number of FIR filter iterations that is applied to each cell. Thus the values α and n adjust the dendritic spreading out for each cell ($2n + 1$ by $2n + 1$). Coefficient α is adjusted to obtain a pseudo-gaussian low-pass frequency response (best approximation for $\alpha = 0.25$) [5]. 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 connections with one downstream layer. It has two exceptions for the transduction layer (luminance is the input for the cone layer) and the differentiation layers (two downstream layers outputs are inputs for bipolar and ganglion layers). Cone layers sample another luminance image I for each k value (which form a video sequence input). Spatially we consider that layer and cell output are, respectively, modelized by image and pixel in our processings. 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 cell operation and neighbourhood evolution. Thus, we have naturally chosen to model each cell layer through a cell network associated with a cellular automaton [26]. The cellular automaton constitutes a network of cells sharing common geometrical properties to which a change of state (definite set of rules) is applied according to a common synchronization. The cellular automatons enable us to approach the local properties of the retinian layers simply from an architectural and behavioural point of view. The simplicity of the cells and connections makes it possible to consider efficient implementation using GPU processors characterized by their high parallelism architectures and their reduced instruction sets. Each cell follows 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 of states associated with behavioural rules. The four steps constitute an iterative cycle of processing which is performed by the cellular automaton. The individual cell behaviour of the automaton is driven by the four 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 the layer upstream ouput for all the cells (or 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

Table 1 Table of rules of the CA driving a $1D$ retinian layer. e' and s' are the sampled internal values of respectively e and s in each CA cell

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	$s_{x,k} \leftarrow s'_{x,k}$ $i_x \leftarrow -1$	$i_x > n_{x,k}$

not successful. The cell moves then on to step 3 (duration of one iteration). The cell remains at step 4 (variable duration) as long as all the other cells individually have not reached this step.

For a row of cells, this processing method is presented in Fig. 2. The computing steps for the cell at location x are given by a counter value i_x (initialized by -1 at the start). The following Table 1 shows their actions and activation conditions for all processing steps:

The 2D release of the network described in Fig. 2 slightly complicates the whole processing. The network grid has a square shape and we have chosen to use only four connections for each cell. Thus each cell is connected with four cells of its immediate neighbourhood. The four connections used are, respectively, towards horizontal and vertical directions centred on the processed cell.

4 GPU retina simulation

4.1 Retina pipeline software design

Similarly to [2]—which suggests a CPU-based OPL simulation—our representation of the retina pipeline is modelized by using a succession of two-dimensional matrices often associated with a cellular automaton (CA) for computation.

For readers not familiar with CA, Gérard Vichniac [8] gives the following definition fitting issues of current retina model: *Cellular automata are dynamical systems where space, time, and variables are discrete. Cellular automata are capable of non-numerical simulation of physics, chemistry, biology, and others and they are 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 domains using CA. Here is a list of references presenting CA mixed with computer graphics and/or image processing:

- [21] and [1], respectively, for notions of grammar and computer engineering;
- [13,24,25] for CA and graphics based simulations;
- [16] for *neural networks* (similar to CA) and image processing—especially segmentation;
- [15,18,23] for CA and GPU programming.

Figure 3 shows OPL and IPL sections. They are computed by a pipelined architecture and composed of five major steps:

From an input I—that can be an *AVI* video or a *webcam*—the information is first transmitted to the cone layer. The resulting computation C is then transmitted both into 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 Fig. 3, each layer has a specific behaviour defined by one or two functions. These functions can be simulated through 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 (Table 1). Similar 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 step 3 (Table 1). 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.

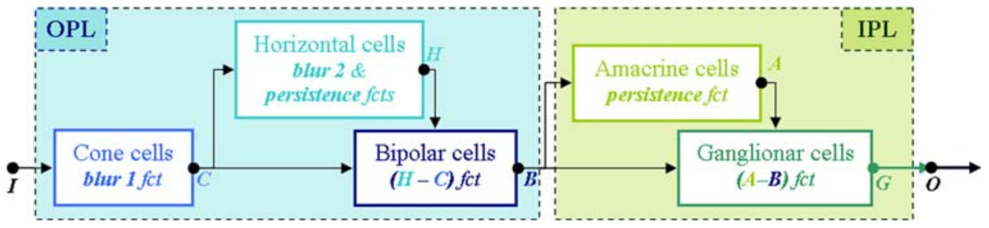

Fig. 3 Simplified retina pipeline viewed in terms of cellular layers

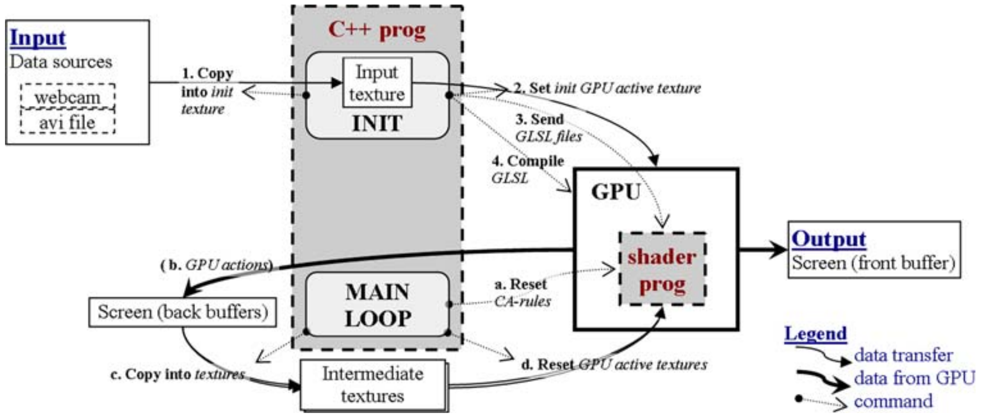

Fig. 4 Settings of different CA on the GPU

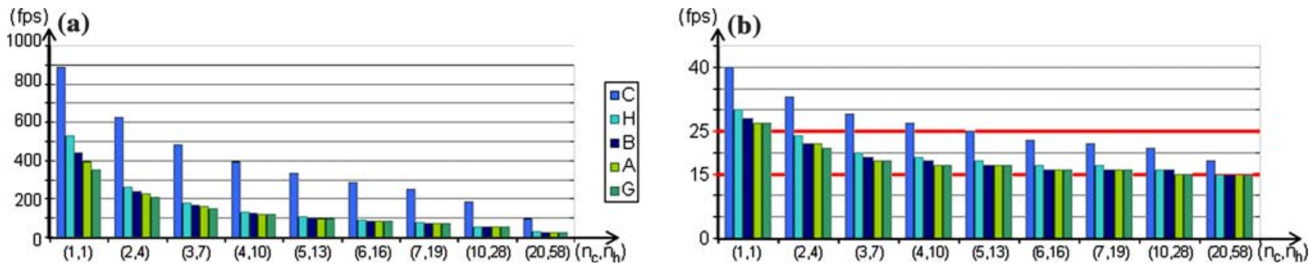

Fig. 5 Graphical interpretations of Table 2: **a** outputs using a direct input file and GPU; **b** outputs using a webcam input and GPU

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—graphical cards now have high computational potential. This potential can be accessed by using graphical processor units that can be directly programmed by assembly language or specific hardware 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 [12,15,24,25]. Note that a graphical card is composed of a set of GPUs capable of parallelizing these shaders programs and therefore is much more powerful than a single CPU of the same generation.

As mentionned in the previous section, we suggest using this powerful tool not for rendering purposes, but as a computional unit specialized in the retina pipeline data transformation. To do so, we first associate every cellular layer matrice with two-dimensional graphical textures. Second, with GLSL we associated a general cellular automaton program which can be adjust for each specific retina functionality—see Table 1.

The communication links between the CPU program (that control the retina pipeline) and the GPU program (that control local cellular behaviour) is presented in Fig. 4.

The first main task consisted in making our C++ software builds the CA *shader* program—see Appendix 6. The second task was 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 Fig. 4 we have tested in this paper two types of input animations: an AVI file and a webcam buffer. From this starting point, the CPU program leads the GPU via specific

Table 2 Output image rates in frames per second (fps)

Blur factors		GPU approach										CPU model, Webcam inputs, B-output
		Direct input file					Webcam inputs					
n_c	n_h	C	H	B	A	G	C	H	B	A	G	
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	$<< 1$

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 affect for these tests

settings of the CA *shader* code: for each specific layer behaviour, CA rules are reset. All GPU results are projected onto virtual windows, then screen-copied and sent back into different pipeline texture buffers. Finally, output texture is then shown onto the screen.

5 Results

In this section, after presenting the hardware environment used for testing, we first discuss why and how to estimate the capacities of our model (Sect. 5.2). Second, we illustrate this retina simulation with resulting images emphasizing characteristics which have been often obtained in real-time testing conditions—see Sect. 5.3.

5.1 Environment

Data from each cell in the CA are directly transmitted via OpenGL^{TM} 1.4 shading language using *ARB* extensions [7] for the best compatibility. Note that *ARB* extensions are not trivial as they lack documentation and standardization; that is why we advise using the brand new (2006) OpenGL^{TM} 2.0 [22] 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.

Note that most computations were dedicated to the graphical card and that in terms of pure memory cost, the fully developed program at the most reaches 35 Mo of RAM. Therefore, it does not require a dual core processor unit with 2 Go to obtain results similar to the ones presented in the following subsections. The minimum 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 higher.

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

5.2 GPU solution testings

Many parameters interacted with our model, therefore the testing capacities of such algorithm was not an easy task. Nevertheless, here were 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;
- last, to find execution cost of each retina layers.

As explained above, we mainly used our webcam resolution (640×480) for experiments. Our technique implied the use of graphical cards which currently limit resolution to a maximum of 4096×4096—note 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

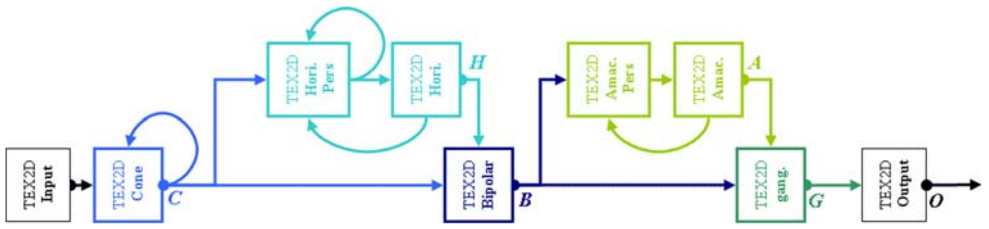

Fig. 6 Simplified retina pipeline viewed in terms of textures

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.

Figure 5a and b graphically illustrate the two right sections of Table 2. From this figure, we can infer that on the left-hand diagram, horizontal cell computation is far from being the most extended time as shown in the right-hand diagram.

For a set of n couple—see parameters of the layer modelling described in Sect. 3.1–Table 2 proposes resulting *frames per second* (fps) at C, H, B, A , and G outputs (see Fig. 6). As horizontal cell are characterized by a blur factor much higher than cone cells, we have selected n couple so that n_h increase three times faster than n_c. The use of such a regular factor is justified by Bonnet [3], p.13: sizes of receptor fields (simulated in this paper by n couple) increase linearly with the retina eccentricity.

First, on the left-hand box, the table presents values, assuming direct files are used as input—such as an AVI sequence or even a single image. Even for relatively high (n_c, n_h) values, results are always over the real-time threshold rate—around 25 fps. These high frame frequencies demonstrate that the model of artificial retina soon allows to design more advanced and powerful models (see perspectives in the Sect. 6).

Resulting time expenses when using a webcam as input are shown on the right-hand box (without the right-hand column). The sharp drop in performance between previous results and the ones schown 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 often appears jerky.

For significant comparison of our model with CPU-based works [2], 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 with CPU 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 when we compare GPU rates fps with those of CPU, respectively, 1,537 divided by 85 fps and 82 divided by 4—see Table 3.

Table 3 Input image rates in fps

GPU approach	Direct input file 1,537	Webcam inputs 82
CPU model	Direct input file 85	Webcam inputs 4

For webcam tests, high values of fps are obtained by over sampling of the video input buffer

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

- First, as both functions aim at an identical constant value for n reaching high values, the time cost for a single blur functionality computation using GPU is of approximately 0.47 ms.
- Second, cones appear 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 visible 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.

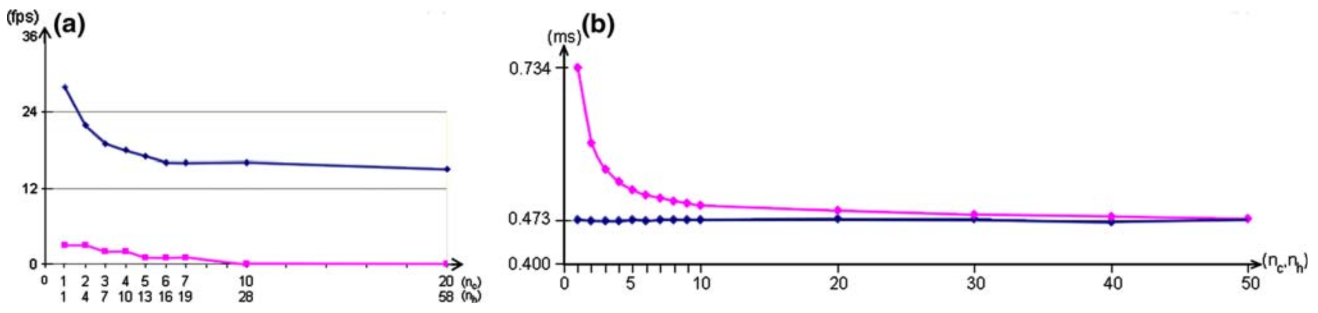

Fig. 7 Compared results: **a** CPU and GPU approach using a webcam input, graphical interpretations of Table 2—*red* for CPU and *blue* for GPU; **b** cone versus horizontal cells shader computational expenses

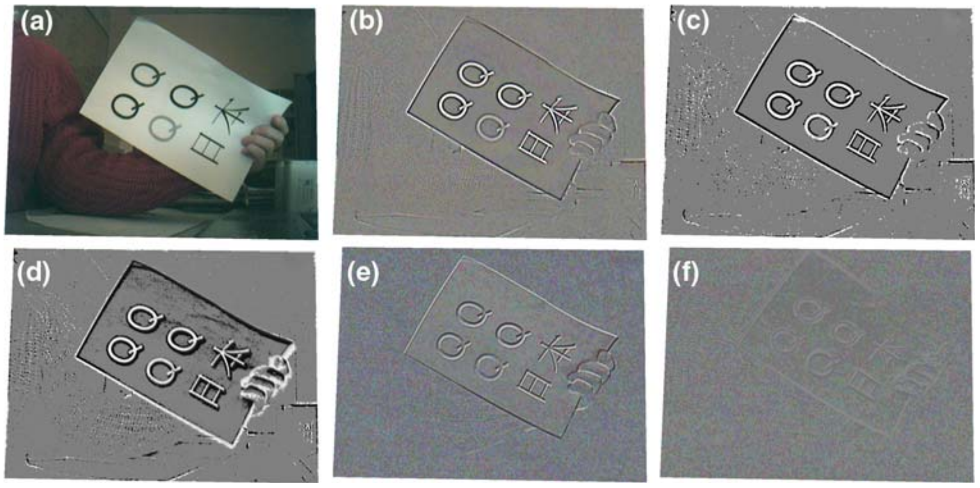

Fig. 8 Main step of the real-time retina pipeline: **a** input webcam animation; **b** bipolar output; **c** saturated (i.e. [0.0, 0.5, 1.0]) bipolar output; **d** amacrine output; **e** and **f** ganglion outputs, respectively, with and without movements

5.3 Graphics results

Figure 8 shows the main resulting steps of the retina pipeline using webcam inputs. As main viewed object, we used a sheet of printed paper with a large, highly 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 used the term Japanese kanjis *ni-hon* (actually meaning *Japan*).

This plate shows, in particular, that from an average/low contrasted input sequence (Fig. 8a) our model produces expected behaviours for each main retina layer in real time. Figure 8b illustrates the bipolar information enhancing: localization, magnitude, and length of the light. These crucial properties can be used for edge detection. We tried to focus on that contour determination, using thresholds Δ_b on cell's brightness c_b in Fig. 8c 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 $c_b \leftarrow 0.5$.

Figure 8d presents the amacrine output cellular layer which is similar to delayed bipolar outputs.

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

Figure 9 shows a comparison, on a ganglion layer, between a fast and abrupt hand movement (Fig. 9a) and a slightly detectable one, the tip of the forefinger—see Fig. 9b. From these two pictures, five properties can be observed: the faster the movement, the brighter apparent contrast seems (see Fig. 9a, b) behind, contour object also appears contrasted (see Fig. 9a) similar to comic strips, the directions of moving objects are also obvious (see Fig. 9a); all the scene seems to disappear as movement decreases (see Fig. 9b); on the contrary, a motionless object is visible when a movement occurs (see Fig. 9a).

The authors would like to point out that characteristics and properties demonstrated by using the model

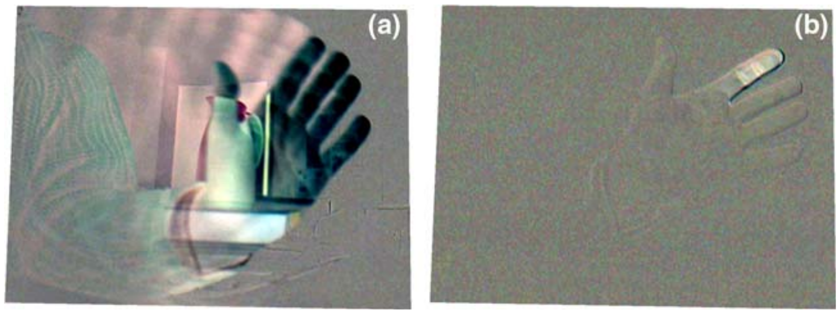

Fig. 9 Ganglion multi functionalities: **a** fast movement; **b** slight motion

we presented have been identified in so far as real-time computation was possible. Hence, studying a GPU-based architecture was not only a way to have frame rate improvements, but also an open door to major improvements in the field of the artificial retina.

6 Conclusion and future work

Focusing on the OPL and IPL layers, we presented a new, original model in order to simulate retina cellular layer behaviours. 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 behaviours could be interpreted in terms of a pipeline. We then proposed a computer-aided design for this pipeline, detailing our model of cellular automaton GPU approach and its relationship with the CPU program. To support 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 20 times as fast as a classical CPU model.

Many valuable aspects are to be further investigated. We are currently extending our model to a simulation specialized in contour real-time restoration and vectorization. A complete analysis of the influence of parameters—about 20 in the whole pipeline—defined in this model should reveal interesting properties. In addition, 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 capable of focusing automatically on specific image areas, thus vastly reducing the analysis cost. Furthermore, a non-trivial study would consist in adding 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 will trigger off the study of multiple pipelined architectures and would lead to fast, low-cost, and efficient artificial retina.

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

Appendix: Algorithm

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

```
1. Initialization
   (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 Fig. 6)
       – open virtual projection buffer
       – cone LCB[2]:
         inputTex ↦ coneTex, call shader[ blur-CA( coneTex ) ] and scan
         virtual buffer $n_c$ times and StopPipeline if needed
       – horizontal LCB:
         coneTex ↦ persHoriTex, call shader[ blur-CA( persHoriTex ) ]
         and scan virtual buffer $n_h$ times, call shader[ 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
```

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

References

1. Rosenfeld, A.: Picture Languages. Academic, New York (1979)
2. 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)
3. Bonnet, C.: Traité de psychologie cognitive 1. Dunod (in french) (1989)
4. Mead, C.: Analog VLSI and neural systems. Addison-Wesley, New York (1989)
5. Marr, D.: Vision. Freeman, San Francisco (1982)
6. Dacey, D.M.: Circuitry for color coding in the primate retina. Proc. Natl. Acad. Sci. USA **93**, 582–588 (1996)
7. Shreiner, D., Woo, M., Neider, J., Davis, T: OpenGL Programming Guide v1.4. Addison-Wesley Professional, New York (2003)
8. Vichniac, G.Y.: Simulating physics with cellular automata. Physica **10D**, 96–116 (1984)
9. Kobayashi, H., Matsumoto, T., Yagi, T., Kanaka, K.: Light-adaptive architectures for regularization vision chips. Proc. Neural Netw. **8**(1), 87–101 (1995)
10. Kolb, H.: The neural organization of the human retina. Principles and Practices of Clinical Electrophysiology of Vision pp. 25–52 Mosby Year Boook, St. Louis (1991)
11. Kolb, H.: How the retina works. Am. Sci. **91**, 28–35 (2003)
12. Wässle, H., Boycott, B.B.: Functional architecture of the mammalian retina. Physiol. Rev. **71**(2), 447–480 (1991)
13. Conway, J.H.: Game of life. Sci. Am. **223**, 120–123 (1970)
14. VanBuren, J.M.: The retinal ganglion cell layer. Springfield, Illinois edn. Charles C. Thomas (1963)
15. Tran, J., Jordan, D., Luebke, D.: New challenges for cellular automata simulation on the GPU. http://www.cs.virginia.edu (2003)
16. Deshmukh, K.S., Shinde, G.N.: An adaptive color image segmentation. Electr. Lett. Comput. Vis. Image Anal. **5**, 12–23 (2005)
17. MacNeil, M.A., Masland, R.H.: Extreme diversity among amacrine cells: implications for function. Neuron **20**(5), 971–982 (1998)
18. Harris, M.: Implementation of a cml boiling simulation using graphics hardware. In: CS. Dept, UNCCH, Tech. Report, vol. 02-016 (2003)
19. Mahowald, M.: Analog vlsi chip for stereocorrespondance. Proc. IEEE Inter. Circuits Syst **6**, 347–350 (1994)
20. Ahnelt, P., Kolb, H.: Horizontal cells and cone photoreceptors in human retina. Comput. Neurol. **343**, 406–427 (1994)
21. Chaudhuri, P.P., Chowdhury, R.D., Nandi, S., Chattopaghyay, S.: Additive Cellular Automata Theory and Applications. IEEE Computer Society Press and Wiley (1993)
22. Rost, R.J.: OpenGL Shading Language, 2nd edn. Addison-Wesley Professional, New York (2006)
23. Druon, S., Crosnier, A., Brigandat, L.: Efficient cellular automaton for 2D/3D free-form modelling. Journal of WSCG **11** (1, ISSN 1213-6972) (2003)
24. Gobron, S., Chiba, N.: 3D surface cellular automata and their applications. J. Vis. Comput. Animat. **10**, 143–158 (1999)
25. Gobron, S., Chiba, N.: Crack pattern simulation based on 3D surface cellular automata. Vis. Comput. **17**(5), 287–309 (2001)
26. Wolfram, S.: A New Kind of Science, 1st edn. Wolfram Media, Champaign (2002)
27. Bernard, T.M., Guyen, P.E., Devos, F.J., Zavidovique, B.Y.: A programmable VLSI retina for rough vision. Mach. Vis. Appl. **7**(1), 4–11 (1993)
28. Beaudot, W.H.A.: 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, pp. 228 (1994)
29. Yang, X.L., Tornqvist, K., Dowling, J.E.: Modulation of cone horizontal cell activity in the teleost fish retina I. Effects of prolonged darkness and background illumination on light responsiveness.. J. Neurosci. **8**(7), 2259–2268 (1988)

Author Biographies

S. Gobron Specialized in software engineering and computer graphics. Stephane Gobron embarked on an international studies plan at the age of 19, starting with a 4-year US B.Sc. degree and then he did a 2-year M.Sc. degree in France. Finally, he spent 5 years in Japan teaching and studying for a Ph.D. degree specialized in cellular automata at Iwate University. After graduating, his research activities 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 the European Union, the United States of America, and Japan.

F. Devillard François Devillard was born in Luxeuil-les-Bains, France, on October 27, 1963. He was awarded 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 has been 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 implementation allowing real-time performances. Currently his research is oriented toward the 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 in the Centre de Recherche en Automatique de Nancy–Henri Poincare University in France. Ten years later he was promoted Associate Professor, and since 1997, he has been 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 have been focusing on computer implementation of retinal layer architectural model based on mechanisms observed in the biological visual system.