A Salient Region Detector for GPU Using a Cellular Automata Architecture

David Huw Jones, Adam Powell,
Christos-Savvas Bouganis, and Peter Y.K. Cheung

Imperial college London, Electrical and Electronic Engineering
Princes Gardens, London, SW7 2AZ

Abstract. The human visual cortex performs salient region detection, a process critical to the rapid understanding of a scene. This is performed on large arrays of locally interacting neurons that are slow to simulate sequentially. In this paper we describe and evaluate a novel, bio-inspired, cellular automata (CA) architecture for the determination of the salient regions within a scene. This parallel processing architecture is appropriate for implementation on a graphics processing unit (GPU). We compare the performance of this algorithm against that of CPU implemented salient region detectors. The CA algorithm is less subject to variation due to changing scale, viewpoint and illumination conditions. Also due to its GPU implementation, this algorithm is able to detect salient regions faster than the CPU implemented algorithms.

Keywords: Cellular automata, Graphic Processing Units, Evolution, Saliency, Low-level vision.

1 Introduction

The human vision system incorporates fast image processing algorithms that are a function of low-level, local interactions between large quantities of parallel processing neurons. These include orientation, frequency and color filters, edge, as well as motion and salient region detection. We have had little success imitating their function to a similar scale or speed on sequential systems[1]. New parallel processing platforms such as the graphic processing unit (GPU) have more comparable architectures to the biological visual cortex that may allow us to simulate its function faster and on larger scales. However designing large-scale, locally interacting parallel systems such that they perform some function or display certain complex properties is computationally difficult.

Saliency is the measure of object conspicuity within the scene. This is determined by the V4 region of the primary visual cortex and is used to direct the rapid movements of the eye (saccades). In this paper we will show that a large array of low-level processes can be effectively designed on, and for, a GPU to determine the salient regions of any given scene. We will then compare the speed and transformation invariance of this algorithm to that of three other

K.W. Wong et al. (Eds.): ICONIP 2010, Part II, LNCS 6444, pp. 501–508, 2010.

salient region detectors: the classic Itti and Koch algorithm[2] which uses a combination of color, intensity and orientation filters, feature maps and local centre-surround difference calculations; a directed, weighted graph-based visual saliency (GBVS)[3] approach which uses a Markov chain analysis to detect conspicious nodes; and a recently published algorithm that uses local entropy analysis to determine regions for attention based on information maximization (AIM)[4].

2 Low-Level Architecture for Image Processing

One class of locally interacting arrays of small processors that have been used to simulate biological computation is cellular automata. Cellular automata (CA) are dynamic systems in which space and time are discrete. CA consist of a number of identical cells in an array. Each cell can be in one of a number of states. The next state of each cell is determined at discrete time intervals according to the current state of the cell, the current state of the neighbouring cells and a next-state rule that is identical for each cell.

Von Neumann developed CA to study self-reproducing systems. Since then CA have been extensively used to study or mimic the capabilities of biological systems [5]. CA have also been used to mimic some of the low-level capabilities of biological vision systems. These include edge detection[6] and feature extraction[7].

We propose to use a CA architecture with a set of bio-inspired image-processing filters local to each cell. We will use an evolutionary algorithm to determine how each cell calculates its next state from the output of these filters.

2.1 Proposed CA Architecture

The CA will be a bounded rectangular array that has the same dimensions as the input image. The luminescence of each pixel of the input image is the initial state (at time $t = 0$) of each cell.

Saliency is an output of the V4 region of the primary visual cortex. Between this stage and the optic nerve are the V1 and V2 stages. These perform orientation, gradient and edge detection. We will use similar filters (applied locally to a 10x10 neighbourhood about each cell) as inputs to each cell:

1. Orientation filters. Calculate the minimum,ψ_{min}, maximum, ψ_{max}, and standard deviation, ρ_ψ, of the response of a bank of five Gabor filters defined by the convolution kernel:

$$k(i,j) = e^{-\frac{i^2+j^2}{20}} \cos\left(\frac{2\pi}{5}(i\cos\theta + j\sin\theta)\right) \qquad (1)$$
$$i,j \in [-5,5],\ \theta \in \{0,30,60,90,120\}$$

2. Gradient filters. The mean response of ten gradient responses, $\bigtriangledown(x, y, \theta)$ about each cell $c_{x,y}$.

$$\bigtriangledown(x,y) = \sqrt{(x' c_{x+x',y+y'})^2 + (y' c_{x+x',y+y'})^2} \tag{2}$$
$$x' = 5\cos(\theta) \quad y' = 5\sin(\theta)$$
$$\theta \in \{0, 18, 36, 54, 72, 90, 108, 126, 144, 162\}$$

3. The standard deviation, ρ_c, and mean,$\bar{c}$, state values within the cell neighbourhood.

The scale and coefficients of each of these filters has been chosen for three reasons: to maximise the difference in response to unique and common input sources, to achieve an approximate conservation of sum pixel states across the automata, and to achieve fast processing times. The first iteration ($t = 1$) will be determined by a sum of five of these variables,λ_i (we exclude $\bar{c}$ to improve illumination invariance) weighted by various coefficients, k_i. We are going to use an evolutionary algorithm to choose values for k_i.

$$c_{t=1,x,y} = \sum_i k_i \lambda_i (c_{t=0,x,y}) \tag{3}$$

Further iterations are determined by the previous iteration, the six variables and a decay function α.

$$c_{t+1,x,y} = c_{t,x,y} + \alpha \sum_i k_i \lambda_i (c_{t,x,y}), \ \alpha = e^{-\frac{t}{2T}} \tag{4}$$

Where T is the total number of iterations to be performed by the CA. The decay function ensures the CA converges and the location of any output peaks is at the centre of its corresponding inputs. By treating each cell in the neighbourhood equally we ensure some degree of rotation invariance.

2.2 Implementation

As well as being an appropriate medium on which to mimic biological systems, CA are particularly suited for implementation on graphics processing units (GPU). The GPU architecture allows us to execute many thousands of parallel threads, but each thread must run the same code. This is comparable to a CA, in which each of many thousands of cells run the same program synchronously.

3 The Evolutionary Design of Cellular Automata for Image Processing

As saliency is such a subjective measure there may be one or more effective solutions, making it an interesting task for evolutionary algorithms. We need an evolutionary algorithm appropriate for exploring search spaces with many

possible solutions. Here we describe the fitness function and mutation strategy for the coefficients, k_i, of equation (4).

3.1 Training Set and Fitness Function

To train our CA we create a set of test input grayscale images and use their pixel saturations to set the initial states of each cell within the CA. We then repeatedly iterate the CA, each cell using equations (3), (4) and evolved values for k_i to determine its next state. The final states $c_{t=T,x,y}$ of each cell within the CA form a map of the salient regions in the image.

The test images we use are chosen for their different scale of features and subject matter. For each image, I, in our training set we determine a saliency map from Itti's algorithm. The fitness of the evolved solution is determined as a sum-of-squared differences between the final state of the CA and the saliency map. Note that though this constrains the potential of our algorithm to detect salient regions to that of the performance of Itti's algorithm, we expect our algorithm to outperform it by other metrics, such as speed, translation and illumination invariance.

By loading the training images and their corresponding salient maps on the device, we can test each solution with minimal host-device or device-host data transfer; significantly speeding up the training stages of the evolution algorithm.

3.2 Mutation Algorithm

We use a variant of the HereBoy algorithm [8] for the mutation of this CA because its version of constrained simulated annealing is particularly suited to exploring search spaces with many possible solutions.

The HereBoy algorithm uses a population of two solutions: the best solution so far and a mutation of this solution. In most cases, if the mutated solution performs better than the current solution, the current is replaced with the mutated. Otherwise the mutated solution is discarded and another mutation of the best solution is evaluated.

The HereBoy algorithm first attempts to determine the general structure of the solution with high mutation rates, then tries to refine it with lower mutation rates. Thus the probability of mutating each coefficient k_i is function of predefined limits $(p_{m,min}, p_{m,max})$, the fitness of the solution and the expected maximum fitness, f_{max}:

$$p_m = \frac{f_{max} - f}{f_{max}} \times (p_{m,max} - p_{m,min}) + p_{m,min} \tag{5}$$

In order to ensure the evolutionary algorithm doesn't settle on local maxima of the search space, occasionally the algorithm will select the sub-optimal solution to mutate. The probability p_r of this occurring is determined according to the same formula for determining p_m. We determine the limits of (5) by trial and error.

3.3 Evolved Solution

This algorithm took approximately three days[1] to find the following maximum in the search space:

$$c_{t=1,x,y} = 0.95\rho_c + 0.1 \bigtriangledown +0.75\psi_{min} - 1.0\psi_{max} - 0.9\rho_\psi \quad (6)$$

$$c_{t+1,x,y} = c_t + \alpha(0.95\rho_c + 0.77\bar{c} + 0.1 \bigtriangledown +0.75\psi_{min} - 1.0\psi_{max} - 0.9\rho_\psi) \quad (7)$$

Figure 1 shows the salient regions detected by this algorithm over 30 iterations. Figure 2 compares the salient regions detected by this algorithm with those detected by other salient region detectors. The input images for both figures are from a set of images used to evaluate the performance of the AIM algorithm. None of these images are part of the evolutionary algorithm training set.

Fig. 1. Example output of iterations t=5,10,15,20

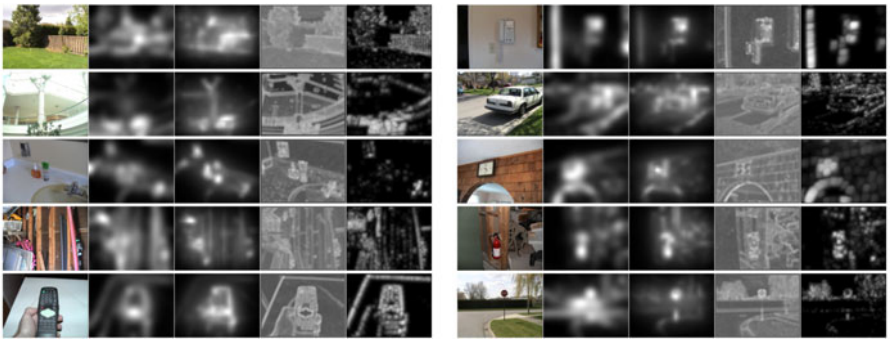

Fig. 2. Input images from [4], then salient regions detected by Itti, GBVS, AIM and CA algorithms

4 Performance of CA Salient Region Detectors

The salient regions detected by the CA are comparible in location and detail to those of the Itti and GBVS algorithms; the detail returned by the AIM algorithm is much greater than the others but at a cost to the speed of the algorithm. We now need to test the sensitivity of this algorithm to transformations of the input, that is, the salient regions of a scene should be the same regardless of the point of view, illumination conditions and quality of the image capture device.

[1] On an Intel Core 2, 2.8Ghz with an Nvidia C2050 GPU.

4.1 Performance Metric

We test our salient region detector using a standard sequence of images provided by Mikolajczyk[9]. This set includes various textured and structured images and their transforms under illumination, scale, viewpoint, JPEG compression and blur. This test was designed to compare detectors that return feature locations and descriptions, so we have adapted the test to compare region detectors instead.

Each set of images consists of the same scene subject to an increasingly large transformation. Where this affects the image geometry this transformation is described by a known affine transformation, $(\mathbf{A}, \bar{b})$. The error introduced by the transformation is calculated as the sum-of-squared differences between the salient regions detected in the original image and the transformed image.

1. The salient regions, $s(I)$ of both the original image I_0 and the transformed image I_t are determined.
2. The affine transformation, $\overline{x_t} = \mathbf{A}\overline{x} + \overline{b}$, of I_t is used to correct for the distortion of I_t, giving $s(I_t)'$.
3. The affine transformation of I will have affected the size and location of the scene present in I_t. The part of I that is visible in I_t is calculated by taking the inverse affine transform of the corners, c of the bounding box of I_0:

$$\begin{pmatrix} c_x^t \\ c_y^t \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{A}^{-1} & -\mathbf{A}^{-1}\overline{b} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} c_x \\ c_y \\ 1 \end{pmatrix} \tag{8}$$

4. The sum of squared differences,e_t between $s(I_0)$ and $s(I_t)'$ is calculated for every pixel in this common part.
5. To compensate for the reduced area,a_t, (and thus potential error) of I_t, the error introduced by the transformation,e is determined as $e = e_t a_0 / a_t$.

4.2 Results

Figure 3(a) shows the speed of this algorithm compared to that of the Itti, AIM and GBVS algorithms. Note that the CA algorithm speed is primarily executed on a GPU (Nvidia C2050) whereas the other algorithms are executed, using MATLAB code provided by the authors, on a 2.8Ghz CPU. Figures 3(b-f) compare the sensitivity of this algorithm to various input transformations. From these results we can see the CA salient region detector performs well under conditions of varying illumination, scale and viewpoint. However it does not perform as well under compression and blur variance. We believe this is due to the dependence of the algorithm on changes to local gradient and orientation and might be corrected by the use of various de-blurring convolution filters. Furthermore we believe the error due to viewpoint change could be improved by the use of circular neighbourhoods about each cell, instead of the rectangular neighbourhoods the algorithm currently uses. The speed of the CA algorithm decreases relative to image size faster than the Itti and GBVS algorithm. This is because as the CA gets larger, it also requires more iterations to distribute feature information across the automata.

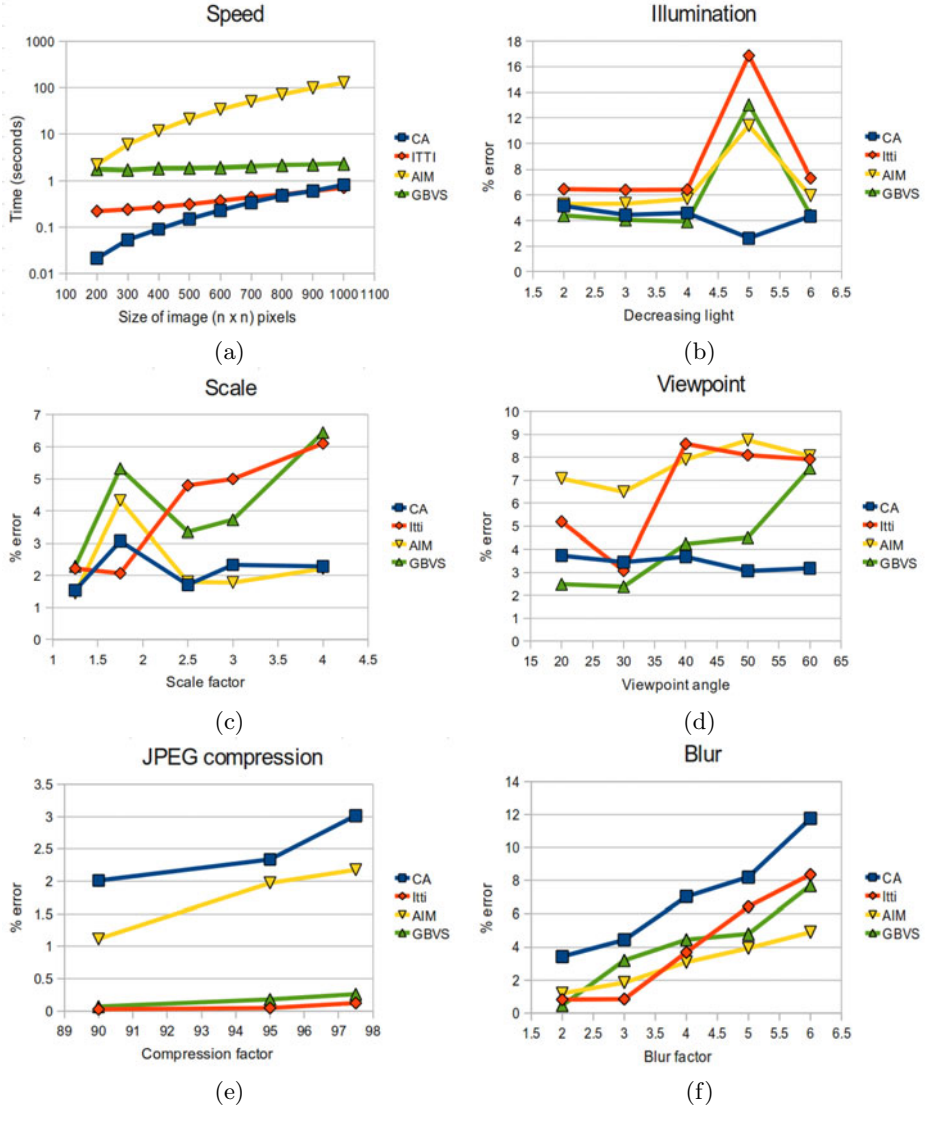

Fig. 3. (a) Speed vs Image size; (b-f) Error due to various input image transforms

5 Conclusions

The CA on a GPU architecture we propose is a biologically plausible emulation of the architecture of the visual cortex and runs considerably faster than CPU alternatives. With the assistance of an evolutionary design algorithm we have designed this CA to perform salient region detection. This salient region detector has proved more invariant to illumination, scale and viewpoint transforms

than three other salient region detectors (Itti, AIM and GBVS). However this algorithm was more susceptible to variation under compression and blur transformations.

Acknowledgements

The authors acknowledge the support received from EPSRC on grants EP/C549481 and EP/E045472.

References

1. Markram, H.: The blue brain project. Nature Reviews Neuroscience 7, 153–160 (2006)
2. Itti, L., Koch, C., Niebur, E.: A model of saliency-based visual attention for rapid scene analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence 20, 1254–1259 (1998)
3. Harel, J., Koch, C., Perona, P.: Graph-based visual saliency. In: Proceedings of NIPS (2006)
4. Bruce, N., Tsotsos, J.: Attention, and visual search: An information theoretic approach. Journal of Vision 9(3), 1–24 (2009)
5. Kansal, A.R., Torquato, S., Harsh, G.R., Chiocca, E.A., Deisboeck, T.S.: Simulated brain tumor growth dynamics using a three-dimensional cellular automaton. Journal of theoretical biology 203, 367–382 (2000)
6. Chang, C., Zhang, Y., Gdong, Y.: Cellular automata for edge detection of images. In: Proceedings of 2004 International Conference on Machine Learning and Cybernetics, vol. 6, pp. 3830–3834 (August 2004)
7. Morie, T., Nagata, M., Iwata, A.: Design of a pixel-parallel feature extraction vlsi system for biologically-inspired object recognition methods. In: Proc. Int. Symp. on Nonlinear Theory and its Applications, pp. 371–374 (October 2001)
8. Levi, D.: Hereboy: a fast evolutionary algorithm. In: Proceedings of the 2nd NASA/DoD Workshop on Evolvable Hardware, pp. 17–24 (2000)
9. Mikolajczyki, K., Tuytelaars, T., Schmid, C., Zisserman, A., Matas, J., Schaffalitzky, F., Kadir, T., Gool, L.V.: A comparison of affine region detectors. Internation Journal of Computer Vision 65, 43–72 (2005)