3

VLSI Implementation of Contour Extraction from Real Time Image Sequences

Elmar Melcher, Lírida Naviner, João Marques de Carvalho, Jean François Naviner, Ricardo A. S. Moreira, Yuri de Mello Villar, Marcos Morais

Universidade Federa da Paraíba - DEE - CCT
58.109-970 Campina Grande - PB, Brasil
elmar@dee.ufpb.br
tel: +55 83 310 1357
fax: +55 83 310 1418

Abstract

In this work a parallel architecture is proposed for VLSI implementation of a data-flow algorithm for 2D boundary (or contour) detection.The algorithm works on the gradient image and uses a set of primitive paths to generate all possible contour paths on a neighborhood defined by a 5x5 window. The objective is to determine whether or not the neighborhood central pixel belongs to a continuous boundary line passing across the window. Test results show that one-pixel wide continuous boundary lines can be extracted using this algorithm.

Keywords

Contour Extraction, Robot Vision, MPEG4

VLSI: Integrated Systems on Silicon R. Reis & L. Claesen (Eds.)

1. INTRODUCTION

Polygonal modeling has traditionally been one of the most popular methods for shape representation and still is largely used in applications where shape recognition of two-dimensional objects or surfaces is needed [1, 2].

Currently a lot of attention is paid to image sequence and video coding due to the increasing importance of high speed multimedia applications [1-3]. In fact, model-based image coding is a powerful technique for compressing head-shoulder images, as in the MPEG-4 sequences imposing very low bitrate coding. In image coding, compression ratio and reconstruction quality are two issues of great importance. The use of adaptive coding based on appropriate image models is one of the most effective approaches to achieve these goals simultaneously. So, polygonal modeling is an important tool for these intelligent coding schemes, where the coder searches each image for objects which are identified according to some underlying objects models [4]. Large compression ratios result since once an object is identified, it can be tracked through a number of frames in a sequence and only subsequent changes in the model parameters (shape, motion, etc.) need to be transmitted.

Polygonal models have the advantage of being a local representation, i.e., they preserve local shape features therefore allowing for object recognition, even in the presence of partial occlusion of objects. Additionally, they can be made insensitive to rotation, translation, and scaling, (a requirement for any practical recognition system) and are much less computationally expensive than higher-order polynomial approximations. As a drawback, the representation provided by polygonal models is usually not as compact as those based on global features such as moments or transform descriptors. A more complete analysis of the issues involved in those and other forms of shape representation has been made by Pavlidis [10]. Further details about the advantages of polygonal modeling can also be found in the literature [8,9].

In order to operate properly, algorithms for polygonal modeling of 2D objects require shapes with continuous and well defined boundaries. Generation of this boundary is the objective of the preprocessing phase to which the original image of the object to be modeled is normally submitted. As part of a typical preprocessing operation initially a discrete gradient operator is employed to generate a gradient image, upon which boundary tracking segmentation can be performed [10].

Most of the work so far reported on algorithms for boundary extraction on digital images assume a sequential software implementation, either on a general purpose computer or on an specialized signal processor. This type of approach is inadequate if real-time high speed operation is desired, due to the computationally intensive character of low-level image processing. Applications such as vision systems for mobile robots or video compression may require 512×512 image frames with 256 gray levels (8 bits) to be processed at a rate of 30 frames per

second. Real-time operation at this rate would require a processing time of 120 nanoseconds for each image element or pixel. This performance figure becomes clearly out of reach for any sequential general purpose computer when one considers the amount of multiplications, additions and other operations usually involved in each output pixel computation. The obvious solution is to design parallel algorithms and architectures which can be implemented in specialized integrated circuits called ASIC's using CAD-based VLSI design tools.

The availability of very powerful and easy to use VLSI design tools has fueled the development of several real-time image processing systems, particularly for low-level feature detection and extraction applications. Bhanu et al. have designed and implemented a real-time segmentation processor which makes use of a gradient relaxation algorithm (iterative) to assign pixels into classes, based on their gray value and the gray values of neighboring pixels [11]. Ranganathan et al. have proposed a VLSI architecture which convolves images with eight 15×15 kernels in order to implement a technique for corner detection which is based on the concept of half-edge and on the first derivative of Gaussian [12]. Cheng et al. utilized the theory of dynamic programming to develop a backtracking method for curve detection and designed an associated VLSI architecture which solves the problem in O(n) time, where n is the length of the curve to be found [13]. All these architectures make use of pipelining and parallelism in order to achieve real-time performance.

In this work a parallel architecture is proposed for VLSI implementation of a data-flow algorithm for 2D boundary (or contour) detection. The algorithm works on the gradient image and uses a set of primitive paths to generate all possible contour paths on a neighborhood defined by a 5×5 window. The objective is to determine whether or not the neighborhood central pixel belongs to a continuous boundary line passing across the window.

The rest of the work is organized as follows: Section 2 describes the algorithm for boundary detection. Section 3 shows results obtained by simulating actual circuit operation. Section 4 estimates hardware implementation cost. Finally, conclusions are drawn in section 5.

2 ALGORITHM

For hardware implementation of image processing, the most performant algorithms are those of the data-flow type. The image pixels come in serially, pixel by pixel, one line after the other. For every incoming image pixel, a data-flow algorithm produces one output pixel. The output pixel is determined by a function of the corresponding input pixel and neighboring pixels. These pixels form a window. Apart from the function, the performance and the data storage requirements of data-flow algorithms depend on the size of the window, as shown in table 1. The storage size and delay times are for images with 8 bits per pixel, 512 pixels per line, 512 lines per image, 30 images per second.

The performance characteristics of data-flow algorithms makes them the best choice for real time image processing.
The boundary detection algorithm proposed in this paper consists of the first three processing stages shown in figure 1. Each stage will be described in detail in the following paragraphs.
Table1 Window size and performance of data-flow algorithms

Window pixel × pixel	*storage space required (bits)*	*image delay (μs)*
2×2	1 line + 2 pixels = 4112	1 pixel = 0.1
3×3	2 lines + 3 pixels = 8216	1 line + 2 pixels = 61.7
5×5	4 lines + 5 pixels = 16424	2 lines + 4 pixels = 123.3

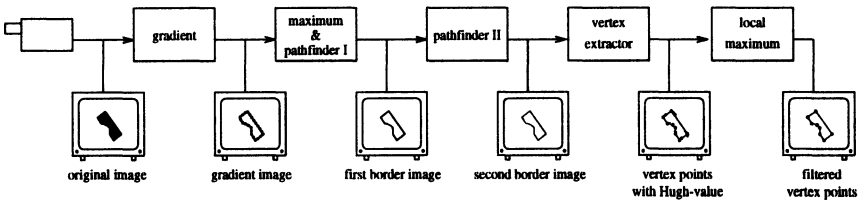


Figure 1 The image processing stages of the proposed algorithm

2.1 Spline Gradient

In a 5×5 matrix we define four masks for the Spline gradient based on the one-dimensional spline coefficients obtained as described in [14]. The two-dimensional spline gradient g_P is obtained from the horzontal, vertical, and diagonal gradients g_{PX}, g_{PY}, g_{P1}, g_{P2}, respectively:

$$g_P = 3(|g_{PX}|+|g_{PY}|)+2(|g_{P1}|+|g_{P2}|) \quad (1)$$

One of the main advantages of this gradient operator is that it is less sensitive to noise. This comes from the fact that the Spline gradient uses a 5×5 window and determines its result from the mean value of horizontal, vertical, and diagonal derivatives. Apart from that the boundaries are sharper, making the gradient values of the contour pixels significantly higher than those in the neighborhood. That happens because the spline interpolation best approximates the discrete points to the original function.

2.2 Local Maximum and Pathfinder I

This phase works with two criteria. If both criteria are validated, the center pixel of the window becomes the output pixel, otherwise the output pixel becomes zero. Note that the output image at this phase is not a binary image. It has as many gray levels as the input image.

The *local maximum* criterion is based on a 5×5 window. A simple criterion would be that if the center pixel were among the five biggest values in the window it would be chosen as a contour pixel [15]. However, this condition is insufficient because in the cases where many pixels had the same gray level value the center pixel had its significance lowered. Therefore, there should be some kind of weight assigned to the center pixel in order to help make the decision. This weight is defined as:

$$W = \begin{cases} 0 & , \text{if } N_S = 0 \\ 2N_S + N_e + p_c & , \text{otherwise} \end{cases} \qquad (2)$$

where, W = weight of center pixel;
N_s = number of pixels smaller than the center pixel;
N_e = number of pixels equal to the center pixel;
p_c = gray level value of the center pixel ($0 \leq p_c < 32$).

The multiplication of N_s by 2 is used to emphasize the importance of this term in comparison to N_e.

The *local maximum* criterion is then validated if the center pixel value is greater than 13% of the full scale value and if its weight is bigger than a *threshold value*, which was found empirically to be 30.

The *pathfinder* criterion also works on a 5×5 window. This criterion determines whether the center pixel of the window belongs to a continuous border line passing across the window. To accomplish this, all possible border paths across a 3×3 window, shown in figure 2a inside the squares formed by the pixels in the *positions 1*, together with their possible continuations, represented by the *positions 2* and *3*, must be checked. The possible paths are derived from the primitives in figure 2a by mirror and rotation operations. Eliminating repetitive patterns, a total of 44 distinct paths was obtained. Note that the paths in figure 2a do not contain any sharp corners, which would difficult subsequent border tracking.

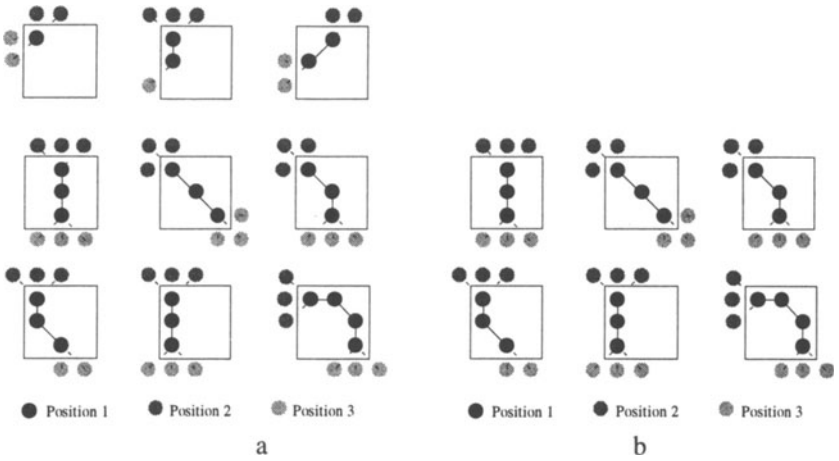


Figure 2 Primitive paths for *pathfinder I (a) and pathfinder II (b)*

The *pathfinder* criterion is validated if the center pixel belongs to two of the eight paths with the largest values. The value of a path is determined by the weighted sum of its pixel values:

$$Ws = \begin{cases} \dfrac{M_2 + M_3}{2} + \dfrac{12}{N_{p1}} \sum_{i=1}^{N_{p1}} p_i & \text{, if } condition \\ 0 & \text{, otherwise} \end{cases} \quad (3)$$

where, Ws = Weighted sum;
N_{p1} = Number of *positions 1* in the 3x3 path;
p_i = Gray level value of the i^{th} pixel in *position 1*;
M_n = Maximum value among the pixels in *position n*.
condition = p_i's and M_n's must be all bigger than 13% of the full gray scale value.
Note that Ws is always an integer value.

2.3 Pathfinder II

The *pathfinder II* algorithm uses only paths containing at least 3 pixels in the *positions 1*. These 28 paths were derived from the primitives in figure 2b applying the same technique used in *pathfinder I*.
The same equation (3) is used to calculate the weighted sum for each path. Then the two highest results, called here W_{s1} and W_{s2} associated to the masks m_1 and m_2, respectively, are determined. If the highest sum, always assumed to be W_{s1}, is equal to zero then the center pixel of the window does necessarily not belong to the path, and the output pixel is set to 0.
A second condition to be analyzed is when W_{s1} and W_{s2} have an equal value and m_1 and m_2 are adjacent (adjacent paths are those which are different from one another by just one pixel). This situation is considered a stalemate, because the two different pixels in each path could be equally labeled contour pixels. This stalemate is resolved with a simple logic. If there is a vertical stalemate, this logic

keeps the pixel on top and discards the other one. If the stalemate is horizontal, the pixel on the left is kept and the one on the right is discarded.
In the absence of stalemate, if m_1 contains a *position 1* in its center the output is set to 1, else it is set to 0.
This criterion assures that only one of the pixels of the different paths is set to 1, providing a one pixel wide border line. In the examples shown in section 3, the stalemate situation occurs at about 4% of the border pixels.

3. SIMULATION RESULTS

Operation of the border detection hardware implementation was simulated by a program written in C language. The choice of C is justified by the fact that it allows fast simulations at functional level providing fast turn-around time for debugging and parameter adjustment. A hardware description language like Verilog or VHDL runs slower and is more complicated to debug.
Images of two objects were used to validate the algorithm: An arch-shaped toy block, figure 3 and a screwdriver, figure 4.
The toy block is made of 64×64 pixels. 32 gray levels (5 bits) are used.
Figure 3a shows a shape with sharp contrast. The gradient's figure 3b reaches strong gray level values all around the border and the contour line is relatively thin. This makes the task of border extraction rather easy, resulting in a perfectly continuous line no more than one pixel wide in figure 3d.

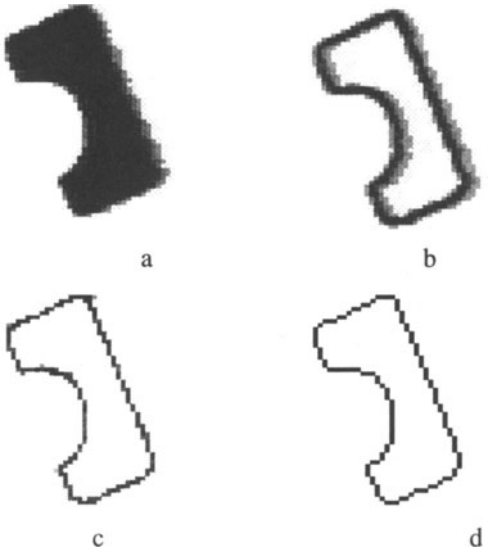
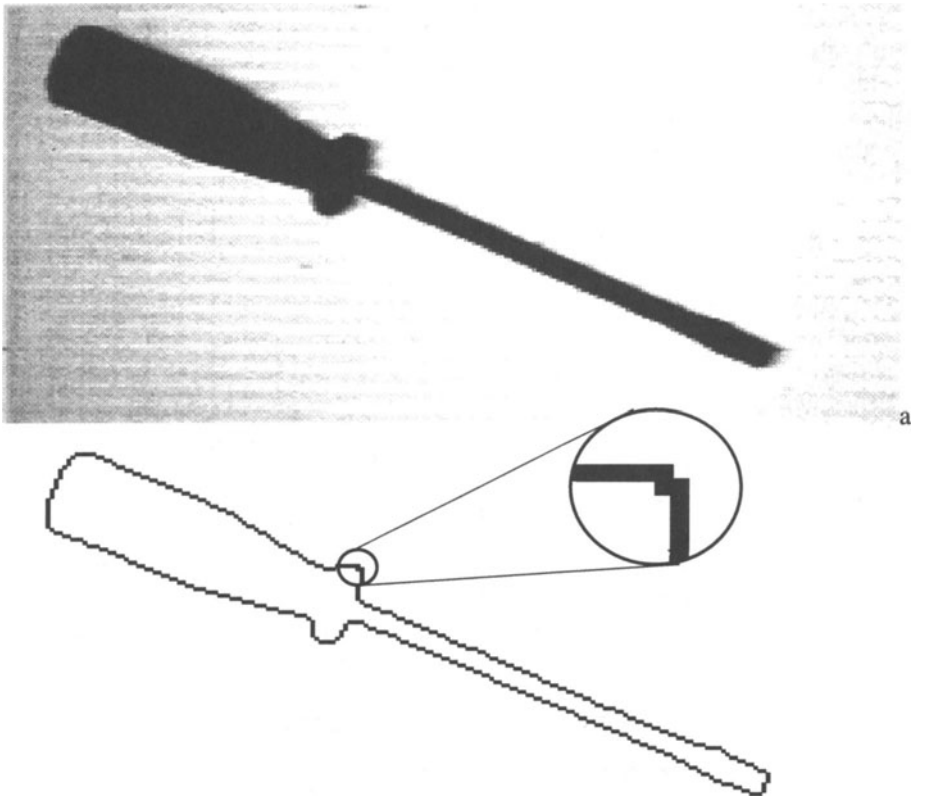


Figure 3 Original image of toy block (a), gradient image (b), first border(c) and second border (d).

The screwdriver is made of 220×128 pixels. 32 gray levels (5 bits) are used.



a

b

Figure 4 Original image of the screwdriver (a) and second border (b).

In comparison to the toy block, the screwdriver in figure 4a has poor contrast, mainly due to its round shape at perpendicular sections of the image plane and because of the shadow caused by light that falls onto the image plane in angle from the left side. To avoid the shadow, the light source should be located next to the camera. The contrast is particularly bad at the tip of the screwdriver and at the top of the handle. Even though, the results obtained after the image was run through the algorithm were very good, except for a two pixel wide border point located in the upper bottom corner of the handle, zoomed in figure 4b.
Still, the quality of the contour in figure 4b is clearly sufficient for a subsequent vertex extraction procedure [16].

4 HARDWARE IMPLEMENTATION

This section will show how the operators used in the algorithm described in section 3 are implemented in hardware. An estimation of chip area required for a 3-metal 0.5µm technology is also given.
The operators described in the following also use the faster system clock to reduce hardware costs by a 5 stage pipelining.
The first operator is the gradient operator. It is divided in two parts. The first part is used in the first 4 pipelining stages to calculate the absolute values of the 4 one-dimensional gradient values g_X, g_Y, g_{P1}, g_{P2}. During the last stage, the weighted sum according to equation (1) is calculated in the second part.

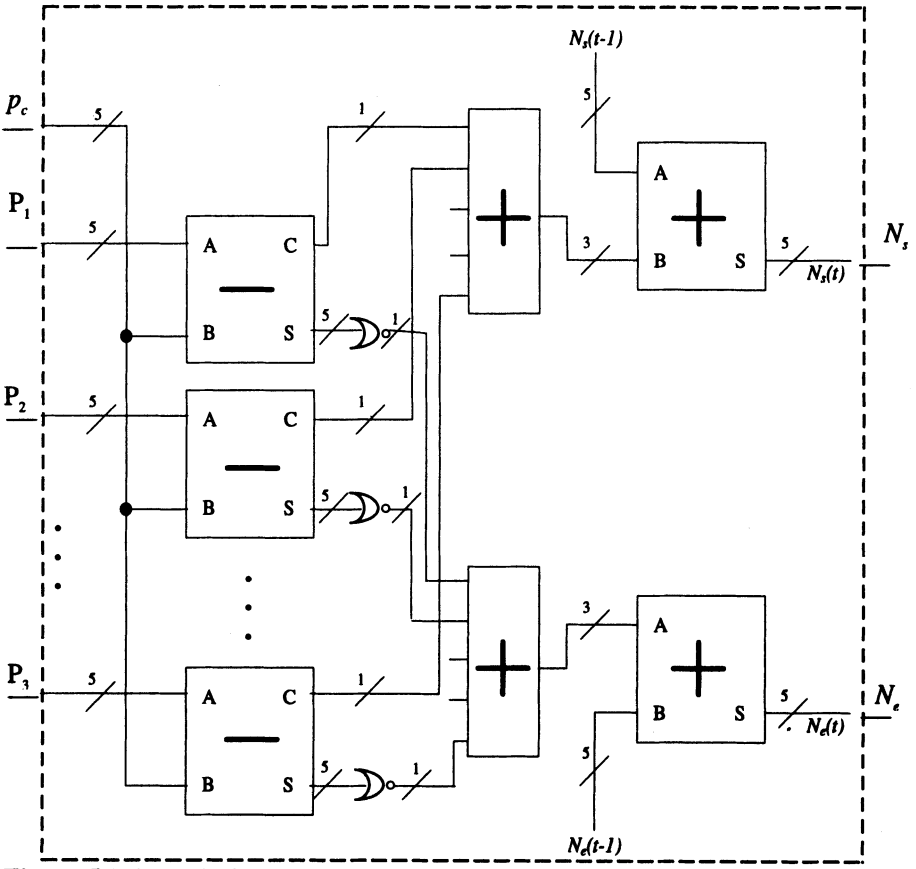


Figure 5 Schematic for determination of parameters

The implementation uses off-chip RAM for the line memory (see table 1). Only the registers required for storing the pixels of one window are on the chip. In each pixel clock cycle, one new window column consisting of 5 pixels is loaded from the off-chip RAM. The pixels are loaded serially pixel by pixel using a system

clock that divides the pixel clock cycle by 5 in order to reduce the number of pins of the circuit is thus reduced.

The second operator is the *local maximum* operator. In each stage, 5 pixel values are compared to the center pixel and the number of pixel values bigger than the center pixel is accumulated across the stages (figure 5). The accumulated value is compared to the threshold value in the first stage of the next pixel clock cycle, increasing the output image latency by the time corresponding to one pixel clock cycle.

The third operator is *pathfinder I*. As there are 44 paths, equation (3) (see figure 6) must be evaluated 9 times in each stage. In the same circuit, the 9 resulting weights are sorted and the 8 highest weights are stored. In the following pipelining stages, these 8 weights are sorted together with 9 new weights. In this manner, the 8 highest weights of all 44 paths are available and the final result can be easily obtained.

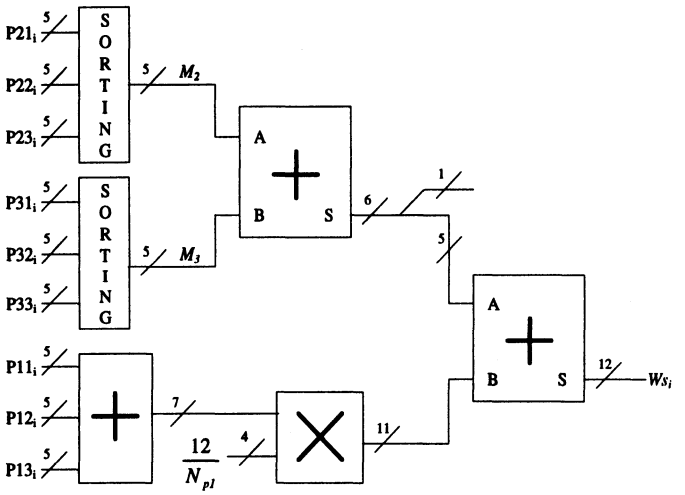


Figure 6 Schematic of equation (3)

The fourth operator, *pathfinder II*, works in a similar way to *pathfinder I*. However, the number of paths weights to calculate and to sort is smaller.

Note that all the four operators can be implemented on the same chip. Two external pins can be used to select the output of the chip in order to program the function that the chip will actually perform. This option allows to use three identical circuits with only 16 pins each to implement all the operations needed for the algorithm.

A transistor count and chip surface estimation is given in table 2.

Table 2 Transistor count and chip area

operator	*number of transistors)*	*chip area (mm^2)*
gradient	5323	0.59
local maximum	1484	0.16
pathfinder I	27252	3.03
pathfinder II	15528	1.73
total	49587	5.51

5 CONCLUSION

The simulation results show that the hardware implementation of the proposed algorithm is capable of producing good results even for images with poor contrast. In both cases a continuous, well defined, one pixel wide contour was extracted by the circuit, which will ease considerably the task of the vertex extraction algorithm, last stage of the polygonal modeling operation.

A data-flow solution for the vertex extraction procedure is rather more complicated than a sequential one. Sequential algorithms progress pixel by pixel along the shape contour, testing each new pixel in order to detect vertex for the polygonal approximation. A list of the boundary points has to be available beforehand, which precludes its use on high-speed real-time operation. In the other hand, data-flow algorithms work on the image as it is being acquired line by line. Therefore, they represent the only possible solution if real-time performance is to be achieved. However, data-flow algorithms look at the image through a rectangular window of a given size, 5×5 or 8×8, for instance. This fact makes it difficult to determine to which shape a contour segment belongs, when more than one object (or shapes) are present in the image scene being modeled. A compromise solution to the above problem may be a hybrid architecture, where vertex detection would be followed by a non data-flow procedure in charge of assigning detected vertex to objects.

The development of an algorithm for VLSI implementation of the complete polygonal modeling procedure is presently under way. Once it is completed and integrated with the circuit described in this paper, a complete high-speed system for polygonal modeling of 2D shapes will be available, capable of performing at the rates required for real-time applications.

6 REFERENCES

[1] I. K. Sethi and N. Ramesh. Local association based recognition of two-dimensional objects. Machine Vision and Applications, 5:265-276, 1992.

[2] F. Stein and G. Medioni. Structural indexing: Efficient 2-d object recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 14(12):1198-1204, 1992.

[3] Y. Q. Zhang and S. Zafar, Motion-compensated wavelet transform coding for color video compression, IEEE Trans. Circuits Syst. Video Technol., 2, 285-296, 1992

[4] T. Y. Tian, M. Shah, Motion estimation and segmentation, Machine Vision and Applications, 9, 32-42, 1996

[5] M. Ghandari, S. De Faria, I. N. Goh, K. T. Tan, Motion compensation for very low bitrate video, Signal Processing: Image Communication, 7, 567-580, 1995

[6] S. Zhang, M. Liang, J. A. Robinson, G. L. Greg, Motion coding of image primitives, Signal Processing: Image Communication, 7, 457-469, 1995

[7]T. Pavlidis. Survey: A review of algorithms for shape analysis. Computer Graphics and Image Processing, 7:243-258, 1978.

[8]N. Ayache and O. D. Faugeras. HYPER: A new approach for the recognition and positioning of two-dimensional objects. IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(1):44-54, 1986.

[9]Y. Kurozumi and W. A. Davis. Polygonal approximation by the minimax method. Computer Graphics and Image Processing, 19:248-264, 1982.

[10]A. D. Marshall and R. R. Martin. Computer Vision, Models and Inspection. World Scientific, London, U. K., 1992.

[11]B. Bhanu, B. L. Hutchings, and K. F. Smith. VLSI design and implementation of a real-time image segmentation processor. Machine Vision and Applications, 3:21-44, 1990.

[12]N. Ranganathan, S. J. Nichani, and R. Mehrotra. A VLSI architecture for half-edge based corner detector. Machine Vision and Applications, 4:165-181, 1991.

[13]H. D. Cheng, C. Tong, and Y. J. Lu. VLSI curve detector. Pattern Recognition, 23:35-50, 1990.

[14]E. Melcher, J. M. Carvalho, M. Barros, L. Naviner, J. F. Naviner et al. A novel gradient operator suited for VLSI implementation of 2D shape recognition. Proceedings of IX SBCCI, Simpósio Brasileiro de Concepção de Circuitos Integrados, pages 321-332, Recife - PE, March 1996.

[15]E. Melcher, J. M. Carvalho, H. Mehrez, N. Vaucher, A. Hoelle, L. Naviner, J. F. Naviner et al. A 2D shape boundary detection algorithm for VLSI implementation. Proceedings of VII SIBGRAPI, Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens, pages 191-196, São Carlos - SP, October 1995

[16]E. Melcher, J. M. Carvalho, P. C. Cortez, L. Naviner, J. F. Naviner. A vertex detection algorithm for VLSI implementation. Proceedings of IX SIBGRAPI, Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens, pages 367-368, Caxambu - MG, October 1996.