Ultra Fast Grey Scale Face Detection Using Vector SIMD Programming

Olivier Vermeulen, Antoine Manzanera, Lionel Lacassagne

Ultra fast grey scale face detection using vector SIMD programming

O. Vermeulen
UEI
ENSTA
Paris

A. Manzanera
UEI
ENSTA
Paris

L. Lacassagne
IEF/AXIS
Univ. Paris XI
Orsay

Abstract

This paper presents an ultra-fast detection algorithm for locating faces in grey scale videos. We first use motion detection to reduce the working area and find the approximate position of the head. Then a morphology-based technique is applied in this area to detect eye-analogue and lips-analogue regions. Next, the resulting components are used to search for potential facial features. Finally we select from the candidate triplets, the one that best represents a real face, calculating a fitness which takes into account things such as the symmetry and the proximity with the extrapolated position of the face. In order to achieve the maximal speed-up, we use the vector parallelism provided by the SIMD (Simple Instruction Multiple Data) extensions, available on most mainstream processors. The final program runs 65 times faster than the real-time. Experiments demonstrate that the success rate for single face videos reaches 85% in good conditions and can go down to 60% in harder cases. This approach can be useful in many applications, where the detection rate is not as important as the computation time, such as video face identification, or human-computer visual interfaces.

1. Introduction

In recent years, detecting human faces or facial features has become an important task in computer vision with numerous potential applications including human computer interaction, video surveillance and face recognition. The objective is to determine whether or not there is any face in the image and, if any, return its location. The difficulty of face and facial features detection can be attributed to the following factors [1]:

- *Pose*: Face images vary due to the relative camera-face pose and some facial features may become partially or wholly occluded.
- *Structural components*: Facial features such as beards, moustaches and glasses may be present or not.
- *Facial expression*: The appearance of faces depends on a personal facial expression.
- *Occlusion*: Faces may be partially occluded by other objects such as hand, scarf, etc.
- *Illumination*: Face images vary due to the position of the light source (shadows).

Despite these problems, the recent literature on face detection presents a wide variety of approaches with some successful results. There are five categories of face and facial feature detection:

1. *Geometry-based methods.* These methods utilise geometrical information [2]. Each feature is considered as a geometrical shape. These methods are generally accurate, but also sensitive to distortions such as occlusions or noise.
2. *Colour-based approaches.* These approaches have difficulties in robustly detecting skin colours in the presence of a complex background and different illuminations [3], [4].
3. *Appearance-based methods.* These methods use the models learnt from a set of training images [5], [6]. Gray value is the most important parameter for the detection. They are not able to perfectly detect face images with poor quality in intensity and some occlusions.
4. *Motion-based methods.* Face and facial features are detected from the image sequence [7], exploiting the motion information.
5. *Edge-based methods.* Faces are detected from the edge information [8], [9]. The goal is to handle larger variations of the face appearance by being less dependent on illumination and motion.

Our detector uses both geometry-based and motion-based methods. The philosophy of our approach is the following: In many applications, such as face recognition or human-machine interfaces, the input is a video sequence. In this case, reaching a very high detection rate for every image is not compulsory. On the other hand, achieving the minimal computation time (or power, or surface) is very important in order to allow further processing. Our goal here is to minimise the computation time. We will present the algorithms in detail, the different optimisations and their effects. We will also use vector parallelism thanks to the SIMD (Simple Instruction Multiple Data) Within a Register (SWAR) paradigm [10]. The instruction set used in our experiments is Altivec [11], available on Power PC processors, but equivalent results can be obtained using SSE2 instructions on Intel and AMD processors. The principle is to use 128-bit vector registers that can represent sixteen 8-bit signed or unsigned characters, eight 16-bit signed or unsigned short integers, four 32-bit integer or floating point variables. As image processing often involve regular operations, we can expect an acceleration equivalent to the number of variables loaded in the vector register.

The first stage of our detector is an approximate localisation of the head based on motion detection. It is presented in Section 2. The output of the first stage is a region of interest, support of computation of the second stage: the facial feature detection is performed by morphology-based method, which provides a segmentation of eye-analogue and lips-analogue components. The potential triplets eyes-mouth are then sorted and selected from their geometrical properties. The second stage is detailed in Section 3. The experimental results and performances are discussed in Section 4. Finally Section 5 concludes the paper.

2. Motion-based head localisation

In applications using a video sequence as input, the faces (at least for living people) are always mobile, even a little: blinking, lips moving, that can be captured by motion detection using a stationary camera. We then exploit this motion information, both to reduce the computation time by reducing the "working area", and to improve the face detection by removing lots of potential false alarms in the background.

A simple motion detection is performed by updating a background image B_t, then computing the "moving pixels" by thresholding the absolute difference between the current image I_t and the background. More precisely, the operations performed are the following. Initialising $B_0 = I_0$, then, for every time index t, we update the background image by $B_t = \alpha I_t + (1-\alpha)B_{t-1}$. $\alpha \in]0,1[$ is an oblivion rate, whose dimension is inverse of the number of frames. So taking α small increases the sensitivity but decreases the precision of localisation and vice versa. We choose

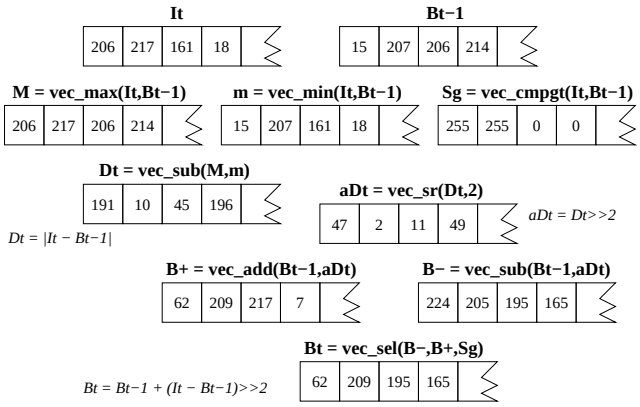


Figure 1. Vector computation of the background subtraction

$\alpha = 2^{-k}$ so that we can use bit-shifting to perform the multiplication. The detection is finally performed by thresholding the absolute difference: $L_t = (D_t > K)$, where $D_t = |I_t - B_{t-1}|$, and K a threshold parameter. L_t then provides the binary image of the motion labels.

This all part of the algorithm can be sped up using vector SIMD programming. To get the maximal speedup, i.e. ($\times 16$) by operating 8-bit images on a 128-bit register, all the operators must preserve the 8-bit dynamics of the operands. Another constraint of the SWAR is to transform conditional instructions into a set of comparisons and selections (equivalent to traditional AND masks). Such transformations increase the number of instructions and then moderate the speed-up. Figure 1 shows an example of vector computation of B_t and D_t using Altivec instructions.

The next step of the algorithm consists in finding the outline of the movement. This is to say, from the binary image of motion labels L_t, we compute, for each column x of the image, the position y of the highest pixel such that $L_t(x,y) = 1$. To be less sensitive to noise, we actually compute, for each column, the position of the highest block of P consecutive pixels such that $L_t = 1$, where P is a defined constant. The result is a vector which size equals the image width representing the outline of movement (see example on Figure 2, with $P = 4$). As all the column are processed independently, this part of the algorithm can be easily vectorised.

The outline vector `Outline[i]` is then used to estimate the approximate location of the head, i.e. the coordinates $\{(x_1,y_1),(x_2,y_2)\}$ of a rectangular region of interest (ROI) within which the facial features will be searched. To do this, we perform the following operations:

`Outline[i]` is first transformed into one increasing vector `OutINC[i]` and one decreasing vector `OutDEC[i]`. Those new vectors are computed by using

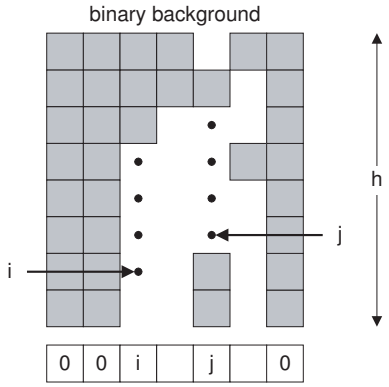


Figure 2. Computation of the outline and resulting vector

two "max" filter: one causal, operating from left to right, and one anti-causal, operating from right to left. More precisely:

```
OutINC[0] = Outline[0];
for (i=1;i<W;i++)
  OutINC[i] = max(Outline[i],OutINC[i-1]);
OutDEC[W-1] = Outline[W-1];
for (i=W-2;i>0;i--)
  OutDEC[i] = max(Outline[i],OutDEC[i+1]);
```

Where `W` is the size of the vector (i.e. the width of the image). The left (resp. right) border of the ROI x_1 (resp. x_2) is then defined as the last significant increasing step while scanning `OutINC[i]` from left to right (resp. `OutDEC[i]` from right to left):

$$x_1 = \arg \max_{0<i<W} \texttt{OutINC[i]} - \texttt{OutINC[i-1]} > \texttt{STEP}$$
$$x_2 = \arg \min_{0<i<W} \texttt{OutDEC[i-1]} - \texttt{OutDEC[i]} > \texttt{STEP}$$

Where $STEP$ is a threshold parameter.

The top border of the ROI y_1 is simply the highest value of the outline vector, i.e.:

$$y_1 = \texttt{OutINC[W-1]} = \texttt{OutDEC[0]}$$

The last coordinate y_2 is finally deduced from morphological proportion of human face:

$$y_2 = y_1 - \lambda(x_2 - x_1)$$

with $\lambda = 1.4$ in our experiments.

This last part of the function are computed by scanning a 1D vector. Then it is not adapted to a vector parallelism. Nevertheless, the computation of x_1 and x_2 are independent and can be performed by 2 concurrent threads. Figure 3 summarises the approximate face localisation method.

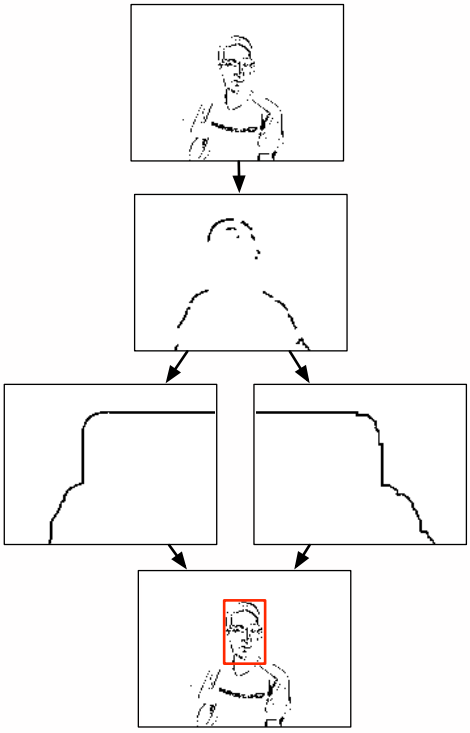


Figure 3. Principle of the approximate face localisation method

3. Facial features detection

As shown by various studies, some facial elements such as eyebrows, eyes, nostrils and mouth are always darker than the rest of the face [12], [13]. Among these facial components, the eyes and the mouth have proved the most stable in appearance with respect to illumination, posture or facial expression.

Therefore, in this section we use a morphology-based method to locate in the ROI the eye-analogue and lips-analogue components as the first step of the facial feature detection. The connected components are first localised, then go through a selection process whose purpose is to find, amongst the candidate features, the three components which best fits a mouth-eyes triplet.

3.1. Morphological processing

Following Han et al [14], we use the morphological top-hat operator to detect the narrow dark regions, which will form the potential eye-analogue and mouth-analogue components. It is computed as the difference between the morphological closing and the original image:

$$\tau_B(X) = \varphi_B(X) - X = \delta_B(\varepsilon_B(X)) - X$$

From a computational point of view, this function makes extensive use of dilation δ_B (resp. erosion ε_B) operator, which consists in computing for every pixel the max (resp. min) over a neighbourhood defined by the structuring element B. Then the optimisation of this primitives is really important.

In our algorithm, we use a combination of recursive scan-based computation of the min/max, which allows to compute erosion/dilation on 1D segment with a constant complexity (i.e. independent on the segment length), and data parallelism, which permits simultaneous computations of erosion/dilation on the direction orthogonal to the structuring element.

The constant time computation of 1D erosion/dilation was proposed independently by van Herk [15] and Gil and Werman [16]. It is mentioned as HGW algorithm from now on. This algorithm allows to reduce the number of operations per pixel to 3 min (or 3 max), whatever the length of the structuring element.

HGW is a 3 steps algorithm. Suppose we want to compute an erosion using a vertical (and symmetric) structuring element of size L. Imagine that every column of the image is partitioned in contiguous blocks of size L. For each block, the minimum is propagated recursively, first from top to bottom:

```
for (j = 0;j < H;j++)
  if (j%L == 0)
    A[x,j] = I[x,j];
  else
    A[x,j] = min(A[x,j-1],I[x,j]);
```

Second, from bottom to top:

```
for (j = H-1;j > 0;j--)
  if (j%L == 0)
    B[x,j] = I[x,j];
  else
    B[x,j] = min(B[x,j+1],I[x,j]);
```

Note that the computation of vectors A and B are independent and then can be computed simultaneously. The erosion is finally achieved by computing a last minimum operator:

```
for (j = 0;j < H;j++)
    E[x,j] = min(A[x,j+L/2],B[x,j-L/2]);
```

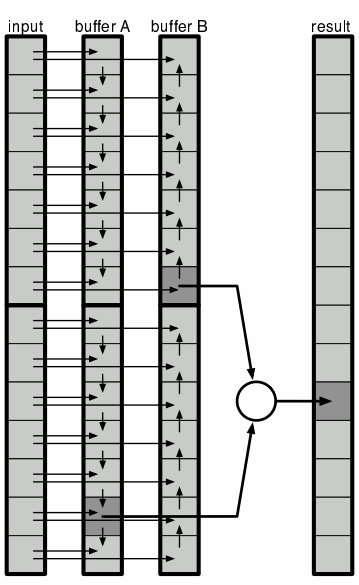
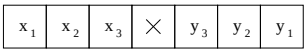


Figure 4. HGW algorithm for constant time erosion/dilation

Figure 4 shows the principle for an erosion of size 7.

Using a vertical structuring element, it is clear that each column can be processed independently. Then using vector parallelism allows to process 16 columns at the same time, accelerating dramatically this part of our algorithm.

The output of the top-hat operators are thresholded to form a binary image (the threshold is chosen as the average value of the output), representing the facial features candidates. When the background is complex, however, lots of false candidates may appear, as mentioned by [14]. A higher level selection process is then necessary.



$$S_1 = |I(x_1) - I(y_1)| + |I(x_2) - I(y_2)| + |I(x_3) - I(y_3)|$$

Figure 5. Test of symmetry for the components

3.2. Components inventory and selection

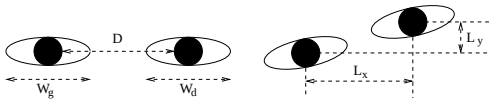
The first step is to list and localise the different components from the previous binary image, made up of some groups of white pixels over a black background. We then perform a connected components inventory, which output is the list of connected components, with their areas (number of pixels), and bounding boxes (determining their rough localisation). For this, we use an adaptation of the classical algorithm of Rosenfeld [17], based on 2 image scans, and using an equivalence table to resolve the ambiguities due to the scan order. This algorithm is well-adapted to real-time constraints, as its complexity is less data-dependent than the other algorithms, and it is mostly regular, i.e. operates on contiguous memory, thus optimising cache management. Furthermore, we do not need a true labelling (i.e. an image associating to every pixel a different label for each connected component), but only the number of pixels and the bounding boxes. All this information is available after the graph transitive closure, at the end of the first scan, then we do not need a second scan, unlike the original Rosenfeld algorithm.

After the connected components inventory, we first sort the components using their area (number of pixels). We remove those whose area is smaller than a defined percentage (0.1%) of the area selected by the head position detector.

As the facial features we want to detect present a high degree of vertical symmetry, we also perform the following symmetry test on the components: Starting from the calculated centre of the component, we take 3 pixels on each side, horizontally. Then we calculate the sum of the absolute differences between the opposite pixels (See Figure 5). The components whose value S_1 is beneath a certain threshold are considered asymmetric and discarded.

At this stage, we have got a set of n components corresponding to the potential features. The rest of the selection is performed by a set of morphometric rules inspired by [18]. We first check all the $\begin{pmatrix} n \\ 2 \end{pmatrix}$ possible pairs of components, and compute the list of the p pairs matching the conditions shown on Figure 6. Note that some conditions use a parameter W_{head} corresponding to the estimation of the head's width, and which is deduced from the head localisation output.

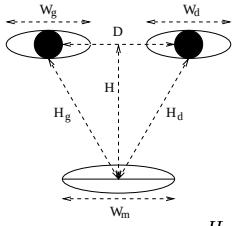
The next set of morphometric rules is checked for the $n \times p$ potential triplets of features, deduced from the two



(a) (b)

$$0.5 \leq \frac{W_g}{W_d} \leq 2 \quad 0.1 \leq \frac{W_g}{D} \leq 0.4 \quad 0.1 \leq \frac{W_d}{D} \leq 0.4$$
$$L_x \leq 0.8 \times W_{head} \quad L_y \leq 0.4 \times 1.4 \times W_{head}$$

Figure 6. Set of tests for eyes pairs



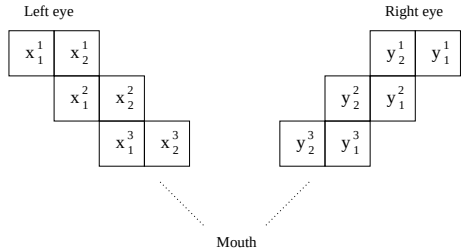
$$H_g \simeq H_d \qquad 1 \leq \frac{H}{D} \leq 1.4$$
$$0.2 \leq \frac{W_g}{W_m} \leq 0.6 \quad 0.2 \leq \frac{W_d}{W_m} \leq 0.6$$

Figure 7. Set of tests for faces

last steps. First we check that the component supposed to represent the mouth is located under and between the eyes. Then the conditions represented on Figure 7 are computed.

At this point we have a few triangles representing the potential triplets eyes-mouth. In order to select the best one, we compute for each triangle a fitness which takes into account (1) the symmetry and (2) the proximity with the extrapolated position of the face. The symmetry is calculated between the left and the right sides of the triangle, as shown on Figure 8 (Computing the symmetry on this two sides only works better than in the whole triangle, specially in the case of profiled faces).

On the other hand, using the last 3 positions of the detected face, we extrapolate the position of the new triangle. Then for each triangle, we calculate its distance from the extrapolated triangle. Finally the fitness is a weighted sum



$$S_2 = \sum_i |I(x_1^i) - I(y_1^i)| + |I(x_2^i) - I(y_2^i)|$$

Figure 8. Test of symmetry for the face

Time(ms)	Motion	Outline	Total
Scalar	4.9	1.95	6.85
Altivec	0.380	0.234	0.614
Speedup	$\times 12.9$	$\times 8.3$	$\times 11.2$

Table 1. Comparison between optimised scalar and vector versions.

of this distance and the previous symmetry factor. Then the triangle that best represents a real face is the one with the smallest fitness.

4. Results and benchmarking

Figures 9 and 10 show some detection results (the triangle is actually drawn on the video stream using the Bresenham's line algorithm [19]). These first results are promising, since our algorithm is able to detect one face with or without glasses, even with slight rotations. Experiments demonstrate that the success rate reaches 85% in good conditions and can go down to 60% in harder conditions (quick motion or strong rotations).

We now discuss the performance obtained for the real-time implementation of the detector. In order to estimate the impact of vectorisation, two versions of the algorithms have been benchmarked: one optimised scalar C code, and one with Altivec intrinsics. The computation of the proposed algorithm can be split into two parts: the operators on the whole image (for motion detection and outline extraction steps) and those on the extracted ROI (morphological processing, connected component inventory, selection and drawing). The second part of the algorithm has a very low execution time, since the ROI usually represents less than 10% of the image. Then we will only distinguish 2 steps in the following analysis: (1) Motion detection, (2) Outline extraction together with the operators working on the ROI.

The benchmark has been done on a Macintosh G5 running at 2.5 GHz connected to a Fire-Wire camera at 25 Hz and using C+FOX API [20] by Joel Falcou. Table 1 provides the execution time in ms and the speedup for the two steps.

Motion detection step has a speedup smaller than the theoretical one ($\times 16$) because the computational ratio (number of instructions divided by the number of memory accesses) is low. We are faced to a memory wall problem, where the memory bandwidth is the bottleneck of the algorithm. For outline extraction, the vectorisation is better and even greater than the max ($\times 8$ here, because of 16 bit data manipulation) because of cache effects and a better computational ratio (only one memory access per column instead of one per pixel). The global speedup due to Altivec vectorisa-

Figure 9. Some correct detection results

Figure 10. Some false detections

tion is $\times 11.2$, that is $\times 65$ faster than the video constraint of 40 ms. The saved time allows two possibilities: making the algorithm more robust and/or developing other algorithms for face tracking or recognition, still in real-time.

5. Conclusion

We have presented an ultra-fast implementation of a face detection algorithm working on video sequences acquired by stationary camera. For single face scenarios, the per-image detection rate measured is between 60 and 85%, whereas the average computation time is approximately 0.6 ms, hence $65\times$ faster than the real-time. Such implementation can then be useful as a preliminary function in various applications, such as facial identification or human-machine interface systems.

Many improvements remain to be done, however. The balance between the computational loads of the different parts of the algorithm can be changed, by relaxing some constraint or reinforcing other. The effects of this changes on the detection rate must be studied more rigorously. One of the main limitations of our system is that it is practically limited to single face detection, so the extension to multiple face detection will be the first improvement we plan to address in the future.

References

[1] M. Yang, D.J. Kriegman, N. Ahuja, "Detecting faces in images: a survey", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24(1), pp. 34-58, 2002.

[2] F.Y. Shih, C. Chuang, "Automatic extraction of head and face boundaries and facial features", *Information Sciences* vol. 158(1), pp. 117-130, 2004.

[3] F. Tsalakanidou, S. Malassiotis, M.G. Strintzis, "Face localization and authentication using color and depth images", *IEEE Transactions on Image Processing* vol. 14(2), pp. 152-168, 2005.

[4] M. Soriano, B. Martinkauppi, S. Huovinen, M. Laaksonen, "Adaptive skin color modeling using the skin locus for selecting training pixels", *Pattern Recognition*, vol. 36(3), pp. 681-690, 2003.

[5] C. Liu, "A Bayesian discriminating features method for face detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25(6), pp. 725-740, 2003.

[6] P. Viola, M.J. Jones, "Robust real-time face detection", *International Journal of Computer Vision*, vol. 57(2), pp. 137-154, 2004.

[7] E. Loutas, I. Pitas, C. Nikou, "Probabilistic multiple face detection and tracking using entropy measures", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14(1), pp. 128-135, 2004.

[8] S. Phimoltares, C. Lursinsap, K. Chamnongthai, “Tight bounded localization of facial features with color and rotational independence”, Proceedings of the *IEEE International Symposium on Circuits and Systems*, vol. V, pp. 809-812, 2003.

[9] F. Fleuret, D. Geman, “Coarse-to-fine face detection”, *International Journal of Computer Vision*, vol. 41(1-2), pp. 85-107, 2001.

[10] R.J. Fisher, H.G. Dietz, “Compiling for SIMD within a Register”, Proceedings of the *Int. Work. on Languages and Compilers for Parallel Computing*, LNCS vol. 1656/1999, pp 290-304, 1998.

[11] K. Diefendorff, P.K. Dubey, R. Hochsprung, H. Scales, “Altivec Extension to PowerPC Accelerates Media Processing”, *IEEE Micro*, vol. 20(2), pp. 85-95, 2000.

[12] S.H. Jeng, H.Y.M. Liao, C.C. Han, M.Y. Chern, Y.T. Liu, “Facial feature detection using geometrical face model: an efficient approach”, *Pattern Recognition*, vol. 31, pp. 273-282, 1998.

[13] G. Chow, X. Li, “Towards a system for automatic facial feature detection”, *Pattern Recognition*, vol. 26(12), pp. 1739-1755, 1993.

[14] C.C. Han, H.Y.M. Liao, G.J. Yu, L.H. Chen, “Fast face detection via morphology-based pre-processing”, *Pattern Recognition*, vol. 33, pp. 1701-1712, 2000.

[15] M. van Herk, “A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels”, *Pattern Recognition Letters*, vol. 13, pp. 517-521, 1992.

[16] Y. Gil, M. Werman, ”Computing 2D Min, Max and Median Filters”, *IEEE Trans. PAMI*, vol. 15, pp. 504-507, 1993.

[17] A. Rosenfeld, J.L. Pfaltz, “Sequential operations in digital picture processing”, *Journal of the ACM*, vol. 13(4), pp 471-494, 1966.

[18] C.C. Chiang, W.K. Tai, M.T. Yang, Y.T. Huang, C.J. Huang, “A novel method for detecting lips, eyes and faces in real time”, *Real-time imaging*, vol. 9, pp. 277-287, 2003.

[19] J.E. Bresenham, “Algorithm for computer control of a digital plotter”, *IBM Systems Journal*, vol. 4(1), pp. 25-30, 1965.

[20] J. Falcou, “CFOX - A Firewire camera driver for OS X”, `http://cfox.sourceforge.net/`