Moving Object Detection for Real-Time Applications

Lucia Maddalena
National Research Council
Institute for High-Performance
Computing and Networking
Via P. Castellino 111, 80131 Naples, Italy
lucia.maddalena@na.icar.cnr.it

Alfredo Petrosino
University of Naples Parthenope
Department of Applied Science
Via A. De Gasperi 5, 80133 Naples, Italy
alfredo.petrosino@uniparthenope.it

Abstract

Detection of moving objects in video streams is the first relevant step of information extraction in many computer vision applications. Aside from the intrinsic usefulness of being able to segment video streams into moving and background components, detecting moving objects provides a focus of attention for recognition, classification, and activity analysis, making these later steps more efficient.

We present some extensions to the method for moving object detection presented in [4]. Our main contributions are related to the pre-processing of intermediate results (transience maps), aimed at enhancing the accuracy of detection results, and to the parallelization of some of the most computationally intensive steps using SSE2 instructions, in order to enhance efficiency and allow for real-time applications.

1. Introduction

Detection of moving objects in video streams is the first relevant step of information extraction in many computer vision applications, including traffic monitoring, automated remote video surveillance, and people tracking [3]. Aside from the intrinsic usefulness of being able to segment video streams into moving and background components, detecting moving objects provides a focus of attention for recognition, classification, and activity analysis, making these later steps more efficient, since only moving pixels need be considered [2]. The problem is known to be significant and difficult [8]. Conventional approaches to moving object detection include temporal differencing [7], background subtraction [8], and optical flow [1].

Temporal differencing takes into account differences in consecutive sequence frames, which allow to discern static objects (having null differences) from moving objects (having non-null differences). This approach is very adaptive to dynamic environments, but it is strictly dependent on the velocity of moving objects in the scene and it is subject to the foreground aperture problem. In contrast, optical flow techniques aim at computing an approximation to the 2D motion field (projection of the 3D velocities of surface points onto the imaging surface) from spatio-temporal patterns of image intensity [1]. They can be used to detect independently moving objects in the presence of camera motion, but most optical flow computation methods are computationally complex, and cannot be applied to full-frame video streams in real-time without specialized hardware.

Surely background subtraction is the most common and efficient method to tackle the problem (e.g. [6]). It is based on the comparison of the current sequence frame with a reference background, including information on the scene without moving objects. It is independent on the velocity of moving objects and it is not subject to the foreground aperture problem, but it is extremely sensitive to dynamic scene changes due to lighting and extraneous events. Although these are usually detected, they leave behind holes where the newly exposed background imagery differs from the known background model (ghosts). While the background model eventually adapts to these holes, they generate false alarms for a short period of time.

Among different approaches, the one proposed in [4] allows disambiguation of moving objects that stop for a while, are occluded by other objects, and then resume motion. Layered detection is based on two processes: pixel analysis and region analysis. *Pixel analysis* determines whether a pixel is *stationary* or *transient* over time, while *region analysis* detects stationary regions of stationary pixels corresponding to stopped objects. We adopted the layered approach, including a pre-processing of transience maps aimed at suppressing shadows and reducing noise, and clustering non-background pixels using region growing.

Moreover, the need for real-time systems imposes very low computation times. We focused on most computationally intensive steps of the proposed approach, obtaining parallel modules for several tasks. Specifically, we adopted the SIMD approach. SIMD architectures operate concurrently in a single instruction on multiple data. Their usage is especially suited for applications where huge amount of data must undergo the same processing, such as multimedia applications. Therefore we traduced in assembler most time consuming routines, taking advantage of SIMD architectures.

The paper is organized as follows. In Section 2 we give a brief description of the approach adopted for moving object detection. In Section 3 we illustrate the basics of parallelization of computationally demanding steps. In Section 4 we present results obtained with the above mentioned parallelization, while Section 5 includes conclusions and further research directions.

2. Approach to moving object detection

A robust detection system should be able to recognize when objects have stopped and even disambiguate overlapping objects - functions usually not possible with traditional motion detection algorithms. The approach for moving object detection based on layered adaptive background subtraction [4] allows quite efficiently the detection of overlapping objects. Layered detection is based on two processes: pixel analysis and region analysis.

2.1. Pixel analysis

Pixel analysis determines whether a pixel is stationary or transient by observing its intensity value over time. Moving objects passing through a pixel cause an intensity profile step change, followed by a period of instability; then the profile stabilizes, in a manner dependent on the kind of event. To capture the nature of changes in pixel intensity profiles, a gradient based approach is applied.
Let $I_t(x)$ be the intensity of pixel x at a time t occurring k frames in the past. The *motion trigger* T prior to the frame of interest t is the maximum absolute difference between the pixel intensity $I_t(x)$ and its value in the previous l frames:

$$T = \max_{j=1,\ldots,l} \left\{ |I_t(x) - I_{t-j}(x)| \right\}, \qquad (1)$$

where suggested value for l is 5 [2]. Let us also introduce the *stability measure* as the variance of the pixel intensity profile from time t to the present:

$$S = \frac{(k+1)\sum_{j=0}^{k} I^2_{t+j}(x) - \left(\sum_{j=0}^{k} I_{t+j}(x)\right)^2}{k(k+1)},$$

where k is set to correspond to one second of video [2]. Once T and S have been computed, a *transience map* M can be defined for each pixel, taking three possible values: background (BG), transient (TR), or stationary (ST). For each pixel, the corresponding map value is updated to TR if it was ST or BG and if motion trigger is greater than a given threshold (there has been a step change in intensity). Moreover, if it was TR and stability measure is lower than a given threshold (intensity has been stabilized), it is updated to BG if its stabilized intensity value is equal (within a threshold Th) to the background intensity value, and to ST otherwise.

In order to allow for adaptivity of the background model to slow lighting changes, we update the background B by running average with selectivity. Specifically, background model B_t is initially set to the first image ($B_0(x) = I_0(x)$ for every pixel x), and then updated as:

$$B_{t+1}(x) = \begin{cases} \alpha B_t(x) + (1-\alpha) I_t(x), & x \text{ non} - \text{moving} \\ B_t(x), & \text{otherwise} \end{cases} \qquad (2)$$

where α is a time constant that specifies how fast new information supplants old observations, usually chosen in [0.9, 1].

2.2. Region analysis

Non-background pixels in the transience map M are clustered into regions. In [2, 4] clustering is obtained using a nearest neighbor spatial filter; in our implementation, after a pre-processing of the transience map (see §2.3), we cluster non-background pixels using a connected component labeling algorithm based on region growing.

Each spatial region is then analyzed and classified as moving or stopped object on the basis of the number of transient or stationary pixels it includes. According to the algorithm reported in [4], regions that consist of stationary pixels are added as a new layer over the background. A layer management process is used to determine when stopped objects resume motion or are occluded by other moving or stationary objects.

2.3. Transience map pre-processing

Prior to performing region analysis, we pre-process the transience map, in order to suppress shadows and reduce noise.

Shadows in a scene represent a problem for video surveillance systems, especially those operating outdoor. In fact, the cast shadow of an object (the shadow area projected on the scene by the object) alters the shape of the object itself, leading to errors in the measurement of its geometrical properties. This affects both the classification and the assessment of moving object position, making uncertain the subsequent moving object tracking. Moreover, cast shadows of two or more objects can create a false adjacency between the objects, which leads to detecting them as a single

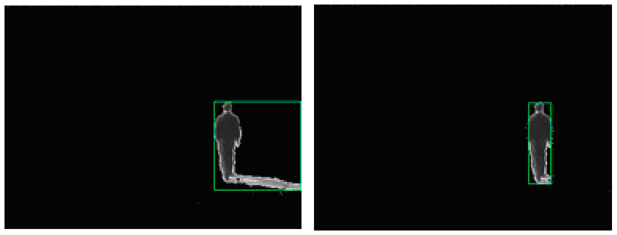


Figure 1. Example of shadow suppression.

object. This affects many higher level surveillance tasks, such as counting and classifying individual moving objects in a scene. Instead, shadows in the background do not pose big problems, as long as the background is correctly updated during time.

Among the many approaches to moving cast shadow suppression, we have adopted the one reported in [3], which proved to be quite accurate and suitable for moving object detection. The approach is based on the HSV color model, which closely correspond to human color perception and has been proved more suitable for detecting shadows compared to RGB model, being able to separate chrominance and intensity information.

For each pixel belonging to the foreground (either stationary or transient), a binary mask is constructed, indicating shadow pixels (value 1). Let $I_k^H(x,y), I_k^S(x,y)$, and $I_k^V(x,y)$ be the hue, saturation, and value components of pixel (x,y) of image I at time k, and assume analogous notation for components of the background image B. The *shadow mask* is defined as:

$$SP_k(x,y) = \begin{cases} 1 & \text{if} \alpha \leq \frac{I_k^V(x,y)}{B_k^V(x,y)} \leq \beta \text{ and} \\ & I_k^S(x,y) - B_k^S(x,y) \leq \tau_S \text{ and} \\ & |I_k^H(x,y) - B_k^H(x,y)| \leq \tau_H \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

with $0 < \alpha < \beta < 1$, where all parameter values are empirically tuned and proved stable under environment changes. The three conditions for identifying a foreground pixel as shadow derive from the observation that in a shadowed area there is a significant illumination variation, but only a small color variation.

An example of application of the algorithm for shadow reduction is given in Figure 1.

The output of shadow suppression, after being binarized, is filtered with a morphological opening using a 3×3 structuring element, in order to reduce noise due to sudden illumination changes or small camera movements.

An example of the complete detection process is given in Figure 2, where we show: (a) an input image; (b) the transience map M computed as described in §2.1 (white pixels are stationary pixels, while green pixels are transient pixels); (c) the transience map M pre-processed as described in §2.3; (d) the output image, obtained as described in §2.2 (the green rectangle indicates moving objects, while the yellow rectangle indicates stationary objects).

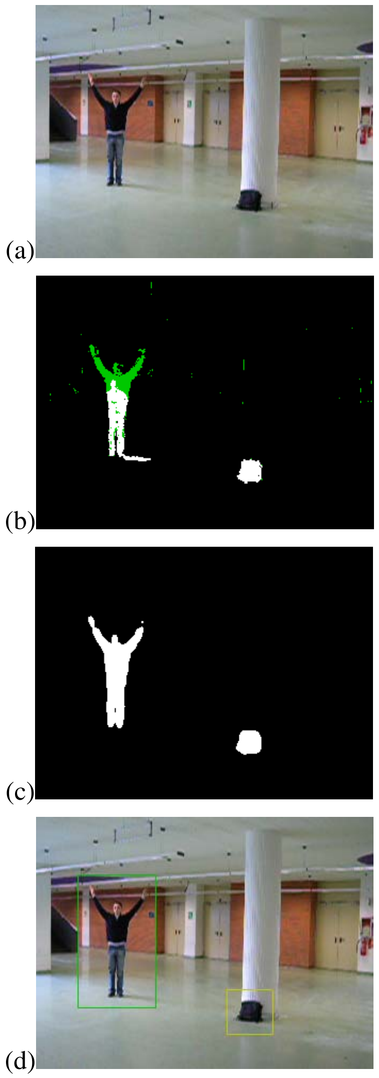


Figure 2. The complete detection process: (a) input image; (b) Transience map; (c) Preprocessed transience map; (d) output image.

For quantitative detection results we adopted the usual Recall and Precision functions computed over tp (true positives), fn (false negatives) and fp (false positives):

$$Recall = \frac{\sum tp}{\sum tp + \sum fn},$$

where $(\sum tp + \sum fn)$ is the total number of objects in the *ground truth*, and

$$Precision = \frac{\sum tp}{\sum tp + \sum fp},$$

where $(\sum tp + \sum fp)$ is the total number of detected objects.

Several image sequences have been considered; here, for space constraints, we only report results obtained for the *Walk1* sequence of the CAVIAR Project [5]. The sequence, which is labeled and comprise 600 frames of 384×288 spatial resolution, presents critical factors such as light change and mimetics. Setting the motion trigger threshold to 70 (the stability measure is not taken into account due to the absence of stationary objects in the sequence), the method correctly detected 1217 over 1550 objects, achieving as performance *Recall = 0.79* and *Precision = 0.73*. In Figure 3 the Recall and Precision values are reported in accordance to the variation of the motion trigger threshold. We remark the role of the threshold, since for values less than 70 the method detects a lot of objects, also not present in the ground truth, although the number of the relevant ones is not significantly greater. This corresponds to smallest Precision values, while letting stable the Recall value.

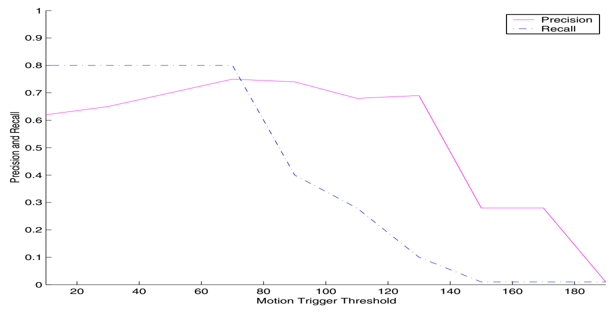


Figure 3. Precision and Recall in terms of the motion trigger threshold.

3. Parallelization

SWAR (SIMD Within a Register) architectures realize concurrency using special registers (SWAR registers) of dimension wider than that of general registers, and suitable instructions on such SWAR registers that allow to operate concurrently on several data loaded into such registers. In order to directly control SWAR registers, it is necessary to program in assembler, and the algorithm must be suitably modified in order to take into account their use.

For our experiments we adopted an Intel Pentium 4 processor, which supports MMX (MultiMedia eXtension), SSE (SIMD Streaming Extension) and SSE2 (SSE extended) extensions. While MMX instructions operate on 64-bit MMX registers physically stored onto usual FPU registers, SSE and SSE2 instructions use a different set of eight 128-bit XMM registers. XMM registers can simultaneously store two 64-bit integers or floats, four 32-bit integers or floats, eight 16-bit integers, or sixteen 8-bit integers. SSE2 instructions (for data movement, data format conversion, arithmetical and logical operations) operate concurrently on XMM registers. Compared with analogous sequential instructions, this approach leads to a speedup which is upper bounded by the number of data items loaded into registers (which depends on the data type chosen).

For our experiments we adopted color RGB images; each pixel is usually represented by 3 bytes (one for each color component) and an image is represented as an array of bytes. The general parallelization strategy consists in working separately on the three RGB channels, using different XMM registers, each containing 16 pixels of one of the color bands. In the case we need a representation with real values (e.g. shadow suppression), we load 4 adjacent pixels for each of the colour bands into the XMM registers. The assembler SSE2 code has been introduced into the ANSI-C source code, using directives that are extensions of ANSI-C (and therefore implemented in different ways in different compilers).

Some of the tasks taken into account for implementation using SSE2 instructions are briefly described in the following.

3.1. Motion trigger

The basic idea of the SSE2 implementation of the algorithm for computing the motion trigger T (see §2.1) consists in applying eqn. (1) concurrently for 16 adjacent pixels loaded into XMM registers.

Since there is no absolute value function that operates on SWAR registers, in order to compute the absolute value differences appearing in eqn. (1) we had to use different instructions. Specifically, the instruction `psubusb` operates a subtraction "with saturation" of two registers, meaning that if the difference value is negative, than it is set to 0 (it is saturated). Computing two subtractions with saturation between the image and the background and or-ing the result, we obtain the absolute value of the difference. Moreover, since there is no compare function that operates on unsigned bytes (but only for signed ones), in order to compare the previous result with the threshold, we first sum the quantity 128 to both the threshold and the previous result, and then compare the obtained signed bytes (using the instruction `pcmpgtb`).

3.2. Background difference

Background difference, used in the construction of the transience table M (see §2.1), can be implemented in SSE2 applying the sequential algorithm concurrently on 16 adjacent pixels loaded into XMM registers.

Absolute value differences between image and background intensity values are computed and compared to the background threshold similarly to the case of motion trigger computation. Results obtained for the three bands of each image are then or-ed for obtaining the final difference image.

3.3. Shadow suppression

The shadow suppression algorithm adopted for the preprocessing of the transience map (see §2.3) consists, for each non background pixel, in:

1. converting from RGB space to HSV space the pixel in the current image and the corresponding pixel in the background image;
2. computing the shadow mask using equation (3) and updating the corresponding element in the transience map.

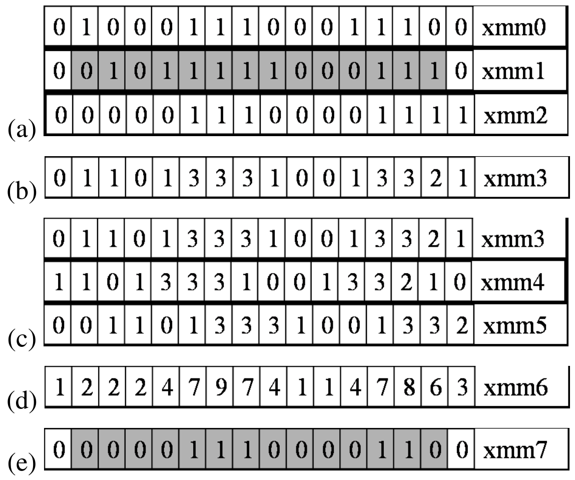
Since input and output data are real (pixel intensities for RGB channels must be normalized in [0,1], and HSV components are real as well), we can load at most four adjacent values into each XMM register. We therefore developed SSE2 implementations for steps 1 and 2 above, working concurrently on four pixels at a time. In both cases, the main concern of SSE2 implementation is in the combination of results depending on if statements.

3.4. Erosion and dilation

Erosion and dilation have been used for the final preprocessing of the transience map (see §2.3). Erosion (dilation) with a 3×3 structuring element is obtained assigning to each pixel of the binary image the value 0 (the value 1) if the number of 8-connected adjacent pixels having value 1 is less (greater) than a given threshold T_e (T_d), and the value 1 (the value 0) otherwise.

To implement erosion and dilation using SSE2 instructions we load 16 adjacent pixels of three consecutive image rows into three XMM registers (see Fig. 4-(a)). Such data allow to compute the result for "center pixels" (dark pixels in central row of Fig. 4-(a)). The result is obtained by

1. summing along the columns (byte by byte) the three registers, obtaining a register that contains the number of white pixels column by column (see Fig. 4-(b));
2. summing three bytes at a time along the register resulting from step 1, to obtain the number of white pixels for each neighborhood of "center pixels", by:
   (a) right shifting and left shifting the register by one byte (see Fig. 4-(c));
   (b) summing the three resulting registers along the columns (see Fig. 4-(d)).
3. comparing the result with the threshold T_e or T_d (see Fig. 4-(e) for erosion with T_e=6).



(a)
0 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 xmm0
0 0 1 0 1 1 1 1 1 0 0 0 1 1 1 0 xmm1
0 0 0 0 0 1 1 1 0 0 0 0 1 1 1 1 xmm2
(b)
0 1 1 0 1 3 3 3 1 0 0 1 3 3 2 1 xmm3
(c)
0 1 1 0 1 3 3 3 1 0 0 1 3 3 2 1 xmm3
1 1 0 1 3 3 3 1 0 0 1 3 3 2 1 0 xmm4
0 0 1 1 0 1 3 3 3 1 0 0 1 3 3 2 xmm5
(d)
1 2 2 2 4 7 9 7 4 1 1 4 7 8 6 3 xmm6
(e)
0 0 0 0 0 1 1 1 0 0 0 0 1 1 0 0 xmm7

Figure 4. Example of erosion: (a) 16 adjacent pixels of 3 consecutive image rows loaded into XMM registers xmm0, xmm1, xmm2 to compute the result for central (dark) pixels; (b) result of step 1 over xmm0, xmm1, xmm2; (c) register xmm3 together with registers xmm4 and xmm5 obtained by left and right shifting of xmm3; (d) result of step 2 over xmm3, xmm4, xmm5; (e) result of step 3 over xmm6 (threshold T_e=6).

3.5. Background update

As usual, the basic idea of the SSE2 implementation of the algorithm for the update of the background B (see §2.1) consists in loading 16 adjacent pixels into XMM registers and applying the sequential algorithm concurrently on such data, according to eqn. (2).

The main concern here is the need of dealing at the same time with byte data (image and background) and real data (the α parameter). The problem has been afforded unpacking bytes to float, performing the necessary computations four floats at a time, and packing back to bytes the results.

4. Experimental results

We implemented the above described procedures on a Pentium 4 with 2.40 GHz and 512 MB RAM, running Windows XP operating system. We tested them on several color image sequences; for space constraints, here we report results for just one sequence of 400 color images with size 320 $\times$ 240. All timings have been obtained using the Windows high-resolution performance counters (QueryPerformanceCounter and QueryPerformanceFrequency).

In Table 1 we compare times (in msecs) obtained as the mean execution times for each frame on the whole video with the implementation of the sequential and the SSE2 algorithms for all tasks reported in §3. Moreover, we report speedup values, obtained as the ratio of sequential and SSE2 execution times. Results for dilation are not shown, since they perfectly agree with those obtained for erosion.

Task	**(1) Seq. time**	**(2) SSE2 time**	**Speedup (1)/(2)**
Motion trigger	3.49	0.39	8.95
Backg. difference	1.30	0.10	13.00
Shadow suppr.	0.61	0.29	2.10
Erosion	1.11	0.12	9.25
Backg. update	7.98	0.50	15.96

Table 1. Mean execution times (in msecs) of the sequential and the SSE2 implementations for all tasks reported in §3 on color images of size 320 $\times$ 240, and related speedup.

Here we can notice that in most cases good speedup values could be achieved. Specifically, in all the cases where we could operate solely on byte data (motion trigger, background difference, erosion and dilation), we achieved speedup values not too distant from the ideal speedup of 16 (16 bytes into a XMM register). In the case of shadow suppression, where we had to operate on float data, achieved speedup is more than half of the ideal speedup of 4 (4 floats into a XMM register); this result can be considered appreciable if we consider that available parallelism is very limited here, since computations apply only to non background pixels (see §3.3). Finally, in the case where we had a mixture of byte and float data (background update) the adopted parallelization strategy has led to an extremely high speedup value.

In the general case, it should be observed that, even if the number of operations and memory accesses in the parallel implementation decrease by a factor of 16 (for byte data) or 4 (for float data) compared to the sequential case, execution times reduce by a smaller factor. This is mainly due to the fact that the standard and the SWAR instruction sets are noticeably different: not all instructions are present in both sets and the number of clock cycles for the same instruction is different (generally SWAR instructions require more clock cycles).

5. Conclusions

We presented a method for moving object detection that allows disambiguation of moving objects that stop for a while, are occluded by other objects, and that then resume motion. Such method strongly relies on the work presented in [4]. Our main contributions are related to the preprocessing of transience maps, aimed at suppressing shadows and reducing noise, and clustering non-background pixels using region growing. Moreover, being concerned with real-time applications, we focused on some of the most computationally intensive steps of the proposed approach, obtaining SIMD parallel modules for several tasks: motion trigger, background difference, background update, shadow suppression and morphological operations. First experimental results show that in most cases we could achieve speedup values close to the ideal speedup. Future research will be devoted to the parallelization of other tasks of the moving object detection process.

References

[1] J. Barron, D. Fleet, and S. Beauchemin. Performance of optical flow techniques. *IJCV*, 12, no. 1:42–77, 1994.

[2] R. T. Collins, A. J. Lipton, T. Kanade, H. Fujiyoshi, D. Duggins, Y. Tsin, D. Tolliver, N. Enomoto, O. Hasegawa, P. Burt, and L. Wixson. A system for video surveillance and monitoring. *The Robotics Institute, Carnegie Mellon University*, CMU-RI-TR-00-12, 2000.

[3] R. Cucchiara, M. Piccardi, and A. Prati. Detecting moving objects, ghosts, and shadows in video streams. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(10):1–6, 2003.

[4] H. Fujiyoshi and T. Kanade. Layered detection for multiple overlapping objects. *IEICE Trans. Inf. & Syst.*, E87-D, no. 12:2821–2827, 2004.

[5] http://homepages.inf.ed.ac.uk/rbf/CAVIARDATA1/.

[6] M. Piccardi. Background subtraction techniques: a review. *Proc. of IEEE SMC 2004 International Conference on Systems, Man and Cybernetics*, 2004.

[7] P. Rosin and T. Ellis. Image difference threshold strategies and shadow detection. *Proc. British Machine Vision Conference*, pages 347–356, 1995.

[8] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers. Wallflower: Principles and practice of background maintenance. *Proceedings of the Seventh IEEE Conference on Computer Vision*, 1:255–261, 1999.