Image Segmentation Inspired by Cellular Models using hardware programming

Javier Carnero, Daniel Díaz-Pernil, Helena Molina-Abril, Pedro Real

University of Sevilla, Department of Applied Mathematics, Sevilla, Spain.
javier@carnero.net, sbdani@us.es, habril@us.es, real@us.es

Abstract Several features of image segmentation make it suitable for bio–inspired techniques. It can be parallelized, locally solved and the input data can be easily encoded using representations inspired by nature. In this paper, we present a new hardware system that follows the Membrane Computing approach, and performs edge–based segmentation, noise removal and thresholding of digital images.

Keyworks: Image Segmentation, Computational Algebraic Topology, Membrane Computing, Tissue-like P Systems, FPGA.

1 Introduction

Natural Computing studies computational paradigms inspired from various wellknown natural phenomena in physics, chemistry and biology [10]. All these computational paradigms have in common an alternative way of encoding the information, adapted to the bio–inspired substrate, and the use of intrinsic parallelism of natural processes.

Within this wide field of bio–inspired models, *Membrane Computing* [13, 14] are theoretical models of computation based on the structure and functioning of cells as living organisms that are able to process and generate information. The computational devices in Membrane Computing are called *P systems*. Roughly speaking, a P system consists of a membrane structure, in the compartments of which multi sets of objects are placed. These multi sets evolve according to given rules. In the most extended model, the rules are applied in a synchronous non-deterministic maximally parallel manner, but some other semantics are being explored.

Several attempts for bridging problems from Algebraic Topology with Natural Computing can be found in the literature ([3, 2, 9]). Nevertheless, the advantages of Membrane Computing have not been yet exploited within the digital topology field. Only sequential tools have been developed (see [5]) in the best case.

In this paper, we design a parallel hardware system based on a membrane computing model for implementing a segmentation algorithm. Segmentation in computer vision (see [16]), refers to the process of partitioning a digital image into multiple regions (sets of pixels). This problem has several features that make it suitable for techniques inspired by nature. One of them is that it 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 it. Another interesting feature is that the basic necessary information can be easily encoded by bio–inspired representations.

Previous versions of the tissue–like P system segmentation model have been published in [4]. The system presented here is more evolved that the ones introduced in those preliminary works, where the noise removal was not included, and the implementation was done using a sequential software that could not take advantage of the parallel nature of the theoretical model.

The hardware tool we propose here consists of a *Field-Programmable Gate Array* unit (*FPGA*). FPGAs are prefabricated silicon devices that can be electrically programmed to become almost any kind of digital circuit or system. They have many advantages over *Application Specific Integrated*

Circuits (*ASIC*). Developing an ASIC takes very much time and is expensive. Furthermore, it is not possible to correct errors after fabrication. In contrast to ASICs, FPGAs are configured after fabrication and they also can be reconfigured. This is done with a hardware description language (HDL) which is compiled to a bit stream and downloaded to the FPGA ([1]). FPGAs contain programable logic components called *logic blocks*, and a hierarchy of reconfigurable interconnects that allow the blocks to be *wired together*. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. The FPGA used here also includes memory elements in the logic blocks.

Although we focus here in a problem that can be considered rather simple from the image analysis point of view, this work provides a first promising step towards the usefulness of membrane computing within the image processing field.

With the ad–hoc system we propose, the execution of the segmentation process consumes always the same time, independently of the image size. The hardware design is unique and valid for dealing with higher dimensional images.

The paper is organized as follows: Firstly, the bio–inspired formal framework of the work is defined. Next, we introduce the family of tissue–like P systems that has been developed to obtain image segmentation. In Section 4 the hardware system that implements the membrane structure is shown. The paper finished with some conclusions and future work.

2 Formal Framework

In the initial definition of the cell-like model of P systems [13], membranes are hierarchically arranged in a tree–like structure. Its biological inspiration comes from the morphology of cells, where small vesicles are surrounded by larger ones. This biological structure can be abstracted into a tree–like graph, where the root represents the skin of the cell (i.e., the outermost membrane) and the leaves represent membranes that do not contain any other membrane.

In *tissue P systems*, the tree-like membrane structure is replaced by a general graph. This model has two biological inspirations (see [12, 11]): intercellular communication and cooperation between neurons. The common mathematical model of these two mechanisms is a net of processors dealing with symbols and communicating these symbols along channels specified in advance. The communication among cells is based on symport/antiport rules. In symport rules, objects cooperate to traverse a membrane together in the same direction, whereas in the case of antiport rules, objects residing at both sides of the membrane cross it simultaneously but in opposite directions.

Formally, a *tissue-like P system* of degree $q \geq 1$ with input is a tuple of the form

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

where

1. Γ is a finite *alphabet*, whose symbols will be called *objects*,
2. $\Sigma(\subset \Gamma)$ is the input alphabet,
3. $\mathcal{E} \subseteq \Gamma$ (the objects in the environment),
4. $w_1, \ldots, w_q$ are strings over Γ representing the multisets of objects associated with the cells at the initial configuration,
5. $\mathcal{R}$ is a finite set of communication rules of the following form: $(i, u/v, j)$, for $i, j \in \{0, 1, 2, \ldots, q\}, i \neq j$, $u, v \in \Gamma^*$,
6. $i_\Pi, o_\Pi \in \{0, 1, 2, \ldots, q\}$,

A tissue-like P system of degree $q \geq 1$ can be seen as a set of q cells labeled by $1, 2, \ldots, q$. We will use 0 to refer to the label of the environment, i_Π and o_Π denote the input region and the output region (which can be the region inside a cell or the environment) respectively.

The strings $w_1, \ldots, w_q$ describe the multisets of objects 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 available in an arbitrary large amount of copies.

The communication rule $(i, u/v, j)$ can be applied over two cells labeled by 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 if either $i = 0$ or $j = 0$ then the objects are interchanged between a cell and the environment.

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 membrane can only be used for one rule (non-deterministically chosen when there are several possibilities), but any object than can participate in a rule of any form must do it, i.e., at each step a maximal set of rules is applied.

3 Segmenting Digital Images

In this section, a family of P systems is defined to obtain an edge–segmentation of a 2D digital image. First of all, the system applies a basic noise filter in order to eliminate some pickle noise that could affect the segmentation process. Next, the system performs a thresholding of the image to solve the problem of degradation of colours of pixels in the boundary of adjacent regions with different colours. Once this process is finished, the system applies the rules defined in [4] obtaining the image segmentation.

In order to remove noise, the system will apply the largely used average filter because of its simplicity and good results. For each pixel, the process consists of calculating the colour average of its adjacent pixels. If the distance between the pixel and its average is bigger than a threshold t, the pixel will be considered as noise and it will be replaced by its colour average.

Thresholding is a method of image segmentation whose basic aim is to obtain a binary image from a colour one. The idea is to split the set of pixels into two sets (black and white) depending on its bright and a fixed valued, the *threshold*. If the bright of the pixel is greater than the threshold, then the pixel is labelled as *object*. Otherwise, it is labelled as *background*. After labelling, a new binary image is created by colouring each pixel white or black, depending on the label.

The basic thresholding method can be generalized in a natural way. Instead of getting a binary image by labelling the original set of pixels by $\{0, 1\}$, we can consider a larger set of labels, $\{1, \ldots, k\}$ so we obtain a final image with k levels. Another natural generalization is to replace the colour information by another scale on the features of the pixel (bright, intensity, gray scale, etc.).

The digital image is subdivided in multiple pixels forming a network of points of $\mathbb{N}^2$. Let $\mathcal{C} \subseteq \mathbb{N}$ be the set of all colours in the given image in a certain order. $|\mathcal{C}|$ is defined as the number of colours in $\mathcal{C}$. We will assume that each pixel is associated to a colour of the image. Then, we encode the pixel (i, j) with associated colour $a \in \mathcal{C}$ with the object a_{ij} of the system.

Concerning the adjacency between pixels, the 4–neighborhood relation between them is considered [15]. From a membrane computing point of view it is easier to work with 8-adjacency.

Given a $2D$ digital image, for each $n \in \mathbb{N}$, we consider the tissue-like P system

$$\Pi = (\Gamma, \Sigma, \mathcal{E}, w_1, w_2, \mathcal{R}, i_\Pi, o_\Pi),$$

defined as follows:

(a) $\Gamma = \Sigma \cup \{a'_{ij}, a''_{ij}, \bar{a}_{ij}, A_{ij} : 1 \leq i, j \leq n, a \in \mathcal{C}\}$,

(b) $\Sigma = \{a_{ij} : 1 \leq i, j \leq n, a \in \mathcal{C}\}$,

(c) $\mathcal{E} = \Gamma - \Sigma$,

(d) $w_1 = t_1^{n^2}$, $w_2 = \emptyset$,

(e) R is the following set of communication rules:

1. $(1, t_i/t_{i+1}, 0)$ with $i = 1, \ldots, 18$.
   These rules are used as counter to know when system can begin the thresholding phase.
2. $(1, a_{ij}, b_{ij-1}, c_{i-1j-1}, d_{i-1j}, e_{i-1j+1}, f_{ij+1}, g_{i+1j+1}, h_{i+1j}, i_{i+1j-1}/$
   $av(a)_{ij}, b_{ij-1}, c_{i-1j-1}, d_{i-1j}, e_{i-1j+1}, f_{ij+1}, g_{i+1j+1}, h_{i+1j}, i_{i+1j-1}, 0)$
   with
   - $1 \leq i, j \leq n$,
   - $a, b, c, d, e, f, g, h, i \in \mathcal{C}$,
   - $av(a) = (b + c + d + e + f + g + h + i)/8$ and
   - $|a - av(a)| > t$.

   These set of rules are used to detect the noise and correct it with the colour average of its adjacent pixels.
3. $(1, t_{19}\, a_{ij}/a'_{ij}, 0)$
   with
   - $a \in \mathcal{C}$ and
   - $1 \leq i, j, k, l \leq n$.

   These rules are used to trade objects a_{ij} from cell 1 against objects a'_{ij} from the environment. So, we can do the thresholding in the next step.
4. $(1, b'_{ij}/a''_{ij}, 0)$
   with
   - $1 \leq i, j \leq n$,
   - $m = (|\mathcal{C}|/k)$,
   - $a, b \in \mathcal{C}$, $a < b \leq a + (m - 1)$, $a = m \cdot l$ and
   - $l = 0, 1, 2, \ldots, k$.

   These rules are used to discretize the colors dividing the set of colors in k intervals of length m.

5. $(1, a''_{ij} b''_{kl}/\bar{a}_{ij} A_{ij} b''_{kl}, 0)$,
   with
   - $a, b \in \mathcal{C}, a < b$,
   - $1 \leq i, j, k, l \leq n$.

   These rules are used when image has two adjacent pixels with different associated colors (border pixels). Then, the pixel with less associated color is marked and the system brings from the environment an object representing this marked pixel (edge pixel).

4. $(1, \bar{a}_{ij} a''_{ij+1} \bar{a}_{i+1j+1} b''_{i+1j} \,/\, \bar{a}_{ij} \bar{a}_{ij+1} A_{ij+1} \bar{a}_{i+1j+1} b''_{i+1j}, 0)$
   with
   - $a, b \in \mathcal{C}, a < b$ and
   - $1 \leq i, j \leq n - 1$.

   $(1, \bar{a}_{ij} a''_{i-1j} \bar{a}_{i-1j+1} b''_{ij+1} \,/\, \bar{a}_{ij} \bar{a}_{i-1j} A_{i-1j} \bar{a}_{i-1j+1} b''_{ij+1}, 0)$
   with
   - $a, b \in \mathcal{C}, a < b$ and
   - $2 \leq i \leq n,\ 1 \leq j \leq n - 1$.

   $(1, \bar{a}_{ij} a''_{ij+1} \bar{a}_{i-1j+1} b''_{i-1j} \,/\, \bar{a}_{ij} \bar{a}_{ij+1} A_{ij+1} \bar{a}_{i-1j+1} b''_{i-1j}, 0)$
   with

* $a, b \in \mathcal{C}, a < b$ and
* $2 \leq i \leq n,\ 1 \leq j \leq n-1$.

$(1, \bar{a}_{ij} a''_{i+1j} \bar{a}_{i+1j+1} b''_{ij+1} \,/\, \bar{a}_{ij} \bar{a}_{i+1j} A_{i+1j} \bar{a}_{i+1j+1} b''_{ij+1}, 0)$
with

* $a, b \in \mathcal{C}, a < b$ and
* $1 \leq i, j \leq n-1$.

These rules mark with a bar the pixels that are adjacent to two pixels with the same color that were marked before, but with the condition that the marked objects are adjacent to an other pixel with a different color. An edge object representing the last marked pixel is brought from the environment.

5. $(1, A_{ij}/\lambda, 2)$, with $1 \leq i, j \leq n$.
   This rule is used to send the edge pixels to the output cell.

(f) $i_\Pi = 1$, $o_\Pi = 2$.

The input data of the system is given by the set $\{a_{ij} : a' \in \mathcal{C}, 1 \leq i, j \leq n\}$, codifying the pixels of the image.

4 A Hardware Tool

A hardware system implementing the tissue-like P system described above is shown in Figure 1. The system consists of processing units capable to deal with 4×4 images. These units can be combined like a puzzle in order to process $n \times m$ images.

A 4×4 section of the initial image is passed trough the I/O port of each processing unit. The resulting image is also read from this port. The *Border Pixels* and *Adjacent Pixels* ports are used to interconnect processing units in order to deal with bigger size images.

The t and k ports specify the maximum distance between the pixel and its average (noise filter), and the number of different levels for the thresholding respectively.

Looking inside the processing unit, at first sight two *image units* can be distinguished. The purpose of these units is to store the original elements of the image and the resulting ones after applying the system rules (they represent the multi–set of the cell). Although the process could be performed using only one image unit, one of them will be used to read/write the image, while the other one is processing. Therefore, the system does not waste any time in writing the new image and is able to process and read the resulting one (the system is always working).

The interconnection circuit is responsible for connecting each pixel and its adjacent ones to its *pixel processing unit*, and connect the resulting object with the image units. The pixel processing unit sends this information to the four *rule units* (NoiseFiltering, Thresholding and Segmentation one and two), collects the received information and sends the resulting object to the interconnection circuit. Therefore, four rule units are needed for each pixel.

Each rule unit represents a set of theoretical rules that can not be executed at the same time for the same pixel. For example, for a pixel in the noise filtering problem, at most only one rule will be applied to this pixel in order to change the pixel by its average, so for that pixel, all the rules according with the noise filtering can be grouped in one concurrent computation unit, the *noise filtering unit*.

The implementation of this hardware tool allows the system to apply the maximum number of rules at each moment, using only four rule units for pixel to solve the whole problem. Therefore, the system works exactly like the theoretical model in terms of complexity, time, concurrency and results.

The *processing unit controller* controls all the system, in order to make the different parts work properly.

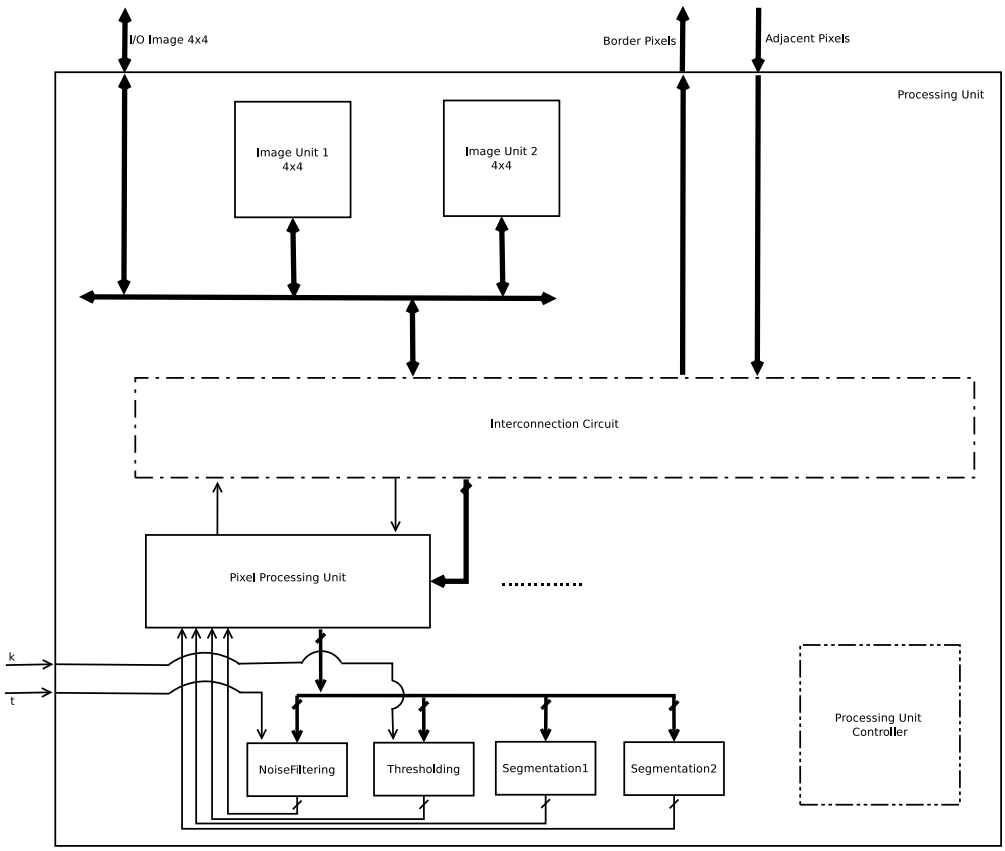

Figure 1: Hardware system

4.1 Implementation

As mentioned above, a FPGA has been used to implement the described system. More precisely, the design is based on the XC6SLX45T-FGG484-3CES FPGA chip of Xilinx. This chip has been chosen due to its quality–cost relation.

The implementation of the system into a FPGA requires four steps: Description, synthesis, implementation and programming.

Firstly, the system is described by a hardware description language. For this purpose, we have used VHD (VHSIC hardware description language). VHD is a well known language that has been widely used in electronic design automation to describe digital (our case) and mixed-signal systems. Due to the fact that we work with a parallel code, the usual programming paradigms are not suitable here. Working with electrical elements forces us to deal with signal fluctuations, tri-state buffers, etc.

Once the system has been described, we are able to perform functional simulations (Figure 2).

The second and third steps are needed in order to translate the described system into a logic system (in terms of logic doors, NAND, XOR, etc..) and to glue them together conforming a unique file respectively.

Once the synthesis and implementation are complete, different temporal simulations (more accurate than the functional one) can be performed.

Finally, the file is compiled to be programmed in the FPGA chip. Because of the inexactness of the simulations, some final adjustments need to be done in order to make the real system work.

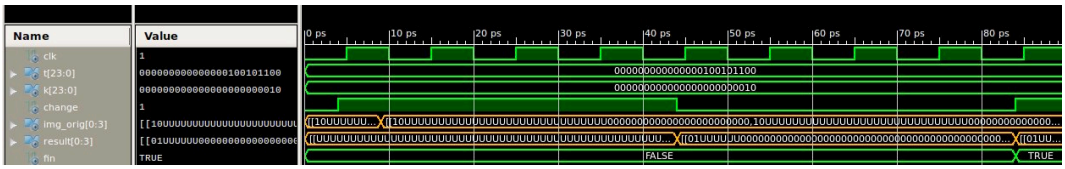

Figure 2: Simulation

Several software tools (called IDEs (Integrated Development Environment)) are available to control the implementation process. We have used the *Xilinx ISE Webpack tool* and the Xilinx *ISim simulation tool* to simulate the behavior of the system.

With this implementation, the system is able to process any image of size $n \times m$ using at most four clock cycles.

In Figure 2 a simple simulation example is shown. In this example two images are segmented within eight clock cycles. The first image consists of a white triangle at the top–right corner of the image. The second one has white pixels at the left side, and black pixels at the right side (`img_orig(0:3)` signal). The signal `change` is used to inform the system that the result has been obtained (`result` singal). Although the result is ready after the fourth clock cycle (40^{th} picoseconds in the simulation), it is not received until the 44^{th} picosecond. The reason is that the simulation, signal `change` changes at that point. That does not change the fact that the system starts processing the new image at the 45^{th} picosecond. Therefore, the second image is resolved at the 80^{th} picosecond (again 4 clock cycles) and then, the result is obtained (when the `change` signalis activated) at the 88^{th} picosecond.

5 Final Remarks

In this paper, we present a new tool to obtain a segmentation of digital images. The system follows a membrane computing approach, and hardware programming (VHDL) is used to develop an ad–hoc processor to solve the problem.

This is a first step in a new research line where very different areas like Computational Algebraic Topology, Image Processing, Membrane Computing and Hardware Programming are being mixed to generate new useful tools.

In the next future, we plan to follow two paths. Firstly, the development of the system in order to deal with more complicated image processing problems. Moreover, it is a compulsory task to compare our system with some of the existing hardware segmentation methods ([8, 6, 7]).

On the other hand, we plan to develop new tools to compute homological information on digital images (homology groups, spanning trees, homology gradient vector field, etc.). Until now, this homological information has been typically obtained using sequential algorithms or, in the best case, partially parallel algorithms. We believe that the use of FPGA and P systems can help in the parallelization of some of the existing methods.

Acknowledgements

DDP 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. JC, HMA and PR acknowledge the support of the project MTM2009-12716 of the Ministerio español de Educación y Ciencia, the project PO6-TIC-02268 of Excellence of Junta de Andalucía, and the *Computational Topology and Applied Mathematics* PAICYT research group FQM-296.

References

[1] C. Bumann. Field programmable gate array (fpga). *Summary paper for the seminar "Embedded System Architecture"*, January 2010.

[2] R. Ceterchi, R. Gramatovici, N. Jonoska, and K.G. Subramanian. Tissue-like p systems with active membranes for picture generation. *Fundam. Inform.*, 56(4):311–328, 2003.

[3] R. Ceterchi, M. Mutyam, G. Păun, and K. G. Subramanian. Array-rewriting p systems. *Natural Computing: an international journal*, 2(3):229–249, 2003.

[4] Hepzibah A. Christinal, Daniel Díaz-Pernil, and Pedro Real Jurado. Segmentation in 2d and 3d image using tissue-like p system. In *CIARP '09: Proceedings of the 14th Iberoamerican Conference on Pattern Recognition*, pages 169–176, Berlin, Heidelberg, 2009. Springer-Verlag.

[5] D. Díaz-Pernil, M.A. Gutiérrez-Naranjo, H. Molina-Abril, and P. Real. A bio-inspired software for segmenting digital images. *Proceedings of the IEEE Fifth International conference on Bio–Inspired Computing: Theories and Applications*, pages 1377–1381, 2010.

[6] P. Dillinger, J.F. Vogelbruch, J. Leinen, S. Suslov, R. Patzak, H. Winkler, and K. Schwan. Fpga-based real-time image segmentation for medical systems and data processing. *IEEE Transactions on Nuclear Science*, 53(4):2097–2101, 2006.

[7] B. Girau and C. Torres-Huitzil. Fpga implementation of an integrate-and-fire legion model for image segmentation. *ESANN'2006 proceedings*, pages 1–10, 2006.

[8] S. Herrmann, H. Mooshofer, and W. Stechele. An architecture concept for hardware accelerated image segmentation. *Institute for Integrated Circuits, Technical University of Munich.*

[9] Chao. J. and Nakayama. J. Cubical singular simples model for 3d objects and fast computation of homology groups. *Proceedings of ICPR'96 IEEE*, pages 190–194, 1996.

[10] L. Kari and G. Rozenberg. The many facets of natural computing. *Communications of the ACM*, 51(10):72–83, 2008.

[11] C. Martín-Vide, Gh. Paun, J. Pazos, and A. Rodríguez-Patón. Tissue p systems. *Theoretical Computer Science*, 296(2):295–326, 2003.

[12] C. Martin-Vide, J. Pazos, Gh. Paun, and A. Rodríguez-Patón. A new class of symbolic abstract neural nets: Tissue p systems. In *COCOON '02: Proceedings of the 8th Annual International Conference on Computing and Combinatorics*, pages 290–299, London, UK, 2002. Springer-Verlag.

[13] Gheorghe Păun. Computing with membranes. *J. Comput. Syst. Sci.*, 61(1):108–143, 2000.

[14] Gheorghe Păun. *Membrane Computing:An Introduction.* 2002.

[15] Azriel Rosenfeld. Digital topology. *American Mathematical Monthly*, 86:621–630, 1979.

[16] G. Stockman and L. G. Shapiro. *Computer Vision.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.