GPU implementation of motion estimation for visual saliency

Anis Rahman, Dominique Houzet, Denis Pellerin, Lionel Agud

GPU implementation of motion estimation for visual saliency

Anis RAHMAN, Dominique HOUZET, Denis PELLERIN and Lionel AGUD
Gipsa-lab, 961 rue de la Houille Blanche, BP 46
38402 Grenoble Cedex, France
{anis.rahman, dominique.houzet, denis.pellerin, lionel.agud}@gipsa-lab.grenoble-inp.fr

Abstract

Visual attention is a complex concept that includes many processes to find the region of concentration in a visual scene. In this paper, we discuss a spatio-temporal visual saliency model where the visual information contained in videos is divided into two types: static and dynamic that are processed by two separate pathways. These pathways produce intermediate saliency maps that are merged together to get salient regions distinct from what surround them. Evidently, to realize a more robust model will involve inclusion of more complex processes. Likewise, the dynamic pathway of the model involves compute-intensive motion estimation,that when implemented on GPU resulted in a speedup of up to $40x$ *against its sequential counterpart. The implementation involves a number of code and memory optimizations to get the performance gains, resultantly materializing real-time video analysis capability for the visual saliency model.*

1. Introduction

The motivation behind the mimicking human vision system is to develop a complete computer vision system. The research in this field includes methods to extract useful information for proper characterization of a visual scene, thus making possible its analysis for a computer system. One of these many techniques are by estimating the motion to find the regions of attention.
Motion estimation is a process to examine movement of an object by generating motion vectors for a sequence of images. The motion estimator implemented in this paper is proposed by Bruno et al. [3], which afterwards is incorporated into the spatio-temporal visual saliency model developed by Marat et al. [7]. The model proposed focuses on the information extracted from the visual field, without any top-down modalities, and in the end resulting in a region of attention.
The saliency model is inspired from Feature Integration Theory [13] that involves the decomposition of the visual field into various simple attributes. These are afterwards treated using several processes, and at the end are combined to find the regions that dominate others. The complete model is compute-intensive, hence it is a perfect candidate for parallelization to be used for real-time video analysis.
GPU presents us with an inherently parallel architecture for image-based algorithms that are often complex and time-consuming. The desired performance can be extracted off these highly parallel GPUs by employing various methods to optimize the memory usage and thread allocation/synchronization. In this paper, we implement the dynamic pathway of the visual saliency model on the GPU. We concentrate on the optimization of the most compute-intensive kernel, the motion estimator, to achieve the desired speedup. The paper is organized as follows: In section 2, we presented a short introduction to the two pathway spatio-temporal visual saliency model. In section 3, we summarize the dynamic pathway of the model with a brief description of the motion estimator used. Afterwards, the section summarizes our implemented algorithm with the different optimizations made. In section 4, we report the achieved speedups, profiling, and validity of these results is evaluated. In the end, a conclusion of the article and its future prospects are discussed.

2. Spatio-temporal visual saliency model

The bottom-up model [7] illustrated in figure 1 is linearly modeled from the retina to the visual cortical cells. Moreover, the separation of useful information into two distinct signals makes the processing of the information relatively easier. Furthermore, the partial maps from the two pathways are fused together into a single saliency map. All these features contribute in developing a model based on the human visual system.

Static pathway The static pathway is based on retinal filtering, which then is followed by a bank of Gabor filters.

The main modalities used here are frequencies and orientations. The retinal model imitates the horizontal, bipolar, and ganglion cells to expose more detail by increasing the luminance of higher frequencies in the visual input. The primary visual cortex is a model of simple cell receptive fields sensitive to visual signal orientations and spatial frequencies. This is modeled as a 2D Gabor filter bank that processes the visual information in different frequencies and orientations, resulting in 24 partial maps. These filters demonstrate optimal localization properties and good compromise of resolution between frequency and spatial domains. In primate visual system, the response of cells is dependent on their neuronal environment; its lateral connections. Thus, this activity is modeled as a linear combination of simple cells interacting with their neighbors. Afterwards, intermediate energy maps are strengthened by normalization and summed up to extract a saliency map for the static pathway.

Dynamic pathway On the other hand, the dynamic pathway finds a saliency map from a moving scene. As a preprocessing, this pathway uses camera motion compensation [9] for estimation of dominant motion of the salient regions according to their background. This estimation is followed by a 2D motion estimation [3] to find local motion with respect to the background. The motion vectors are calculated with modalities of speed, orientation and direction, which are then treated with a temporal median filtering to remove noise and to produce a saliency map. Ultimately, the resulting maps from both pathways are fused together into a master map with salient regions; one's with highest energy.

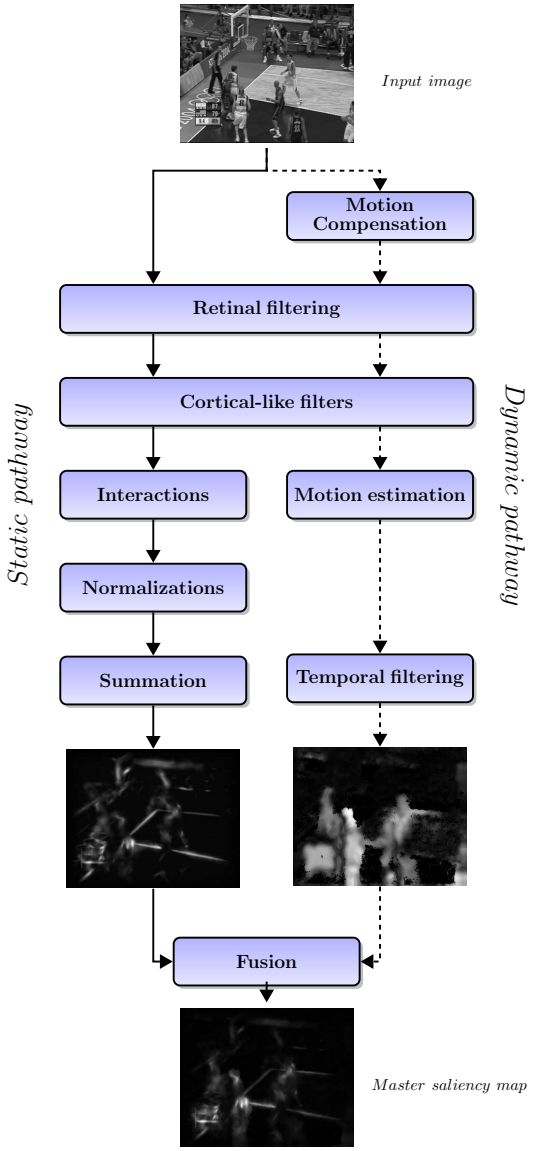

Figure 1. Block diagram of spatio-temporal visual saliency model

3. Implementation of the dynamic pathway

CUDA [1] and OpenCL [5] threading model virtually launches a sea of light-weight threads onto the streaming multiprocessors on the graphics hardware. All in all, this provides fine-grain data parallelism across threads and coarse-grain data and task parallelism across thread blocks. Therefore, almost any algorithm exhibiting data-parallelism is suited to be ported on to graphics processors, and exploiting its raw computational power.

The first step is to develop a sequential version, which can be used to select and code the compute-intensive functional units as kernels that are to be executed on graphics processors. This task is fairly easier as this version acts as a well-structured reference.

The main functions of the dynamic pathway of the visual saliency model include: applying the retinal filtering, convolving with the Gabor filter bank to decompose the rescaled images into sub-bands with different orientations, solving the over-determined system of equations, applying Gaussian prefiltering, calculating spatial, and temporal gradient maps. All these functions are coded and tested individually.

The algorithm 1 presents the various steps of the dynamic pathway, where the first step is camera motion compensation done on the host. This compensation is then followed by other functions implemented on GPU like retina filter, Gabor filter, motion estimator, Gaussian prefiltering, and many others. At the end of this pathway, we get the dynamic saliency map.

input : Input images Im(t) and Im(t-1)
output: Dynamic saliency map M_s

1 Im(t-1) ← `MotionCompensation` (Im(t-1));
2 (pyr(t) ,pyr(t-1)) ← `CalculatePyramid` (Im(t) ,Im(t-1));
3 `RetinalFilter` (pyr(t) ,pyr(t-1));
4 V(t) ← `MotionEstimation` (pyr(t) ,pyr(t-1));
5 M_s ← `TemporalFilter` (V(t, t-1, t-2, t-3));

Algorithm 1: Dynamic pathway of visual saliency model

3.1. Motion estimator

The motion estimator [3] presented here employs a differential method using Gabor filters to estimate local motion. This estimation is done by solving a robust system of equations of optical flow. The method employed is to find the pixels that conserve their luminance for some interval of time. The motion vector $\upsilon(p) = (\upsilon_x, \upsilon_y)$ can be found using the equation of optical flow:

$$\frac{\mathrm{d}I\,(p,t)}{\mathrm{d}t} = \bigtriangledown I\,(p,t) \cdot \upsilon\,(p) + \frac{\partial I\,(p,t)}{\partial t} = 0$$

where $\bigtriangledown I(x,y,t)$ represents the spatial gradients of luminance $I(x,y,t)$. We can use the direction of the velocity component from these gradients to find the direction of the motion of an object.

The motion vectors are calculated by averaging the movement to its spatial neighborhood. This spatial continuity within the optical flow is achieved by the convolving the spatio-temporal image sequence with a Gabor filter bank:

$$\upsilon_x \cdot \frac{\partial\,(I * G_i)}{\partial_x} + \upsilon_y \cdot \frac{\partial\,(I * G_i)}{\partial_y} + \frac{\partial\,(I * G_i)}{\partial_t} = 0$$

the 2D filter bank comprises of N filters Gi with the same radial frequencies. Hence, resulting in a system of N equations for each pixel:

$$\begin{pmatrix} \Omega_2^x & \Omega_1^y \\ \Omega_2^x & \Omega_2^y \\ \cdots & \\ \Omega_n^x & \Omega_n^y \end{pmatrix} \cdot \begin{pmatrix} \upsilon_x \\ \upsilon_y \end{pmatrix} = \begin{pmatrix} \Omega_2^t \\ \Omega_2^t \\ \cdots \\ \Omega_n^t \end{pmatrix}$$

this over-determined system is solved using least squares method(Biweight Tuckey test) [3].
The approximation begins with a sub-sampled image from the highest level of the pyramid. This multi-pattern approach allows robust estimation of motion for every pixel.

Input: Pyramid of sub-sampled images
Output: Velocity vectors V

foreach *Level k of pyramid* **do**
  if $k == K$ **then**
    Gabor filtering $I(x,y,t)$
    Gabor filtering $I(x,y,t\text{-}1)$
  end
  else
    Projection V
    Interpolation $I(x,y,t)$
    Gabor filtering $I(x,y,t)$
    Gabor filtering $I(x,y,t-1)$
  end
  (G_x, G_y, G_t)← Calculate gradients
  Vector v ←Resolution $I(x,y,t)$, $I(x,y,t\text{-}1)$, G
  if $k == K$ **then**
    $V \leftarrow v$
  end
  else
    $V \leftarrow V + v$
  end
  Gaussian filtering V
end

Algorithm 2: Motion estimator algorithm

3.2. Different optimizations

Reducing global accesses After convolving with Gabor filters and applying the spatial and temporal gradients, we get an over-determined system of N solutions for each pixel of the image. This system is resolved using iterative weighted least-square estimations. This method involves the calculation of $N(2N-1)$ intermediate values that are stored as local array variables. Furthermore, these values are accessed at different multiple times, which leads to higher global memory accesses because these arrays are placed on the device memory. To reduce these accesses, we use shared memory. However, the amount of shared memory is limited that makes it impossible to eliminate all the local array variables. As a result, this improvement is not visible because shared memory allocated affects the number of active thread blocks residing on a single multiprocessor.

Reducing register count The estimator kernel is involving several steps that are complex, and causes the register count on the boundary. The higher register count results in a limited number of active thread blocks per streaming multiprocessor, thus the occupancy is low.
In our first implementation, the register count is 22 that is reduced to 15 with the use of shared memory to store some of the local variables. As a result, the number of active

thread blocks increases to 8 per streaming multiprocessor, and the device occupancy increases from 0.33 to 0.67. The use of constant memory would be a preferable solution, in case if each variable is accessed by all the threads simultaneously. Here, they happen to be used and modified in several instructions, thus the candidate variables to be placed in the shared region are carefully selected to minimize the need for synchronization barriers.

Using texture memory The estimation kernel comprises of two operations: one to find N(N-1) equations for the pixel, and then averaging them to get components of its motion vector. To reduce the complexity of the kernel, we cut and place these two operations into separate device kernels. This result is reduced complexity of the kernel for further optimizations and tweaks. On the other hand, it results in calculating and storing of the intermediate equations from the application of spatial and temporal gradients on to the device memory. These equations are used by the second kernel to approximate the motion. Here, the use of texture memory provides faster accesses to these prefetched values, rather than using local arrays that resulted in slower device memory accesses. Therefore, it helps to avoid global memory latency by taking advantage of caching mechanisms of the texture memory, hence, it results up to 10% performance improvement.

Table 1. Profile of motion estimator kernel

	No. of registers	Occupancy	GPU time
Naive solution	18	0.75	32
Shared memory	29	0.5	31.5
Shared + prefetching	32	0.5	31.7
Shared + textures	17	0.375	32

3.3. Multi-GPU solution

In such a multi-threaded model, there is complexity involved in selecting an efficient strategy for creation and destruction of threads, resource utilization, and load balancing. Furthermore, inter-GPU communication between GPUs might be an overhead affecting the overall performance. This overhead can be tackled by overlapping communications with computations using streaming available in the CUDA programming model.

In our implementation of the saliency model, we find that the static pathway is faster than the dynamic pathway of the model. To test for more improvement, we employ a three GPU shared system, where the first GPU executes the static pathway while the dynamic pathway is divided into two halves with almost the same execution times as shown in figure 2. We launch the threads executing their portion of the calculation of the saliency model on their assigned GPU. The threads are initially in sleep mode, and they wait for a signal from the main thread that their inputs are available to be processed. The threads copy their inputs onto their assigned GPU's device memory, and then invoke the device kernels. After the completion of the kernels, the results are copied back to the host, where the next frame is waiting to be fetched for processing.

The estimator kernel takes about 120ms, hence we cut the kernel into two almost equal portions of 60ms as shown in figure 2. The first kernel applies the retina and Gabor filtering, while the other performs the approximation of local motion using the resulting system of equations from the previous kernel. Evidently, there is an expensive device to device transfer within thread 1 and thread 2, which involves the data to be copied on to host memory. Afterwards, thread 2 is responsible for the estimation.

In our multi-GPU implementation, we have three threads for the three available graphics devices each responsible for their portion of the visual saliency model as following: thread 0 calculates the static saliency map, thread 1 applies the two filters, while thread 2 uses the outputted system of equations from thread 1 to estimate the motion vectors. After the temporal filtering, thread two results in dynamic saliency map that is copied back to the host memory. Finally, the main thread fuses the resulting saliency maps from thread 0 and thread 2 into the final visual saliency map.

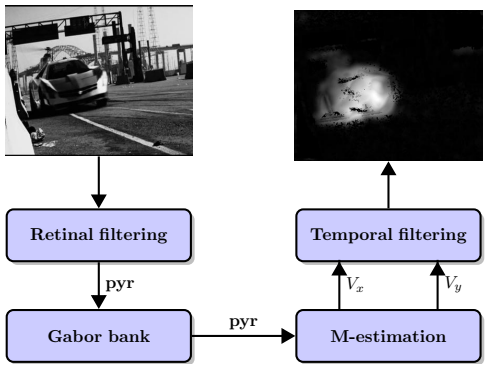

Figure 2. Block diagram for partitioned dynamic pathway

4. Results

All implementations are tested on a 2.67 GHz quad-core system with 10GB of main memory, and Windows 7 running on it. The parallel version is implemented using lat-

est CUDA v3.0, and is evaluated on a NVIDIA GeForce GTX 285 graphics card. The static and dynamic pathways are evaluated with various image sizes, including standard 640×480 pixels, whereas the result of the estimator is evaluated using several datasets of image sequences with sizes ranging from 150×150 to 316×252 pixels.

4.1. Speedup

The CUDA implementation is tested against on the newest GeForce GTX 285. The device has 30 streaming multiprocessor with total 240 cores of clock rates 1.48 GHz each, providing 1.062 TFLOPS of single-precision and 89 GFLOPS of double precision computational power with memory bandwidth 159 GB/sec.The figure 3 shows the gains achieved for the static and dynamic pathway with different image sizes against their sequential versions.

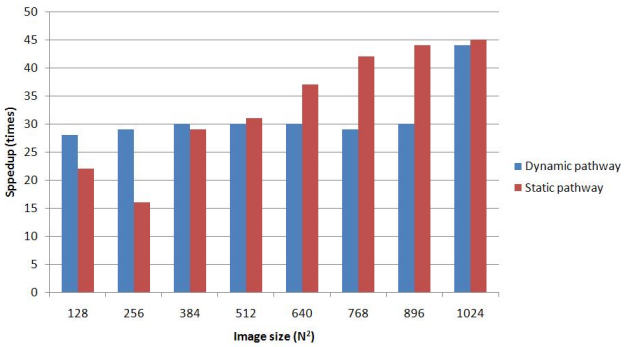

Figure 3. Speedups for two pathways versus sequential C implementation

4.2. Profiling of the pathway

Table 2 illustrates GPU time for different steps of the dynamic pathways of the visual saliency model on a GTX 285 graphics card with test image sequences of size 640×480 pixels.

Figure 4 shows the timings for the static and dynamic pathway on a GTX 285 graphics card using various image sizes. The timing for both pathways starts up at about the same level but as the image size gets bigger the computation time for dynamic pathway increases. The reason is due to the increased complexity of the estimator algorithm involving more computations and device memory accesses.

4.3. Evaluation of results

To evaluate the correctness of the motion estimator, we calculate error between estimated and real optical flows us-

Table 2. Computational cost of each step in dynamic pathway

Kernel Name	GeForce GTX 285 (ms)	GPU time(%)
Estimator	26.91	33.15
Horizontal Gaussian recursive	25.73	31.71
Vertical Gaussian recursive	18.04	22.68
Retinal Filtering	5.35	6.02
Demodulation	1.63	2.00
Calculate gradients	1.37	1.67
Modulation	0.76	0.93
High pass filter	0.15	0.17
Median filter	0.12	0.14
Interpolation	0.10	0.12
Projection	0.04	0.04
Memory transfers	0,47	0.24
Total	81.26	

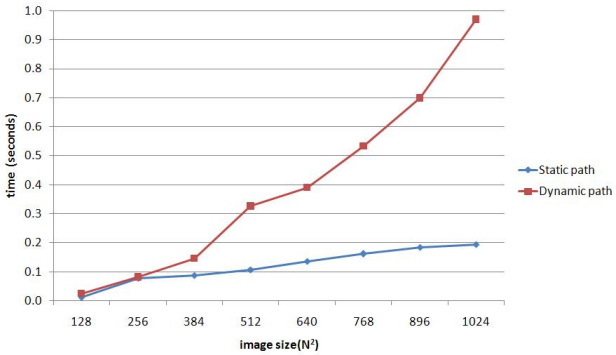

Figure 4. Timings for two pathways on GTX 285 for various image sizes

ing the equation below:

$$\alpha_e = \arccos\left(\frac{uu_r + vv_r + 1}{\sqrt{u^2+v^2+1}\sqrt{u_r^2+v_r^2+1}}\right)$$

where a_e is the angular error for a given pixel with (u, v) the estimated and (u_r, v_r) the real motion vectors. We used "treetran" and "treediv" image sequences for the evaluation, showing translational and divergent motion respectively [3]. The results illustrated in table 3 are obtained using "treetran" and "treediv" image sequences of sizes 150×150 pixels.

Table 3. Evaluating the M-estimator

	Angular error			
	treetran		treediv	
	$\bar{x}$	σ	$\bar{x}$	σ
Matlab	1.63	5.27	6.06	8.22
C	1.10	0.99	4.15	2.69
CUDA	1.19	1.00	5.73	3.91

4.4. Real-time solution

OpenCV [2] and OpenVIDIA [4] are open source platforms providing functions for interfacing to video input and output devices. It also facilitates the programming on GPU through a collection of utility functions, and various implementations of image processing and computer vision algorithms. These algorithms range to include feature detection and tracking, skin tone tracking, projective panoramas, and a lot more.
After the parallel implementation of the spatio-temporal visual saliency model, we used OpenCV for interfacing with webcam and videos to demonstrate the real-time processing of the model. This demonstration resulted in execution of the model at 25 fps on GeForce 285GTX. Therefore, the performance gains achieved shows that GPUs can be a strong candidate for parallelization of computer vision algorithms.

4.5. Conclusion

The GPU implementation for dynamic pathway of the visual saliency model comprises of several compute-intensive kernels like recursive Gaussian filters, the motion estimator, and many others. After various optimizations and tweaks in the code, the final solution resulted in a speedup of up to 40 times against its sequential implementation. Hence, it is now possible to achieve real-time video analysis using the model. Furthermore, the acceleration allows us to add more attributes to enhance the quality of information extracted from a visual scene like the integration of depth proposed in [10]. Moreover, it will be interesting to investigate the inclusion of top-down processes as proposed in [11]. Finally, the performance gains on GPU will enable our model to be used for various applications such as scene recognition for mobile robots [12], video compression [6, 8], computer graphics rendering [14, 15]. The idea, here is to determine a region with better saliency requiring good spatial resolution.

References

[1] *NVIDIA CUDA Compute Unified Device Architecture - Programming Guide*, 2007.

[2] G. Bradski and A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media Inc., 2008.

[3] E. Bruno and D. Pellerin. Robust motion estimation using spatial gabor-like filters. *Signal Process*, 82:297–309, 2002.

[4] J. Fung, S. Mann, and C. Aimone. Openvidia: Parallel gpu computer vision. In *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*, 2005.

[5] K. Group. Opencl - the open standard for parallel programming of heterogeneous systems.

[6] E. Larson, V. Cuong, and M. Chandler. Can visual fixation patterns improve image fidelity assessment ? In *IEEE International Conference on Image Processing*, 2008.

[7] S. Marat, T. Ho Phuoc, L. Granjon, N. Guyader, D. Pellerin, and A. Guérin-Dugué. Modelling spatio-temporal saliency to predict gaze direction for short videos. *Int. J. Comput. Vision*, 82:231–243, 2009.

[8] A. Ninassi, O. L. Meur, P. L. Callet, and D. Barba. Does where you gaze on an image affect your perception of quality? applying visual attention to image quality. In *IEEE International Conference on Image Processing*, 2007.

[9] J.-M. Odobez and P. Bouthemy. Robust multiresolution estimation of parametric motion models applied to complex scenes. *Journal of Visual Communication and Image Representation*, 6:348–365, 1994.

[10] N. Ouerhani. *Visual attention: from bio-inspired modeling to real-time implementation*. PhD thesis, Université de Neuchâtel, Suisse, 2004.

[11] R. J. Peters and L. Itti. Beyond bottom-up : Incorporating task-dependent influences into a computational model of spatial attention. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[12] C. Siagian and L. Itti. Rapid biologically-inspired scene classification using features shared with visual attention. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29:300–312, 2007.

[13] A. M. Treisman and G. Gelade. A feature-integration theory of attention. *Cognitive Psychology*, 12(1):97–136, January 1980.

[14] H. Yee and S. Pattanaik. *In neurobiology of attention*, chapter Attention for computer graphics rendering, pages 649–651. Elsevier Academic Press, 2005.

[15] H. Yee, S. Pattanaik, and D. P. Greenberg. Spatiotemporal sensitivity and visual attention for efficient rendering of dynamic environments. *ACM Transactions on Graphics*, 20:39–65, 2001.