Segmenting images with gradient-based edge detection using Membrane Computing

Daniel Díaz-Pernil [a,*], Ainhoa Berciano [a,c], Francisco Peña-Cantillana [b], Miguel A. Gutiérrez-Naranjo [b]

[a] CATAM Research Group, Dept. of Applied Mathematics I, University of Seville, Spain
[b] Research Group on Natural Computing, Dept. of Computer Science and AI, University of Seville, Spain
[c] Department of Didactic of Mathematics and Experimental Sciences, University of the Basque Country, Spain

Keywords:
Edge detection
Sobel algorithm Tissue
P systems Membrane
Computing CUDA

A B S T R A C T

In this paper, we present a parallel implementation of a new algorithm for segmenting images with gradient-based edge detection by using techniques from Natural Computing. This bio-inspired parallel algorithm has been implemented in a novel device architecture called CUDA™(Compute Unified Device Architecture). The implementation has been designed via tissue P systems on the framework of Membrane Computing. Some examples and experimental results are also presented.

1. Introduction

Paralleling classical digital image algorithms is a big challenge for the next years (Parker, 2010; Davies, 2012). Such paralleling is much more complex than the merely simultaneous application of the sequential algorithm to different pieces of the image. The coordination of different simultaneous processes in a whole algorithm is so hard task that commonly the parallel algorithm needs to be re-designed with only slight references to the classical one. Usually, the design of a new parallel implementation not inspired by the sequential one allows an open-mind vision of the problem and the proposal of new creative solutions.

The key point of paralleling classical sequential algorithms is the search of the efficiency and such efficiency is strongly linked to the development of new parallel hardware architectures with allows a realistic implementation of the theoretical advantages of the parallel processes.

In this paper, the matter of study is the Sobel algorithm (Sobel, 1970) for edge detection. We present a parallel implementation of the algorithm in the 3×3 and 5×5 versions. Based on a detailed study of these parallel implementations, in this paper we also introduce a new edge detection algorithm, the so called *AGP segmentator*. A preliminary experimental comparison with the parallel implementation of the 3×3 and 5×5 Sobel operator shows that the *AGP segmentator* improves the classical version of the Sobel operator.

Paralleling classical computer algorithm is currently a vivid research area where different hardware architectures (clusters, grids, FPGA, ...) propose different solutions (Khalid et al., 2011a; Khalid et al., 2011b; Ogawa et al., 2010; Sanduja and Patial, 2012). The chosen hardware architecture for our parallel implementation has been the Compute Unified Device Architecture,[1] CUDA™. This is a novel general purpose parallel computing architecture that allows the parallel NVIDIA Graphics Processing Units (GPUs) to solve many complex computational problems in a more efficient way than on a CPU. The choice of this parallel architecture is supported by several reasons. The first one is that the computing language CUDA™ allows programmers a friendly model for implementing easily parallel programs, but the main reason comes from the practical side. In the last years, there exists an increasing interest in the specialized industry for the development of more and more powerful Graphic Processing Units which can be used for general purposes. This interest leads, on the one hand, to a more economically accessible (and hence, more extended) hardware and, on the other hand, to the development of more powerful computational units.

The design of new parallel solutions needs a strong theoretical support that allows to control, to formalize, to check and even, sometimes, to formally verify new algorithms. As a novel contribution with respect to recent contributions found in the literature, the theoretical foundation of our parallel implementation of the

* Corresponding author.
E-mail addresses: sbdani@us.es (D. Díaz-Pernil), ainhoa.berciano@ehu.es (A. Berciano), frapencan@gmail.com (F. Peña-Cantillana), magutier@us.es (M.A. Gutiérrez-Naranjo).

[1] See http://www.nvidia.com/object/cuda_home_new.html.

Edge Detection algorithms is based on Natural Computing processes, namely, on Membrane Computing techniques.

As it will be shown below, Membrane Computing techniques are inspired in the flow of metabolites between cells of a living tissue or between the organelles in an eucaryotic cell. This flow of metabolites takes place in parallel in Nature and can be interpreted as a flow of information for computational purposes. Instead of a set of few instructions with complex data structures, the computation steps in a Membrane Computing device are regulated by a set of rules with a notation close to biochemical reactions. From a computational point of view, such reactions can be read as a set of *if A then B* rules where *A* and *B* are very simple data. As we will show below, this theoretical construction fits perfectly for a computational implementation within the GPU architecture.

The paper is organized as follows: firstly, we recall some preliminaries on Natural Computing and the definition of the used model of tissue P systems. Next, we provide a short description of the algorithms object of our study: thresholding and the Sobel algorithm for edge detection. In Section 4 some details of the implementation and several examples are provided. Finally, some remarks are given in the last section.

2. Natural computing

Nature is a big source of inspiration for new computational paradigms. Nature *acts* by performing changes (from microscopic biochemical reactions to ecological global variations) which can be interpreted as *computations*. Natural Computing[2] abstracts the way Nature operates, providing ideas for new computing models. It involves research where the physical support is non standard, as *DNA-based Molecular Computing* (Adleman, 1994) or *Quantum Computing* (Hirvensalo, 2004); but almost all the research lines in Natural Computing are currently supported in silicon-based computers. Among them, we can cite *Artificial Neural Networks* (McCulloch and Pitts, 1943), *Genetic Algorithms* (Holland, 1992), *Swarm Intelligence* (Engelbrecht, 2005), *Artificial Immune Systems* (de and Timmis, 2002), *Amorphous Computing* (Abelson et al., 2000), *Membrane Computing* (Păun, 2002) or *Cellular Automata* (von Neumann, 1966).

All these computational paradigms have in common the use of an alternative way of encoding the information and the use of intrinsic parallelism of natural processes. In this paper, we will use the theoretical framework of Membrane Computing for handling digital images. The use of techniques inspired in Nature for processing digital images is not new. Many problems in such processing have features which make it suitable for techniques inspired by nature. One of them is the treatment of the image can be parallelized and locally solved. Regardless how large is the picture, the segmentation process can be performed in parallel in different local areas of the picture. Another interesting feature is that the local information needed for a pixel transformation can also be easily encoded in the data structures used in Natural Computing. In the literature, we can find many examples of the use of Natural Computing techniques for dealing with problems associated to the treatment of digital images. One of the classic examples is the use of Cellular Automata (Rosin, 2006; Selvapeter and Hordijk, 2009). Other efforts are related to Artificial Neural Networks (Egmont-Petersen et al., 2002).

In Membrane Computing, there is a large tradition in the study of dealing with information structured as two dimensional objects (see, e.g., (Ceterchi et al., 2003a; Ceterchi et al., 2003b; Dersanambika and Krithivasan, 2004; Krishna et al., 2001)). The main motivation for these studies is to bring together Membrane Computing and Picture Grammars. Recently, a new research line has been opened by applying well-known Membrane Computing techniques for solving problems from Digital Imagery as *segmentation* (Christinal et al., 2009; Christinal et al., 2011; Díaz-Pernil et al., 2010b; Díaz-Pernil et al., 2011), *thresholding* (Christinal et al., 2010a), *smoothing* (Peña-Cantillana et al., 2011b) or the *symmetric dynamic programming stereo* algorithm (Gimel'farb et al., 2011).

The theoretical model used in this paper, Membrane Computing, is a model of computation inspired by the structure and functioning of cells as living organisms able to process and generate information. In particular, it focusses on membranes, which are involved in many reactions taking place inside various compartments of a cell. They act as selective channels of communication between different compartments as well as between the cell and its environment (Alberts et al., 2002). The computational devices in Membrane Computing are called *P systems* (Păun, 2000). Roughly speaking, a P system consists of a membrane structure, in whose compartments one places multisets of objects which evolve according to given rules which are usually applied in a synchronous non-deterministic maximally parallel manner.[3] We stress here on the so-called (because of their membrane structure) *tissue P Systems* (Martín-Vide et al., 2003) endowed with cell division.

2.1. Tissue P systems with cell division

In this section we present the formal bio-inspired model where we have implemented our edge detection algorithms. First of all, let us recall some basic preliminaries.

An *alphabet*, Σ, is a non empty set, whose elements are called *symbols*. An ordered sequence of symbols is a *string*. The number of symbols in a string u is the *length* of the string, and it is denoted by $|u|$. As usual, the empty string (with length 0) will be denoted by λ. A *multiset* m over a set A is a pair (A,f) where $f : A \to \mathbb{N}$ is a mapping. If $m = (A,f)$ is a multiset then its *support* is defined as $supp(m) = \{x \in A | f(x) > 0\}$ and its *size* is defined as $\sum_{x \in A} f(x)$. A multiset is empty (resp. finite) if its support is the empty set (resp. finite). If $m = (A,f)$ is a finite multiset over A, and $supp(m) = \{a_1, \ldots, a_k\}$, then it will be denoted as $m = \{\{a_1^{f(a_1)}, \ldots, a_k^{f(a_k)}\}\}$. That is, superscripts indicate the multiplicity of each element, and if $f(x) = 0$ for any $x \in A$, then this element is omitted. A *graph* G is a pair $G = (V,E)$ where V is the set of vertices and E is the set of edges, each one of which is a (unordered) pair of (different) vertices. In what follows we assume the reader is already familiar with the basic notions and the terminology underlying P systems.

Tissue P systems with cell division is a well-established P system model presented by Păun et al. in (Păun et al., 2008). The biological inspiration for considering *cell division* in this model is that alive tissues are not *static* network of cells, since cells are duplicated via mitosis in a natural way. Tissue P systems with cell division have been previously used to design solutions to **NP**-complete problems in polynomial time (see (Díaz-Pernil et al., 2007; Díaz-Pernil et al., 2008a) and the references therein).

Formally, a *tissue P system with cell division* of degree $q \geqslant 1$ is a tuple of the form

$$\Pi = (\Gamma, \Sigma, \mathcal{E}, w_1, \ldots, w_q, \mathcal{R}, i_\Pi, i_0),$$

where:

- Γ is a finite *alphabet*, whose symbols will be called *objects*;
- $\Sigma (\subset \Gamma)$ is the input alphabet;

[2] An introduction on Natural Computing can be found in (de Castro, 2007; Kari and Rozenberg, 2008).

[3] We refer to (Păun, 2002) for basic information in this area, to (Păun et al., 2010) for a comprehensive presentation and the P system web page http://ppage.psystems.eu, for the up-to-date information.

- $\mathcal{E} \subseteq \Gamma$ is the alphabet of objects in the environment;
- $w_1, \ldots, w_q$ are strings over Γ representing the multisets of objects associated with the cells in the initial configuration;
- $\mathcal{R}$ is a finite set of rules of the following form:
  (a) *Communication rules*: $(i, u/v, j)$, for $i, j \in \{0, 1, 2, \ldots, q\}$, $i \neq j, u, v \in \Gamma^*$.
  (b) *Division rules*: $[a]_i \rightarrow [b]_i [c]_i$, where $i \in \{1, 2, \ldots, q\}$ and $a, b, c \in \Gamma$;
- $i_\Pi \in \{1, 2, \ldots, q\}$ is the input cell;
- $i_0 \in \{0, 1, 2, \ldots, q\}$ is the output cell.

A tissue P system with cell division of degree $q \geqslant 1$ can be seen as a set of q cells (each one consisting of an elementary membrane) labelled by $1, 2, \ldots, q$. We will use 0 to refer to the label of the environment, i_Π is the label of the cell where the input is placed and i_0 denotes the output region (which can be the region inside a cell or the environment). The communication rules determine a virtual graph, where the nodes are the cells and the edges indicate if it is possible for pairs of cells to communicate directly. This is a dynamical graph, because new nodes can appear produced by the application of division rules.

The strings $w_1, \ldots, w_q$ describe the multisets of objects initially placed in the q cells of the system. We interpret that $\mathcal{E} \subseteq \Gamma$ is the set of objects placed in the environment, each one of them in an arbitrarily large amount of copies.

The communication rule $(i, u/v, j)$ can be applied over two cells i and j such that u is contained in cell i and v is contained in cell j. The application of this rule means that the objects of the multisets represented by u and v are interchanged between the two cells. Note that either i or j can be equal to 0 and in this case the objects are interchanged between one cell and the environment.

The division rule $[a]_i \rightarrow [b]_i [c]_i$ is applied over a cell i containing object a. The application of this rule divides this cell into two new cells with the same label. All the objects in the original cell are replicated and copied in each of the new cells, with the exception of the object a, which is replaced by the object b in the first one and by c in the other one. Rules are used as usual in the framework of Membrane Computing, that is, in a maximally parallel way (a universal clock is considered). In one step, each object in a cell can only be used for one rule (non-deterministically chosen when there are several possibilities), but any object which can participate in a rule of any form must do it, i.e., in each step we apply a maximal set of rules. This way of applying rules has only one restriction when a cell is divided, the division rule is the only one which is applied for that cell in that step; the objects inside that cell cannot be communicated in that step.

The cells obtained by division have the same labels as the original cell and if a cell is divided, its interaction with other cells or with the environment is blocked during the mitosis process. In some sense, this means that while a cell is dividing it closes the communication channels with other cells and with the environment.

A *configuration* is an instantaneous description of the P system. Given a configuration, we can perform a computation step and obtain a new configuration by applying the rules in a parallel manner as it is shown above. A *computation* is a sequence of computation steps such that either it is infinite or it is finite and the last step yields a halting configuration (i.e., no rules can be applied to it). Then, a computation halts when the system reaches a halting configuration.

2.2. Example

Let us consider the following tissue P system with cell division of degree 3, $\Pi = (\Gamma, \Sigma, \mathcal{E}, w_1, w_2, w_3, \mathcal{R}, i_\Pi, i_0)$, where $\Gamma = \{a_1, a_2, b, c, p, q, r, x\}$, $\Sigma = \{a_1, a_2\}$ and $\mathcal{E} = \{x\}$. The multisets in the initial configuration are $w_1 = b, w_2 = c$ and $w_3 = r$. The set of rules are $R_1 \equiv (1, a_1 b/x^3, 0), R_2 \equiv (1, x/p, 2), R_3 \equiv (1, x/q, 2)$, $R_4 \equiv (1, q/r, 3)$ and $R_5 \equiv [c]_2 \rightarrow [p]_2 [q]_2$. Finally, the input cell is $i_\Pi = 1$ and the output cell is $i_0 = 3$. Notice that rules $R_1 \ldots, R_4$ determine a virtual graph with the cells as nodes. From R_2 and R_3 we can consider an edge between cell 1 and cell 2. Analogously, R_4 determines an edge between cell 1 and cell 3. We also know by rule R_1 that cell 1 can trade some objects with the environment.

Let us consider as input of our computation the multiset $a_1 a_2^2$ (one copy of a_1 and two copies of a_2) placed in the input cell 1. By considering the input, the initial configuration C_0 has three cells, labelled with 1, 2, 3 and with the multisets $w_1 = a_1 a_2^2 b$, $w_2 = c$ and $w_3 = r$. In the first step of computation, rules R_1 and R_5 are applied. Rule 1 interchanges the objects $a_1 b$ from cell 1 with three copies of x taken from the environment. Rule 5 divides the cell 2. Hence, the configuration C_1 has four cells: two of them labelled with 1 and 3 (respectively) and the other two cells have the label 2. The multiset in the cell labelled by 1 is $w_1 = x^3 a_2^2$, the cell labelled by 3 contains the multiset $w_3 = r$ and the cells labelled by 2 have, respectively, the multisets $w_2 = p$ and $w_2 = q$. In the following step of computation rules R_2 and R_3 are applied. These rules send one copy of the object x to the corresponding cell labelled by 2 against one copy of p and q (respectively). Therefore, the configuration C_2 has the same four cells as in the configuration C_1, but with the multisets $w_1 = x a_2^2 pq, w_3 = r$ and the cells labelled by 2 have the same multiset $w_2 = x$. Finally, in the third computation step the rule R_4 is applied and the object q in the cell 1 is interchanged with the object r in the cell 3. We get the final configuration C_3 with $w_1 = x a_2^2 pr, w_3 = q$ and $w_2 = x$ in both cells labelled by 2. As the output cell is cell 3, the multiset r placed in this cell in the last configuration is the output of the configuration.

3. Edge detection and segmentation

In this paper, we present the implementation of a variant of the Sobel algorithm with tissue P systems. Before giving our bio-inspired solution, we will show how the classical thresholding algorithm can be seen from a Membrane Computing perspective.

Segmentation is the process of splitting a digital image into sets of pixels in order to make it simpler and easier to analyze. Segmentation is typically used to locate region of interest (ROI) in medical images or in satellite image by finding the frontiers among regions. Segmentation has shown its utility in bordering tumors and other pathologies, computer-guided surgery or the study of anatomical structure, but also in techniques which are not thought to produce images but it produces positional information as electroencephalography (EEG), or electrocardiography (EKG). Locating such ROI is a hard task even for an expert human eye, due mainly to problems as noise and the degradation of colours. Technically, the process consists of assigning a label to each pixel, in such way the pixels with the same label form a meaningful region.

3.1. Thresholding with membrane computing

Thresholding is one of the simplest and most widely used image segmentation techniques. Its basic aim is to obtain a binary image from a grayscale one. The idea is to split the set of pixels into two sets (black and white) depending on its brightness and a fixed value, the *threshold*. If the brightness of the pixel is greater than the threshold, then the pixel is labeled as *object*. Otherwise, it is labeled as *background*.

The basic thresholding method can be generalized in a natural way. Instead of getting a binary image by labeling the original set of pixels by $\{0, 1\}$, we can consider a larger set of labels,

Fig. 1. Thresholding with 30 classes of same size.

$\{1,\ldots,k\}$ so we obtain a final image with k grayscale levels. Another natural generalization is to replace the grayscale by another scale on the features of the pixel (brightness, intensity, color, etc.).

The process of thresholding of digital images can be considered in the framework of Membrane Computing (see (Peña-Cantillana et al., 2011a)). In order to do this, let us consider an ordered alphabet of colors, $\mathcal{C} \subset \mathbb{N}$, and let us divide it in $k \in \mathbb{N}$ intervals with the same length ($m \in \mathbb{N}$). We choose the first color of each interval as its representative. According to the basic algorithm, each pixel in the interval will be replaced by the representative of the interval. Fig. 1 shows an example of this process.

This algorithm can be easily seen in the framework of Membrane Computing. We can encode an $n \times n$ image as a set of objects a_{ij} where $i,j \in \{1,\ldots,n\}$ and $a \in \mathcal{C}$, the set of colors. Since the algorithm interchanges the color of a pixel a_{ij} with another a'_{ij}, we can consider that such trading is produced by an antiport process crossing a biological membrane. Therefore, we can consider a family of tissue P system of degree 1, where no rules for cell division are needed. Each member of the family $\mathbf{\Pi}_1(k,n)$ processes all the images of size $n \times n$ with k intervals.

$$\mathbf{\Pi}_1(k,n) = (\Gamma, \Sigma, \mathcal{E}, w_1, \mathcal{R}, i_{\Pi_1}, i_o),$$

where $\Gamma = \Sigma = \mathcal{E} = \{a_{ij} : a \in \mathcal{C},\ 1 \leqslant i,j \leqslant n\}$, $w_1 = \emptyset, R$ is the following set of communication rules:

- $(1, b_{ij}/a_{ij}, 0)$, for $1 \leqslant i,j \leqslant n$, $m = (n/k)$, $l = 0,1,2,\ldots,h-m$, $a = m \cdot l$, and $b \in \mathcal{C}, a < b \leqslant a + (m-1)$.

These rules are used to divide the set of colors in r intervals of length m. $i_{\Pi_1} = i_o = 1$.

Notice that each tissue P system of the family uses the massive parallelism of the model to interchange all the pixels through the membrane simultaneously, so, in the theoretical model, the thresholding process takes only one computational step.

3.2. Segmenting images with gradient-based edge detection using tissue P systems

The Sobel operator (Sobel, 1970; Davis, 1975; Peli and Malah, 1982; Pal and Pal, 1993) is a discrete operator which computes an approximation of the gradient of the image intensity function applied to each pixel of a grey-level image. In fact, for each point of the image, the result of the Sobel operator is either the corresponding gradient vector or the norm of this vector. It uses two different filters, in X edge and in Y edge (G_x and G_y respectively), that average the image in the direction perpendicular to the differentiation applied. Next, given the pixel a_{ij}, let us define as x_{ij} the grey level of the pixel a_{ij}.

If we consider the neighbours of the pixel a_{ij}, we can construct the following grid of the grey levels of them.

$x_{i-1,j-1}$	$x_{i-1,j}$	$x_{i-1,j+1}$
$x_{i,j-1}$	$x_{i,j}$	$x_{i,j+1}$
$x_{i+1,j-1}$	$x_{i+1,j}$	$x_{i+1,j+1}$

Then, the Sobel gradient functions are defined by:

$$G_x = \frac{1}{4}\{(x_{i-1j-1} + 2x_{ij-1} + x_{i+1j-1}) - (x_{i-1j+1} + 2x_{ij+1} + x_{i+1j+1})\}.$$

$$G_y = \frac{1}{4}\{(x_{i-1j-1} + 2x_{i-1j} + x_{i-1j+1}) - (x_{i+1j-1} + 2x_{i+1j} + x_{i+1j+1})\}.$$

The Sobel operator is determined by $G = G_x \vec{i} + G_y \vec{j}$, with the gradient norm calculated as

$$\|G\| = |G_x| + |G_y|$$

Equivalently, by using 2D-convolution operator, $*$, G_x and G_y can be identified with the following operators applied over the previous grid of pixels, A:

$$G_x = \frac{1}{4}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A \quad G_y = \frac{1}{4}\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * A$$

Here, we have presented how to apply a 3×3 Sobel operator to an image. The 5×5 Sobel operator is defined in a very similar way to 3×3 operator. Next, we will show how this algorithm can be implemented with a family of tissue P systems.

We give the following two steps to get a segmentation of digital images. First, we detect edges applying a new operator, called *AGP segmentator (A Graphical P segmentator)*, which is a variation of the Sobel operator. Secondly, we do a binarization of the resulting image of the previous step. We use the algorithm previously described.

The basic idea is to work with more information data than the 3×3 Sobel operator, but to be more efficient than 5×5 Sobel operator, always from a parallel processing point of view, as we can check in Section 4. We consider to work with a dynamical perspective. Firstly, for each pixel, we take the possible four directions and look for the appropriate direction. Later, we control the efficiency of our algorithm considering to work with only 12 pixels (the 5×5 Sobel operator uses 20 pixels).

So, given a digital image with n^2 pixels ($n \in \mathbb{N}$), we define a tissue P system with cellular division whose input is given by the set $\{a_{ij} : 1 \leqslant i,j \leqslant n\}$.

Next, we give outline of how we can obtain a new approximation of the intensity gradient function *AGP operator*, of an image using tissue P systems with cell division.

The functioning of a tissue P system of the family consists on the following stages:

1. Generating stage: The system creates the necessary number of copies of the pixels for the following stage. To do this, the P system uses *the environment* and *division rules*.
2. Choosing a direction stage: The system decides among four directions: west–east, northwest–southeast, north–south and northeast–southwest.
3. Gradient stage: Chosen a direction, the system approximates the intensity gradient function.
4. Output stage: The system sends to the output cell the results of the previous stage.

3.3. A family of tissue P systems with cell division

So, we define a family of tissue P systems to approximate an intensity gradient function of a 2D image. For each image of size n^2 with $n \in \mathbb{N}$, we consider the tissue P system and cell division of degree n^2+2:

$$\mathbf{\Pi}_2(r,n) = (\Gamma, \Sigma, \mathcal{E}, w_1, w_2, w_{(1,1)}, \ldots, w_{(n,n)}, w_{(1,1)'}, \ldots, w_{(n,n)'} \mathcal{R}, i_{\Pi_2}, o_{\Pi_2}),$$

where

- $\Gamma = \Sigma \cup \mathcal{E} \cup \{\beta, T, \overline{\gamma}\}$,
- $\Sigma = \{a_{ij} : a \in \mathcal{C},\ 1 \leqslant i,j \leqslant n\}$,
- $$\begin{aligned}\mathcal{E} = {} & \{t_i, (t,1)_i, (t,2)_i, (t,3)_i, \alpha_i : 1 \leqslant i \leqslant n\} \\ & \cup \{a_{ij}, (a,l)_{ij}, :\leqslant i,j \leqslant n,\ 1 \leqslant l \leqslant 6,\ a \in \mathcal{C}\} \\ & \cup \{z_i : 1 \leqslant i \leqslant \beta_1 + 1, \\ & \beta_1 = \lceil \log_2 |\mathcal{C}| \rceil\} \cup \{\delta_i, \overline{\delta}_i, \delta'_i, \delta''_i :\ 1 \leqslant i \leqslant 4\} \\ & \cup \{y_1, y_2, x_1, x_2, x_3\} \cup \{(A,l)_{ij} : 1 \leqslant i,j \leqslant n, 1 \leqslant l \leqslant 4\} \\ & \cup \{A_l : 1 \leqslant l \leqslant 4\} \cup \{p_1, p_2, q_1, q_2, q_3, \mu, \} \\ & \cup \{o_l : 1 \leqslant l \leqslant \beta_1 + 1\}\end{aligned}$$
- $w_1 = \alpha_1;\ w_2 = t_1, \ldots, t_n; w_{(1,1)} \ldots = w_{(n,n)} = T,\ \overline{\gamma}^{255}$,
- $\mathcal{R}$ is the following set of communication rules:
  1. $(1, a_{ij}/(a,1)_{ij}(a,2)_{ij}(a,3)_{ij}, 0)$ for $1 \leqslant i,j \leqslant n$ and $a \in \mathcal{C}$,
  2. $(2, t_i/(t,1)_i, 0)$ for $1 \leqslant i \leqslant n$,
  3. $(1, (a,1)_{ij}(a,2)_{ij}/\lambda, 2)$ for $1 \leqslant i,j \leqslant n$,
  4. $(2, (t,1)_i/(t,2)_i, 0)$ for $1 \leqslant i \leqslant n$,
  5. $(1, \alpha_i/\alpha^2_{i+1}, 0)$ for $1 \leqslant i \leqslant n$,
  6. $(2, (a,1)_{ij}/(a,4)^4_{ij}, 0)$ for $1 \leqslant i,j \leqslant n$,
  7. $(2, (t,2)_i/(t,3)_i, 0)$ for $1 \leqslant i \leqslant n$,
  8. $[(t,3)_i]_2 \to [\beta]_2[\beta]_2$ for $1 \leqslant i \leqslant n$,
  9. $(1, (a,3)_{ij}\alpha_{n+1}/(a,5)_{ij}, 0)$ for $1 \leqslant i,j \leqslant n$,
  10. $(1, (a,5)_{ij}/T, (i,j))$ for $1 \leqslant i,j \leqslant n$,
  11. $((i,j), (a,5)_{ij}/(a,6)_{ij} z_1, 2)$ for $1 \leqslant i,j \leqslant n$,
  12. $((i,j), z_i/z^2_{i+1}, 0)$ for $i = 1, \ldots, \beta_1$,
  13. $$((i,j), (a,6)_{ij} \Big/ \begin{matrix}(b,2)_{i-1j-1} & (c,2)_{i-1j} & (d,2)_{i-1j+1} \\ (e,2)_{ij-1} & (a,2)_{ij} & (f,2)_{ij+1} \\ (g,2)_{i+1j-1} & (h,2)_{i+1j} & (k,2)_{i+1j+1}\end{matrix}, 2)$$ for $2 \leqslant i,j \leqslant n+1$,
  14. $((i,j), (b,2)_{i-1j-1}(k,2)_{i+1j+1}/\delta^b_1 \overline{\delta}^k_1, 0)$ for $1 \leqslant i,j \leqslant n$ and $b,k \in \mathcal{C}$,
  15. $((i,j), (c,2)_{i-1j}(h,2)_{i+1j}/\delta^c_2 \overline{\delta}^h_2, 0)$ for $1 \leqslant i,j \leqslant n$ and $c,h \in \mathcal{C}$,
  16. $((i,j), (d,2)_{i-1j+1}(g,2)_{i+1j-1}/\delta^d_3 \overline{\delta}^g_3, 0)$ for $1 \leqslant i,j \leqslant n$ and $d,g \in \mathcal{C}$,
  17. $((i,j), (e,2)_{ij-1}(f,2)_{ij+1}/\delta^e_4 \overline{\delta}^f_4, 0)$ for $1 \leqslant i,j \leqslant n$ and $e,f \in \mathcal{C}$,
  18. $((i,j), \delta_l \overline{\delta}_l/\lambda, 0)$ for $1 \leqslant i,j \leqslant n$ and $1 \leqslant l \leqslant 4$,
  19. $((i,j), z_{(\beta_1+1)}/y_1 z^4_{(\beta_1+2)}, 0)$ for $1 \leqslant i,j \leqslant n$,
  20. $((i,j), z_{(\beta_1+2)}\delta_l/\delta\prime_l, 0)$ for $1 \leqslant i,j \leqslant n$ and $l = 1,2,3,4$,
  21. $((i,j), z_{(\beta_1+2)}\overline{\delta}_l/\delta'_l, 0)$ for $1 \leqslant i,j \leqslant n$ and $l = 1,2,3,4$,
  22. $((i,j), y_1/y_2, 0)$,
  23. $((i,j), \delta'_l\delta'_{l+1}/\lambda, 0)$ for $1 \leqslant i,j \leqslant n$ and $l = 1,3$,
  24. $((i,j), y_2/x_1 y^2_3, 0)$ for $1 \leqslant i,j \leqslant n$,
  25. $((i,j), x_l/x_{l+1}, 0)$ for $1,2$ and $l = 1, \ldots, \beta_1 + 1$,
  26. $((i,j), y_3\delta'_l/\delta''_l, 0)$ for $1 \leqslant i,j \leqslant n$ and $l = 1,2,3,4$,
  27. $((i,j), \delta''_l\delta''_k/\lambda, 0)$ for $1 \leqslant i,j \leqslant n$ and $1 \leqslant l < k \leqslant 4$,
  28. $((i,j), x_3\delta''_l/(A,l)_{ij}, 0)$ for $1 \leqslant i,j \leqslant n$ and $l = 1,2,3,4$,
  29. $((i,j), (A,l)_{ij}/A_l s_1, 0)$
  30. $((i,j), s_l/s^2_{l+1}, 0)$ for $1 \leqslant i,j \leqslant n$ and $l = 1, \ldots, 24\beta_1$,
  31. (a) $$((i,j), A_1 \Big/ \begin{matrix}(a,4)_{i-1j-2} & (b,4)^3_{i-1j-1} & (g,4)^3_{i-1j+1} & (h,4)_{i-1j+2} \\ (c,4)^2_{ij-2} & (d,4)^4_{ij-1} & (k,4)^4_{ij+1} & (l,4)^2_{ij+2} \\ (e,4)_{i+1j-2} & (f,4)^3_{i+1j-1} & (o,4)^3_{i+1j+1} & (p,4)_{i+1j+2}\end{matrix}, 2)$$ for $2 \leqslant i,j \leqslant n+1$ and $a,b,c,d,e,f,g,h,k,l,o,p \in \mathcal{C}$,
  (b) $$((i,j), A_2 \Big/ \begin{matrix}(a,4)^2_{i-2j-2} & (b,4)_{i-2j-1} & & (g,4)^3_{ij+1} \\ (c,4)_{i-1j-2} & (d,4)^4_{i-1j-1} & (e,4)^3_{i-1j} (h,4)^3_{i+1j} & (k,4)^4_{i+1j+1} (l,4)_{i+1j+2} \\ & (f,4)^3_{ij-1} & (o,4)_{i+2j+1} (p,4)^2_{i+2j+2} & \end{matrix}, 2)$$ for $2 \leqslant i,j \leqslant n+1$ and $a,b,c,d,e,f,g,h,k,l,o,p \in \mathcal{C}$,
  (c) $$((i,j), A_3 \Big/ \begin{matrix}(a,4)_{i-2j-1} & (b,4)^2_{i-2j} & (c,4)_{i-2j+1} (g,4)^3_{i+1j-1} & (h,4)^4_{i+1j} & (k,4)^3_{i+1j+1} \\ (d,4)^3_{i-1j-1} & (e,4)^4_{i-1j} & (f,4)^3_{i-1j+1} (l,4)_{i+2j-1} & (o,4)^2_{i+2j} & (p,4)_{i+2j+1}\end{matrix}, 2)$$ for $2 \leqslant i,j \leqslant n+1$ and $a,b,c,d,e,f,g,h,k,l,o,p \in \mathcal{C}$,
  (d) $$((i,j), A_4 \Big/ \begin{matrix} & (a,4)_{i-2j+1} & (b,4)^2_{i-2j+2} & & (g,4)^3_{ij-1} \\ (c,4)^3_{i-1j} & (d,4)^4_{i-1j+1} & (e,4)_{i-1j+2} (h,4)_{i+1j-2} & (k,4)^4_{i+1j-1} & (l,4)^3_{i+1j} \\ & (f,4)^3_{ij+1} & & (o,4)^2_{i+2j-2} (p,4)_{i+2j-1} & \end{matrix}, 2)$$ for $2 \leqslant i,j \leqslant n+1$ and $a,b,c,d,e,f,g,h,k,l,o,p \in \mathcal{C}$,
  32. $((i,j), (z,4)_{ij}/\gamma^z_1, 0)$ for $1 \leqslant i,j \leqslant n$ and $z = a,b,c,d,e,f \in \mathcal{C}$,
  33. $((i,j), (z,4)_{ij}/\gamma^z_2, 0)$ for $1 \leqslant i,j \leqslant n$ and $z = g,h,k,l,o,p \in \mathcal{C}$,
  34. $((i,j), \gamma_1\gamma_2/\lambda, 0)$ for $1 \leqslant i,j \leqslant n$,
  35. $((i,j), s_{24\beta_1+1}\gamma_i\gamma_l/p_1\gamma\prime, 0)$ for $1 \leqslant i,j \leqslant n, l = 1,2$,
  36. $((i,j), p_1q_1/p_2q_2, 0)$ for $1 \leqslant i,j \leqslant n$,
  37. $((i,j), \overline{\gamma}\gamma\prime/\lambda, 0)$ for $1 \leqslant i,j \leqslant n$,
  38. $((i,j), q_2/q_3, 0)$ for $1 \leqslant i,j \leqslant n$,
  39. $((i,j), p_2\gamma\prime/\mu, 0)$ for $1 \leqslant i,j \leqslant n$,
  40. $((i,j), \mu/(0,6)_{ij}, 0)$ for $1 \leqslant i,j \leqslant n$,
  41. $((i,j), p_2q_3/q_4, 0)$ for $1 \leqslant i,j \leqslant n$,
  42. $((i,j), q_4\overline{\gamma}/(1,6)_{ij}o_1, 0)$ for $1 \leqslant i,j \leqslant n$,
  43. $((i,j), \overline{\gamma}(l,6)_{ij}/(l+1,6)_{ij}, 0)/$ for $1 \leqslant i,j \leqslant n$ and $l = 1, \ldots |\mathcal{C}|$,
  44. $((i,j), o_l/o_{l+1}, 0)/$ for $1 \leqslant i,j \leqslant n$ and $l = 1, \ldots \beta_1$,
  45. $((i,j), o_{\beta_1+1}(z,6)_{ij}/a_{ij}, 1)/$ for $1 \leqslant i,j \leqslant n$ and $a,z \in \mathcal{C}$,
- $i_{\Pi_2} = 1$
- $i_o = 1$.

3.4. Overview of a computation

The input data of a tissue P system of our family is an input image of size $n \times n$. This image is codified by objets of the type a_{ij} where $a \in \mathcal{C}$ and $1 \leqslant i,j \leqslant n$. The computation of a P system of our family is split in four stages. When the input objects arrive to the cell 1, the computation begins. This first stage has a technical meaning: the system prepares the copies of the input objects to send them to the cell 2. So, we can divide this cell as many times as the number of copies are needed by the system. This stage finishes with the rules of type 9.

Then, the second stage starts sending (by each pixel of the input image) an object codifying the pixel (i,j) to the cell (i,j). For each cell (i,j) (for each pixel a_{ij}), a direction (west-east, northwest-southeast, north–south and northeast-southwest) is chosen (This information will be used in the next stage). To do this choice, we take the value of the adjacent pixels to a_{ij} (rule 13) and we make up four pairs of pixels taking their positions (rules 14 to 17): $(i,j-1)$ and

$(i,j+1),(i-1,j-1)$ and $(i+1,j+1),(i-1,j)$ and $(i+1,j)$ and finally $(i-1,j+1)$ and $(i+1,j-1)$. Then, their values are taken to do a substraction (rule 18). We keep the absolute value of these numbers (rules 19 to 21) and select the biggest value of them (rules 22 to 28). To do the last step we do the following: we codify these four values as the number of copies of four different objects (δ'_l with $l=1,\ldots,4$). We take two pairs of these objects and do a substraction (rule 23). We keep a number of copies of two different objects (it does not need the same number of copies) codifying each object with all their copies in only one direction. So, we have eliminated two possible directions. Then, we repeat the process with other substraction (rule 27) and keep only one direction (rule 28).

So, the direction given by the maximum value among the four obtained is chosen. The P system finishes this stage with the rules of type 28.

The third stage starts with the rules of type 29. The tissue P system brings new copies of the cells 2 to use them in each one of the cells (i,j) (rules 31, four cases, one four each possible direction). We do an approximation of the gradient vector of the intensity function, but not working with 3×3 pixels, but working with a 5×5 grid and we only take $6+6$ pixels from one of the four following possibilities:

$x_{i-1,j-1}$	$x_{i-1,j}$	$x_{i-1,j+1}$
$x_{i,j-1}$	$x_{i,j}$	$x_{i,j+1}$
$x_{i+1,j-1}$	$x_{i+1,j}$	$x_{i+1,j+1}$

The choice is done by the direction considered as appropriate in the previous stage. Then, we add the two values obtained with absolute value, one for the positive numbers (rule 32) and a second one for the negative numbers (rule 33). Finally, we do a substraction (rule 34) and normalize our result dividing by 2 (rule 35). This manner we have approximated the value of $|G|$. This stage finishes with the rules of type 35.

The last stage is devoted to the output of the P system. First, for each (i,j) we take the inverse value of the obtained value (rule 37). We should take into account the cases where our value is less than 0. In this case, we consider the value 0 associated to our pixel (rules 39 and 40). Finally, we add our value to the pixel (rules 42 and 43) generating an object $(z,6)_{ij}$ and send it to the output cell (rule 44 and 45), in our case the cell 1. In this way, we obtain a new image ready to do a thresholding and achieve a segmentation of the original image.

3.5. Complexity and necessary resources

Bearing in mind that the size of the input data is $O(n^2)$, the amount of necessary resources to define the systems of our two families and the complexity of our solutions can be observed in the Table 1.

Table 1
Complexity and necessary resources.

Complexity	AGP operator	Thresholding
Number of parallel steps of a computation	$n+26\lceil\log_2\lvert\mathcal{C}\rvert\rceil+17$	1
Necessary resources		
Size of the alphabet	$O(n^2)$	$O(n^2)$
Initial number of cells	n^2	1
Initial number of objects	$256n^2+n+1$	0
Number of rules	$O(n^2)$	$O(n^2)$
Upper bound for the length of the rules	$\lvert\mathcal{C}\rvert$	2

4. Experimental simulation

Simulation of different variants of P systems have been widely studied in the last years. Since there does not exist implementations of P systems *in vivo* nor *in vitro*, the natural way to explore the behavior of designed P systems is to simulate them in conventional computers. A short description of some of these simulators can be found in (Díaz-Pernil et al., 2010a; Gutiérrez-Naranjo et al., 2006). In (Borrego-Ropero et al., 2007), a first simulator for tissue P systems was presented. In (Díaz-Pernil et al., 2011), a sequential simulator for solving a problem from Digital Imagery with Membrane Computing techniques was presented. Currently, a big effort is being developed in the *P-lingua project* (Díaz-Pernil et al., 2008b), by combining an efficient simulation engine with an *ad hoc* programming language.

Recently, a new research line in the simulation of P systems has started by using novel technologies as *Field Programmable Gate Arrays* (FPGAs) (Nguyen et al., 2010) or Compute Unified Device Architecture CUDA™(Cecilia et al., 2010; Peña-Cantillana et al., 2011b).

In this paper, the algorithms have been implemented by using CUDA™, (Compute Unified Device Architecture) (Nickolls et al., 2008; Owens et al., 2008). CUDA™is a general purpose parallel computing architecture that allows the parallel NVIDIA Graphics Processing Units (GPUs) to solve many complex computational problems in a more efficient way than on a CPU.

The way GPUs exploit parallelism differ from multi-core CPUs, which raises new challenges to take advantage of its tremendous computing power. GPU is especially well-suited to address problems that can be expressed as data-parallel computations. GPUs can support several thousand of concurrent threads providing a massively parallel environment. This parallel computation model leads us to look for a highly parallel computational technology where a parallel simulator can run efficiently. The newest generations of graphics processor units (GPUs) are massively parallel processors which can support several thousand concurrent threads. CUDA™comes with a software environment that allows developers to use C as a high-level programming language.

The experiments have been performed on a computer with a CPU AMD Athlon II x4 645, which allows to work with four cores of 64 bits to 3.1 GHz. The computer has four blocks of 512 KB of L2 cache memory and 4 GB DDR3 to 1600 MHz of main memory. The used graphical card (GPU) is an NVIDIA Geforce GT240 composed by 12 Stream Processors with a total of 96 cores to 1340 MHz. It has 1 GB DDR3 main memory in a 128 bits bus to 700 MHz. So, the transfer rate obtained is by 54.4 Gbps. The used Constant Memory is 64 KB and the Shared Memory is 16 KB. Its Compute Capability level is 1.2 (from 1.0 to 2.1).

4.1. Experiments and examples

Next, we show graphically how the simulation with CUDA™of 3×3 Sobel, 5×5 Sobel and AGP algorithms work in different images, detecting the representative edges in each case. Furthermore, we show here 2 examples (see Fig. 2) where the results of 3×3 Sobel, 5×5 Sobel and AGP can be compared in different ways (see Fig. 3, Fig. 4): first of all, with respect to edges detection (first column in Fig. 2 and 3); second, applying a thresholding (180, in second column), the resulting binarized images can be compared; and finally (in third column), using the algorithm defined in (Christinal et al., 2010b) over them, the white connected components enclosed by black connected components (we call *white holes*

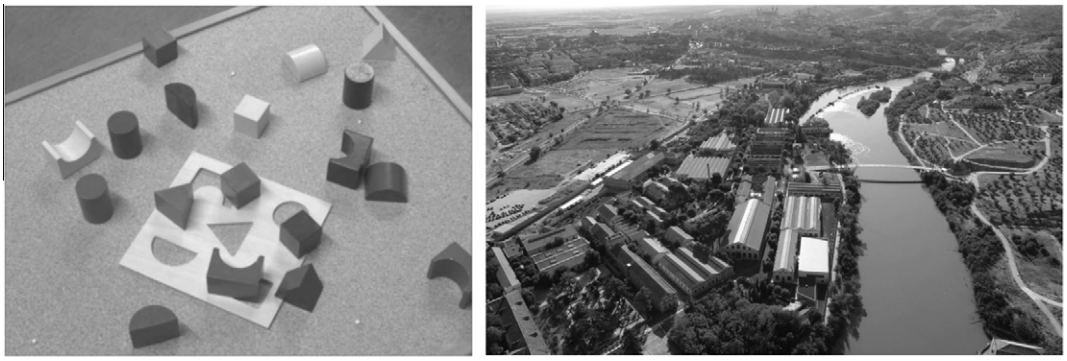

Fig. 2. Original images.

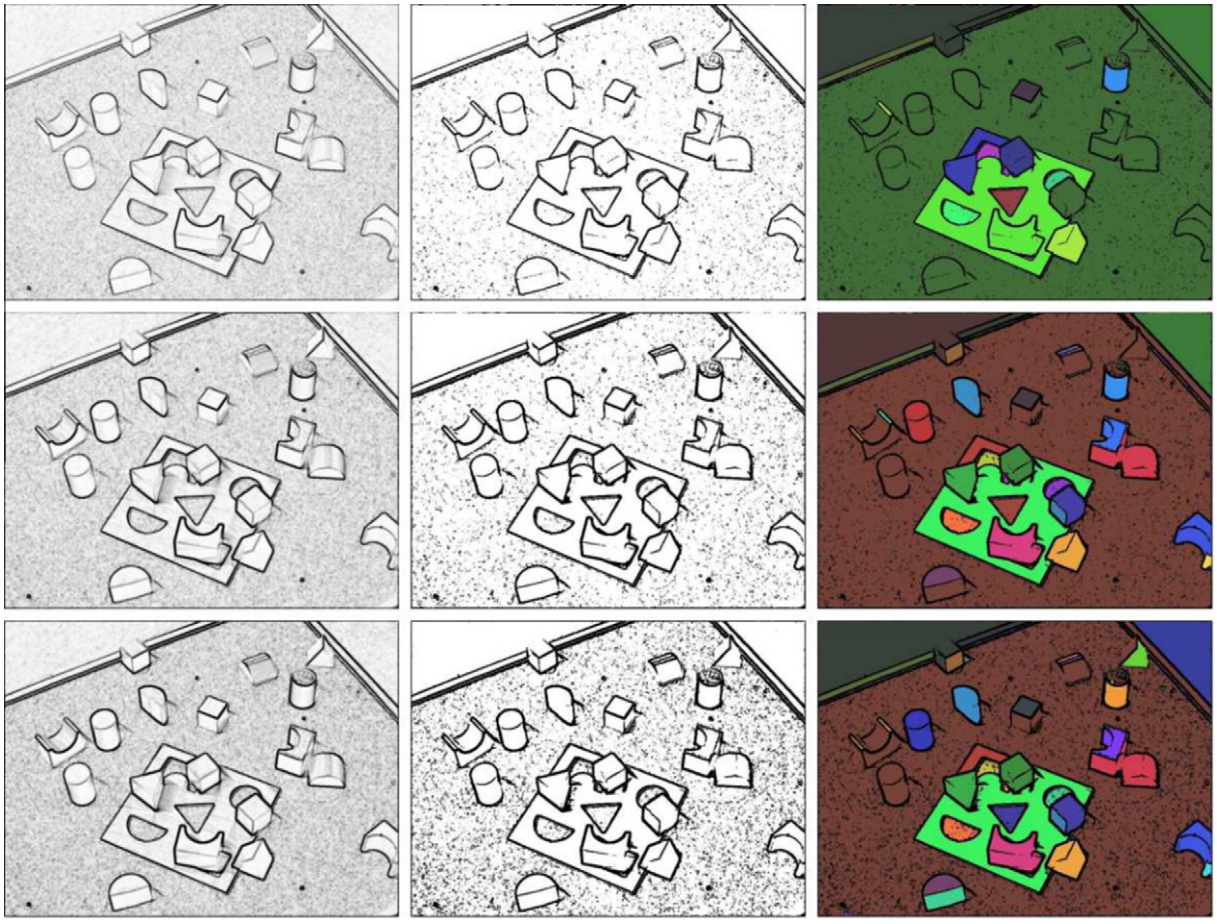

Fig. 3. First comparative example between 3×3 Sobel, 5×5 Sobel and AGP operators.

to this type of components) have been determined. In fact, it is easy to see that the number of white connected components determined (enclosed by black connected components and painted by different colors) using AGP is much bigger than using 3×3 Sobel and even bigger than using 5×5 Sobel operator.

The AGP segmentator improves the behavior of the other two operators. AGP operator uses more pixels with respect to 3×3 Sobel operator and the same number of pixels what 5×5 Sobel operator. The reason to work better with respect the last one is we work in a dynamical way. We do a previous work to find the best direction to apply an algorithm of this type. So, it is logical AGP operator obtains more white holes.

Finally, we show some experimental comparisons of the methods.

In Fig. 5 we have two graphics.[4] In the first one we compare the time cost of the sequential 3×3 Sobel and the parallel AGP algorithm depending on the size of the image. In the second one, we compare the parallel efficiency on time of the 3×3 Sobel, 5×5

[4] Size (pixel×pixel): 1. $0.5K \times 0.5K$, 2. $1K \times 1K$, 3. $1.5K \times 1.5K$, 4. $2K \times 2K$, 5. $2.5K \times 2.5K$, 6. $3K \times 3K$, 7. $3.5K \times 3.5K$, 8. $4K \times 4K$, 9. $4.5K \times 4.5K$, 10. $5K \times 5K$, 11. $5.5K \times 5.5K$, 12. $6K \times 6K$, 13. $6.5K \times 6.5K$, 14. $7K \times 7K$, 15. $7.5K \times 7.5K$, 16. $8K \times 8K$, 17. $8.5K \times 8.5K$, 18. $9K \times 9K$.

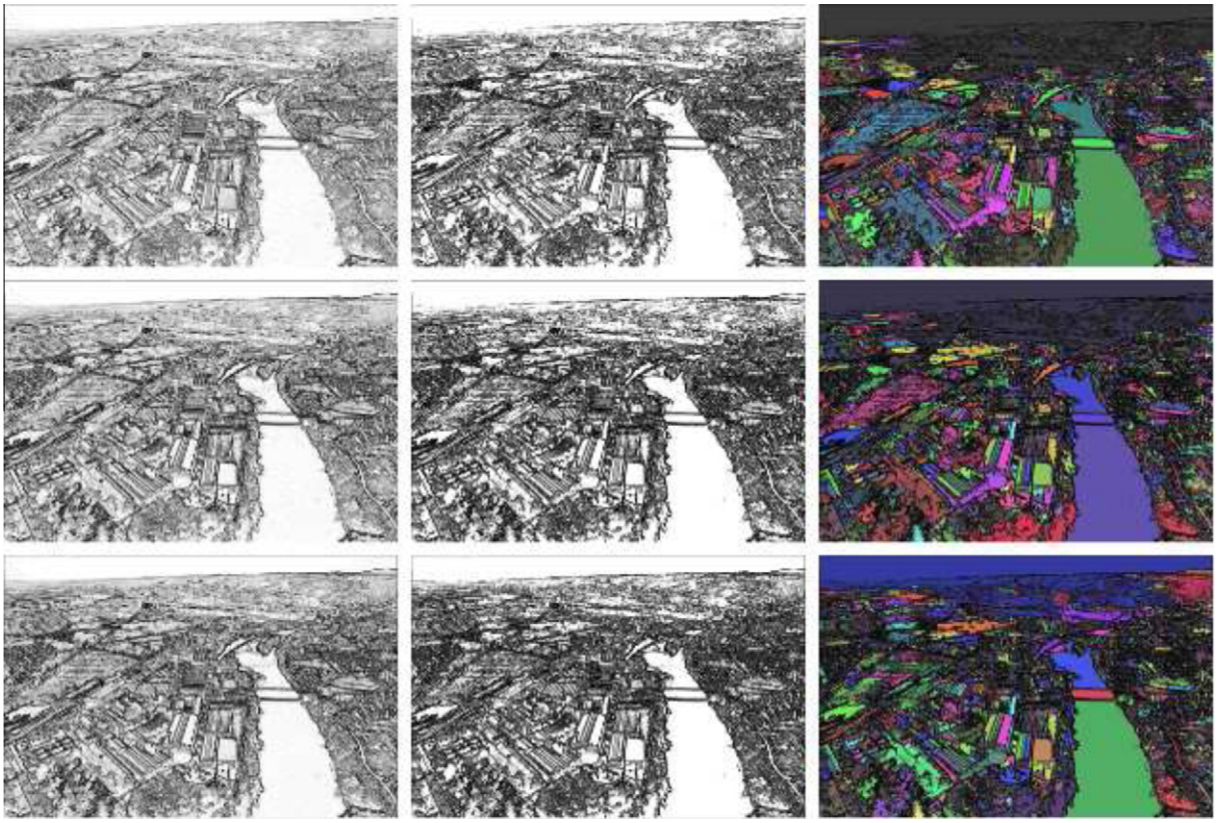

Fig. 4. Second comparative example between 3×3 Sobel, 5×5 Sobel and AGP operators.

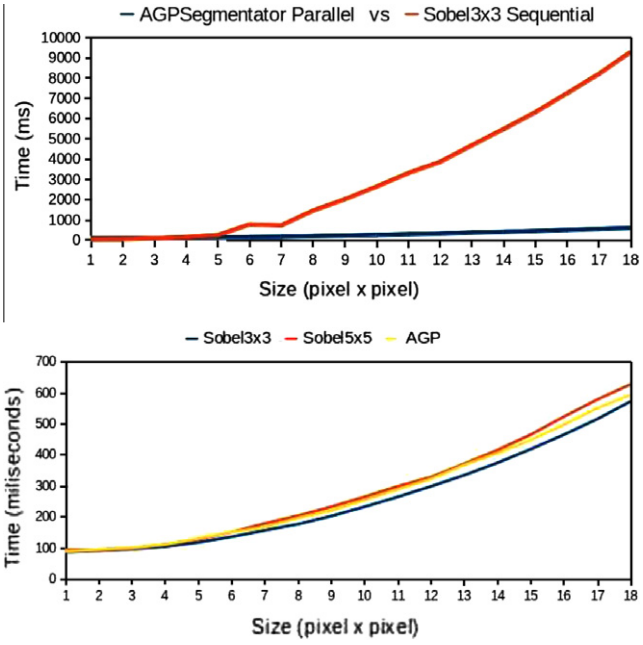

Fig. 5. A time complexity of 3×3 Sobel, 3×5 Sobel and AGP segmentator algorithms.

Sobel and AGP. We can check our operator is more efficient than 5×5 Sobel and very close to 3×3 Sobel.

From a computational point of view, it is clear that for small images, there is no advantage using CUDA™, but in the way the image increases, the time cost in sequential algorithm (in our case 3×3 Sobel) increases rapidly, but a parallel implementation (AGP in this case) remains with little time computational cost.

Table 2
Comparison among our implementations and (Khalid et al., 2011b).

	$1K \times 1K$	$2K \times 2K$	$3K \times 3K$
DUO CORE Sequential	3.04	11.56	25.26
QUAD CORE Sequential	3.39	12.92	28.12
DUO CORE (best 2 threads)	1.75	6.125	13
QUAD CORE (best 8 threads)	1	3.125	6.125
Sobel 3×3 Sequential	0.05	0.152	0.77
Sobel 5×5 Sequential	0.076	0.267	1.215
Sobel 3×3 CUDA	0.092	0.105	0.137
Sobel 5×5 CUDA	0.094	0.112	0.152
AGP CUDA	0.096	0.112	0.152

Finally, we compare our methods with some implementations found in the literature. The reference has been (Khalid et al., 2011b). In this paper, the authors present the results for images whose size is $1K \times 1K, 2K \times 2K$ or $3K \times 3K$ pixels.

Table 2 shows the execution time of the methods with the previous sizes. For the four first rows the columns present the best time obtained from 10 examples for images of each size. In this case, the times were obtained by performing the experiments on two different computers (DUO Core and QUAD Core) for sequential and parallel implementations.

The five last rows show the times obtained with our implementations of the Sobel algorithm in the 3×3 and 5×5 versions for sequential and parallel implementations. We also provide the time obtained for the AGP segmentator.

5. Final remarks

Classical sequential algorithms need to be revisited and adapted to the novel technologies, but the new developments also need the support of deep theoretical foundations.

The bio-inspired computing techniques have features as the encapsulation of the information, a simple representation of the knowledge and parallelism, which are appropriate in dealing with digital images. In this paper, we have shown an example of how a combination of parallel bio-inspired algorithms together with a parallel implementation can improve classical techniques for dealing with digital images.

From the Computer Vision side, the use of bio-inspired techniques opens many possibilities for a deep revision of classical algorithms. The new architectures have shown to be powerful tools for a real implementation of the intrinsic parallelism of the processes of Nature.

From the Natural Computing side, the application of its techniques to a new class of problems opens a new research line by exploring new data structures and bio-inspired algorithms to solve digital image problems in a more efficient way.

Acknowledgements

DDP and MAGN acknowledge the support of the projects TIN2008-04487-E and TIN-2009-13192 of the Ministerio de Ciencia e Innovación of Spain and the support of the Project of Excellence with *Investigador de Reconocida Valía* of the Junta de Andalucía, grant P08-TIC-04200. AB acknowledges the support of the project MTM2009–12716 of the Ministerio de Educación y Ciencia of Spain and the project PO6-TIC-02268 of Excellence of Junta de Andalucía, the "CATAM" PAICYT research group FQM-296 and EHU09/04 project of University of the Basque Country.

References

Abelson, H., Allen, D., Coore, D., Hanson, C., Homsy Jr., G., Knight, T.F., Nagpal, R., Rauch, E., Sussman, G.J., Weiss, R., 2000. Amorphous computing. Comm. ACM 82, 74.

Adleman, L.M., 1994. Molecular computation of solutions to combinatorial problems. Science 266, 1021–1024.

Alberts, B., Johnson, A., Lewis, J., Raff, M., Roberts, K., Walter, P., 2002. Molecular Biology of the Cell, fourth ed. Garland Science, London, UK.

Borrego-Ropero, R., Díaz-Pernil, D., Pérez-Jiménez, M.J., 2007. Tissue simulator: A graphical tool for tissue P systems, in: Vaszil, G. (Ed.), Proceedings of the International Workshop Automata for Cellular and Molecular Computing, MTA SZTAKI, Budapest, Hungary, pp. 23–34. Satellite of the 16th International Symposium on Fundamentals of Computational Theory.

Cecilia, J.M., García, J.M., Guerrero, G.D., Martínez-del-Amor, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J., 2010. Simulation of P systems with active membranes on CUDA. Briefings Bioinf. 11, 313–322.

Ceterchi, R., Gramatovici, R., Jonoska, N., Subramanian, K.G., 2003a. Tissue-like P systems with active membranes for picture generation. Fundam. Inf. 56, 311–328.

Ceterchi, R., Mutyam, M., Păun, G., Subramanian, K.G., 2003b. Array-rewriting P systems. Nat. Comput. 2, 229–249.

Christinal, H.A., Díaz-Pernil, D., Real, P., 2009. Segmentation in 2D and 3D image using tissue-like P system. In: Bayro-Corrochano, E., Eklundh, J.O. (Eds.), Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications. Proc. 14th Iberoamerican Conf. on Pattern Recognition, CIARP 2009, Guadalajara, Jalisco, Mexico, November 15–18. Springer, Berlin, Heidelberg, pp. 169–176.

Christinal, H.A., Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., 2010a. Thresholding of 2D images with cell-like P systems. Rom. J. Inf. Sci. Technol. (ROMJIST) 13, 131–140.

Christinal, H.A., Díaz-Pernil, D., Real, P., 2010b. P systems and computational algebraic topology. J. Math. Comput. Model. 52, 1982–1996, The BIC-TA 2009 Special Issue, Internat. Conf. on Bio-Inspired Computing: Theory and Applications.

Christinal, H.A., Díaz-Pernil, D., Real, P., 2011. Region-based segmentation of 2D and 3D images with tissue-like P systems. Pattern Recognition Lett. 32, 2206–2212. Advances in Theory and Applications of Pattern Recognition, Image Processing and Computer Vision.

Davies, E., 2012. Computer and Machine Vision: Theory, Algorithms, Practicalities. Elsevier Science.

Davis, L.S., 1975. A survey of edge detection techniques. Comput. Graphics Image Process. 4, 248–270.

de Castro, L.N., 2007. Fundamentals of natural computing: An overview. Phys. Life Rev. 4, 1–36.

de Castro, L.N., Timmis, J., 2002. Artificial Immune Systems: A New Computational Intelligence Approach. Springer.

Dersanambika, K.S., Krithivasan, K., 2004. Contextual array P systems. Internat. J. Comput. Math. 81, 955–969.

Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A., 2007. Solving subset sum in linear time by using tissue P systems with cell division. In: Mira, J., Álvarez, J.R. (Eds.), IWINAC (1). Springer, Berlin, Heidelberg, pp. 170–179.

Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A., 2008a. A uniform family of tissue P systems with cell division solving 3-COL in a linear time. Theor. Comput. Sci. 404, 76–87.

Díaz-Pernil, D., Pérez-Hurtado, I., Pérez-Jiménez, M.J., Riscos-Núñez, A., 2008b. A P-lingua programming environment for membrane computing. In: Corne, D.W., Frisco, P., Păun, G., Rozenberg, G., Salomaa, A. (Eds.), Workshop on Membrane Computing. Springer, Berlin, Heidelberg, pp. 187–203.

Díaz-Pernil, D., Graciani, C., Gutiérrez-Naranjo, M.A., Pérez-Hurtado, I., Pérez-Jiménez, M.J., 2010a. Software for P systems. In: HandbookMC10, pp. 437–454.

Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Molina-Abril, H., Real, P., 2010b. A bio-inspired software for segmenting digital images. In: Nagar, A.K., Thamburaj, R., Li, K., Tang, Z., Li, R. (Eds.), Proc. 2010 IEEE Fifth Internat. Conf. on Bio-Inspired Computing: Theories and Applications BIC-TA. IEEE Computer Society, Beijing, China, pp. 1377–1381.

Díaz-Pernil, D., Gutiérrez-Naranjo, M.A., Molina-Abril, H., Real, P., 2011. Designing a new software tool for digital imagery based on P systems. Nat. Comput., 1–6. http://dx.doi.org/10.1007/s11047-011-9287-4.

Egmont-Petersen, M., de Ridder, D., Handels, H., 2002. Image processing with neural networks – a review. Pattern Recognition 35, 2279–2301.

Engelbrecht, A.P., 2005. Fundamentals of Computational Swarm Intelligence. Wiley and Sons.

Gimel'farb, G., Nicolescu, R., Ragavan, S., 2011. P systems in stereo matching. In: Real, P., Díaz-Pernil, D., Molina-Abril, H., Berciano, A., Kropatsch, W. (Eds.), Computer Analysis of Images and Patterns, Lecture Notes in Computer Science, vol. 6855. Springer, Berlin/Heidelberg, pp. 285–292.

Gutiérrez-Naranjo, M.A., Pérez-Jiménez, M.J., Riscos-Núñez, A., 2006. Available membrane computing software. In: Ciobanu, G., Pérez-Jiménez, M.J., Păun, G. (Eds.), Applications of Membrane Computing, Natural Computing Series, vol. 6855. Springer, pp. 411–436.

Hirvensalo, M., 2004. Quantum computing. Natural Computing Series. Springer.

Holland, J.H., 1992. Adaptation in Natural and Artificial Systems. MIT Press, Cambridge, MA, USA.

Kari, L., Rozenberg, G., 2008. The many facets of natural computing. Comm. ACM 51, 72–83.

Khalid, N.E.A., Ahmad, S.A., Noor, N.M., Fadzil, A.F.A., Taib, M.N., 2011a. Analysis of parallel multicore performance on sobel edge detector. In: Proc. 15th WSEAS Internat. Conf. on Computers. World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, USA.

Khalid, N.E.A., Ahmad, S.A., Noor, N.M., Fadzil, A.F.A., Taib, M.N., 2011b. Parallel approach of Sobel edge detector on multicore platform. Internat. J. Comput. Comm. 5, 236–244.

Krishna, S.N., Rama, R., Krithivasan, K., 2001. P systems with picture objects. Acta Cybernet. 15, 53–74.

Martín-Vide, C., Păun, G., Pazos, J., Rodríguez-Patón, A., 2003. Tissue P systems. Theor. Comput. Sci. 296, 295–326.

McCulloch, W.S., Pitts, W., 1943. A logical calculus of the ideas immanent in nervous activity. Bull. Math. Biophys. 5, 115–133.

Nguyen, V., Kearney, D., Gioiosa, G., 2010. An extensible, maintainable and elegant approach to hardware source code generation in reconfig-P. J. Logic Algebraic Program. 79, 383–396.

Nickolls, J., Buck, I., Garland, M., Skadron, K., 2008. Scalable parallel programming with CUDA. Queue 6, 40–53.

Ogawa, K., Ito, Y., Nakano, K., 2010. Efficient canny edge detection using a gpu. In: Proc. 2010 First Internat. Conf. on Networking and Computing. IEEE Computer Society, Washington, DC, USA.

Owens, J.D., Houston, M., Luebke, D., Green, S., Stone, J.E., Phillips, J.C., 2008. GPU computing. Proc. IEEE 96, 879–899.

Pal, N.R., Pal, S.K., 1993. A review on image segmentation techniques. Pattern Recognition 26, 1277–1294.

Parker, J., 2010. Algorithms for Image Processing and Computer Vision. John Wiley & Sons.

Peli, T., Malah, D., 1982. A study of edge detection algorithms. Comput. Graphics Image Process. 20, 1–21.

Peña-Cantillana, F., Díaz-Pernil, D., Berciano, A., Gutiérrez-Naranjo, M.A., 2011a. A parallel implementation of the thresholding problem by using tissue-like P systems. In: Real, P., Díaz-Pernil, D., Molina-Abril, H., Berciano, A., Kropatsch, W.G. (Eds.), CAIP (2). Springer, pp. 277–284.

Peña-Cantillana, F., Díaz-Pernil, D., Christinal, H.A., Gutiérrez-Naranjo, M.A., 2011b. Implementation on CUDA of the smoothing problem with tissue-like P systems. Internat. J. Natural Comput. Res. 2, 25–34.

Păun, G., 2000. Computing with membranes. J. Comput. Systems Sci. 61, 108–143.

Păun, G., 2002. Membrane Computing. An Introduction. Springer-Verlag, Berlin, Germany.

Păun, G., Pérez-Jiménez, M.J., Riscos-Núñez, A., 2008. Tissue P systems with cell division. Internat. J. Comput. Comm. Control 3, 295–303.

Păun, G., Rozenberg, G., Salomaa, A. (Eds.), 2010. The Oxford Handbook of Membrane Computing. Oxford University Press, Oxford, England.

Rosin, P.L., 2006. Training cellular automata for image processing. IEEE Trans. Image Process. 15, 2076–2087.

Sanduja, V., Patial, R., 2012. Sobel edge detection using parallel architecture based on FPGA. Internat. J. Appl. Inf. Systems 3, 20–24, Published by Foundation of Computer Science, New York, USA.
Selvapeter, P.J., Hordijk, W., 2009. Cellular automata for image noise filtering. In: NaBIC. IEEE, pp. 193–197.
Sobel, I.E., 1970. Camera models and machine perception. Ph.D. thesis. Dept. of Computer Sciences. Stanford, CA, USA. AAI7102831.
von Neumann, J., 1966. Theory of Self-Reproducing Automata. University of Illinois Press, Champaign, IL, USA.